kRPC Release

CONTENTS

1	Getti	ng Started 3
	1.1	The Server Plugin
	1.2	The Python Client
	1.3	'Hello World' Script
	1.4	Going further 6
2	Tutor	rials and Examples 7
	2.1	Sub-Orbital Flight
	2.2	Reference Frames
	2.3	Launch into Orbit
	2.4	Pitch, Heading and Roll
	2.5	Interacting with Parts
	2.6	Docking Guidance
3	C#	27
	3.1	C# Client
	3.2	SpaceCenter API
	3.3	InfernalRobotics API
	3.4	Kerbal Alarm Clock API
4	C++	87
	4.1	C++ Client
	4.2	SpaceCenter API
	4.3	InfernalRobotics API
	4.4	Kerbal Alarm Clock API
5	Java	153
J	5.1	Java Client
	5.2	SpaceCenter API
	5.3	InfernalRobotics API
	5.4	Kerbal Alarm Clock API
	T	210
6	Lua 6.1	219 Lua Client
	6.2	SpaceCenter API
	6.3	InfernalRobotics API
	6.4	Kerbal Alarm Clock API
	U. +	Koroai Alaini Ciock Al I
7	Pytho	on 299
	7.1	Python Client
	7.2	SpaceCenter API

	7.3 7.4	InfernalRobotics API					
8	Com	piling kRPC	381				
	8.1	Using an IDE	381				
	8.2	Running the Tests	381				
9	Exte	nding kRPC	383				
	9.1	The kRPC Architecture	383				
	9.2	Service API	383				
	9.3	Documentation	389				
	9.4	Further Examples	389				
	9.5	Generating Service Code for Static Clients	389				
10	Com	munication Protocol	391				
	10.1	Establishing a Connection	391				
	10.2	Remote Procedures	392				
	10.3	Protocol Buffer Encoding	395				
	10.4	Streams	395				
	10.5	KRPC Service	396				
	10.6	Service Description Message	397				
11		mals of kRPC	401				
	11.1	Server Performance Settings	401				
Рy	Python Module Index						
Lu	Lua Module Index						
Index							

kRPC allows you to control Kerbal Space Program from scripts running outside of the game. It has client libraries for many popular languages including Python, C++, C#, Java, Lua. A Ruby client has also been implemented (by TeWu).

- Getting Started Guide
- Tutorials and Examples

The mod exposes most of KSPs API and includes support for Kerbal Alarm Clock and Infernal Robotics. This functionality is provided to client programs via a Remote Procedure Call server, using protocol buffers for serialization. The server component sets up a TCP/IP server that remote scripts can connect to. This communication could be on the local machine only, over a local network, or even over the wider internet if configured correctly. The server is also extensible. Additional remote procedures (grouped into "services") can be added to the server using the "Service API".

CONTENTS 1

2 CONTENTS

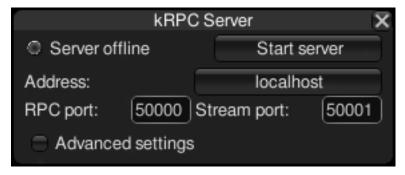
GETTING STARTED

This short guide explains the basics for getting kRPC set up and running, and writing a basic Python script to communicate with the game.

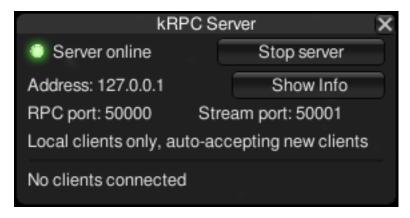
1.1 The Server Plugin

1.1.1 Installation

- 1. Download the kRPC server plugin from one of these locations:
- Github
- SpaceDock
- Curse
- Or the plugin can be obtained via CKAN
- 2. Extract the GameData folder from the archive into your KSP directory.
- 3. Start up KSP and load a save game.
- 4. You should be greeted by the server window:



5. Click "Start server" to, erm... start the server! If all goes well, the light should turn a happy green color:



6. You can hide the window by clicking the close button in the top right, or show/hide the window by clicking on the kRPC icon in the application launcher:



This icon will also turn green when the server is online.

1.1.2 Configuration

The server can be configured using the window displayed in-game. The configuration options are:

- 1. **Address**: this is the IP address that the server will listen on. To only allow connections from the local machine, select 'localhost' (the default). To allow connections over the network, either select the local IP address of your machine, or choose 'Manual' and enter the local IP address manually.
- 2. **RPC and Stream port numbers**: These need to be set to port numbers that are available on your machine. In most cases, they can just be left as the default.

There are also several advanced settings, which are hidden by default, but can be revealed by checking the 'Advanced settings' box:

- 1. Auto-start server: When enabled, the server will start automatically when the game loads.
- 2. **Auto-accept new clients**: When enabled, new client connections are automatically allowed. When disabled, a pop-up is displayed asking whether the new client connection should be allowed.

The other advanced settings control the performance of the server. For details, *see here*.

1.2 The Python Client

Note: kRPC supports both Python 2.7 and Python 3.x.

1.2.1 On Windows

- 1. If you don't already have python installed, download the python installer and run it: https://www.python.org/downloads/windows When running the installer, make sure that pip is installed as well.
- 2. Install the kRPC python module, by opening command prompt and running the following command: $C:\Python27\Scripts\pip.exe install krpc$
- 3. Run Python IDLE (or your favorite editor) and start coding!

1.2.2 On Linux

- 1. Your linux distribution likely already comes with python installed. If not, install python using your favorite package manager, or get it from here: https://www.python.org/downloads
- 2. You also need to install pip, either using your package manager, or from here: https://pypi.python.org/pypi/pip
- 3. Install the kRPC python module by running the following from a terminal: pip install krpc
- 4. Start coding!

1.3 'Hello World' Script

Run KSP and start the server with the default settings. Then execute the following python script:

```
import krpc
conn = krpc.connect(name='Hello World')
vessel = conn.space_center.active_vessel
print(vessel.name)
```

This does the following: line 1 loads the kRPC python module, line 2 opens a new connection to the server, line 3 gets the active vessel and line 4 prints out the name of the vessel. You should see something like the following:



Congratulations! You've written your first script that communicates with KSP.

1.4 Going further...

- For some more interesting examples of what you can do with kRPC, check out the tutorials.
- Client libraries are available for other languages too, including C++, C# and Lua.
- It is also easy to communicate with the server manually from any language you like as long as it can do network I/O. See here for details.

TUTORIALS AND EXAMPLES

This collection of tutorials and example scripts explain how to use the features of kRPC. They are written for the Python client, although the concepts apply to all of the client languages.

2.1 Sub-Orbital Flight

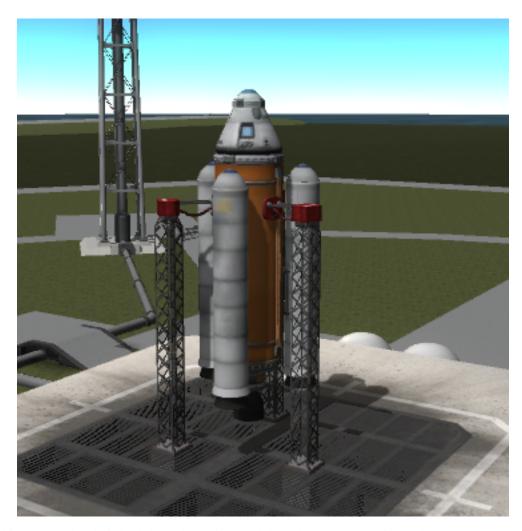
This introductory tutorial uses kRPC to send some Kerbals on a sub-orbital flight, and (hopefully) returns them safely back to Kerbin. It covers the following topics:

- Controlling a rocket (activating stages, setting the throttle)
- Using the auto pilot to point the vessel in a specific direction
- Tracking the amount of resources in the vessel
- Tracking flight and orbital data (such as altitude and apoapsis altitude)

Note: For details on how to write scripts and connect to kRPC, see the Getting Started guide.

2.1.1 Part One: Preparing for Launch

This tutorial uses the two stage rocket pictured below. The craft file for this rocket can be downloaded here and the entire python script for this tutorial from here



The first thing we need to do is load the python client module and open a connection to the server. We can also pass a descriptive name for our script that will appear in the server window in game:

```
import krpc
conn = krpc.connect(name='Sub-orbital flight script')
```

Next we need to get an object representing the active vessel. It's via this object that we will send instructions to the rocket:

```
vessel = conn.space_center.active_vessel
```

We then need to prepare the rocket for launch. The following code sets the throttle to maximum and instructs the auto-pilot to hold a pitch and heading of 90° (vertically upwards). It then waits for 1 second for these settings to take effect.

```
vessel.auto_pilot.target_pitch_and_heading(90,90)
vessel.auto_pilot.engage()
vessel.control.throttle = 1
import time
time.sleep(1)
```

2.1.2 Part Two: Lift-off!

We're now ready to launch by activating the first stage (equivalent to pressing the space bar):

```
print('Launch!')
vessel.control.activate_next_stage()
```

The rocket has a solid fuel stage that will quickly run out, and will need to be jettisoned. We can monitor the amount of solid fuel in the rocket using a while loop that repeatedly checks how much solid fuel there is left in the rocket. When the loop exits, we will activate the next stage to jettison the boosters:

```
while vessel.resources.amount('SolidFuel') > 0.1:
    time.sleep(1)
print('Booster separation')
vessel.control.activate_next_stage()
```

In this bit of code, vessel.resources returns a Resources object that is used to get information about the resources in the rocket.

2.1.3 Part Three: Reaching Apoapsis

Next we will execute a gravity turn when the rocket reaches a sufficiently high altitude. The following loop repeatedly checks the altitude and exits when the rocket reaches 10km:

```
while vessel.flight().mean_altitude < 10000:
    time.sleep(1)</pre>
```

In this bit of code, calling vessel.flight () returns a Flight object that is used to get all sorts of information about the rocket, such as the direction it is pointing in and its velocity.

Now we need to angle the rocket over to a pitch of 60° and maintain a heading of 90° (west). To do this, we simply reconfigure the auto-pilot:

```
print('Gravity turn')
vessel.auto_pilot.target_pitch_and_heading(60,90)
```

Now we wait until the apoapsis reaches 100km, then reduce the throttle to zero, jettison the launch stage and turn off the auto-pilot:

```
while vessel.orbit.apoapsis_altitude < 100000:
    time.sleep(1)
print('Launch stage separation')
vessel.control.throttle = 0
time.sleep(1)
vessel.control.activate_next_stage()
vessel.auto_pilot.disengage()</pre>
```

In this bit of code, vessel.orbit returns an Orbit object that contains all the information about the orbit of the rocket.

2.1.4 Part Four: Returning Safely to Kerbin

Our Kerbals are now heading on a sub-orbital trajectory and are on a collision course with the surface. All that remains to do is wait until they fall to 1km altitude above the surface, and then deploy the parachutes. If you like, you can use time acceleration to skip ahead to just before this happens - the script will continue to work.

```
while vessel.flight().surface_altitude > 1000:
    time.sleep(1)
vessel.control.activate_next_stage()
```

The parachutes should have now been deployed. The next bit of code will repeatedly print out the altitude of the capsule until its speed reaches zero – which will happen when it lands:

```
while vessel.flight(vessel.orbit.body.reference_frame).vertical_speed < -0.1:
    print('Altitude = %.1f meters' % vessel.flight().surface_altitude)
    time.sleep(1)
print('Landed!')</pre>
```

This bit of code uses the <code>vessel.flight()</code> function, as before, but this time it is passed a <code>ReferenceFrame</code> parameter. We want to get the vertical speed of the capsule relative to the surface of Kerbin, so the values returned by the flight object need to be relative to the surface of Kerbin. We therefore pass <code>vessel.orbit.body.reference_frame</code> to <code>vessel.flight()</code> as this reference frame has its origin at the center of Kerbin and it rotates with the planet. For more information, check out the tutorial on <code>ReferenceFrames</code>.

Your Kerbals should now have safely landed back on the surface.

2.2 Reference Frames

- Introduction
 - Origin Position and Axis Orientation
 - * Celestial Body Reference Frame
 - * Vessel Orbital Reference Frame
 - * Vessel Surface Reference Frame
 - Linear Velocity and Angular Velocity
- Available Reference Frames
- Converting Between Reference Frames
- Visual Debugging
- Examples
 - Navball directions
 - Orbital directions
 - Surface 'prograde'
 - Orbital speed
 - Surface speed
 - Angle of attack

2.2.1 Introduction

All of the positions, directions, velocities and rotations in kRPC are relative to something, and *reference frames* define what that something is.

A reference frame specifies:

- The position of the origin at (0, 0, 0),
- the direction of the coordinate axes x, y, and z,
- the linear velocity of the origin (if the reference frame moves)
- and the angular velocity of the coordinate axes (the speed and direction of rotation of the axes).

Note: KSP and kRPC use a left handed coordinate system.

Origin Position and Axis Orientation

The following gives some examples of the position of the origin and the orientation of the coordinate axes for various reference frames.

Celestial Body Reference Frame

The reference frame obtained by calling CelestialBody.reference_frame for Kerbin has the following properties:

- The origin is at the center of Kerbin,
- the y-axis points from the center of Kerbin to the north pole,
- the x-axis points from the center of Kerbin to the intersection of the prime meridian and equator (the surface position at 0° longitude, 0° latitude),
- the z-axis points from the center of Kerbin to the equator at 90°E longitude,
- and the axes rotate with the planet, i.e. the reference frame has the same rotational/angular velocity as Kerbin.

This means that the reference frame is *fixed* relative to Kerbin – it moves with the center of the planet, and also rotates with the planet. Therefore, positions in this reference frame are relative to the center of the planet. Consider the following code prints out the position of the active vessel in Kerbin's reference frame:

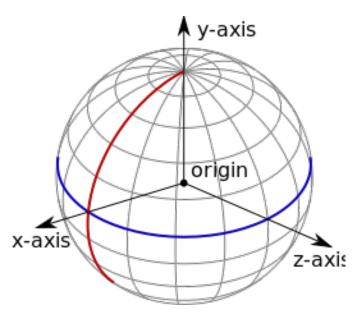


Fig. 2.1: The reference frame for a celestial body, such as Kerbin. The equator is shown in blue, and the prime meridian in red. The black arrows show the coordinate axes, and the origin is at the center of the planet.

```
import krpc
conn = krpc.connect()
vessel = conn.space_center.active_vessel
print(vessel.position(vessel.orbit.body.reference_frame))
```

For a vessel sat on the launchpad, the magnitude of this position vector will be roughly 600,000 meters (equal to the radius of Kerbin). The position vector will also not change over time, because the vessel is sat on the surface of Kerbin and the reference frame also rotates with Kerbin.

Vessel Orbital Reference Frame

2.2. Reference Frames 11

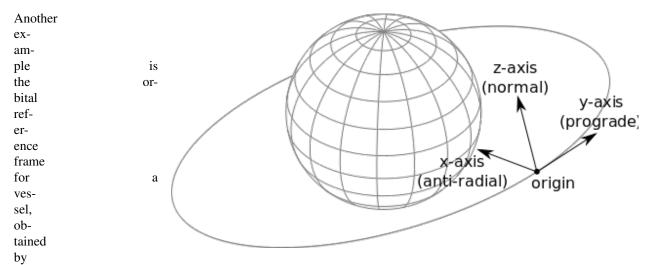


Fig. 2.2: The orbital reference frame for a vessel.

Vessel.orbital_reference_frame. This is fixed to the vessel (the origin moves with the vessel) and it is orientated so that the axes point in the orbital prograde/normal/radial directions.

- The origin is at the center of mass of the vessel,
- the y-axis points in the prograde direction of the vessels orbit,
- the x-axis points in the anti-radial direction of the vessels orbit,
- the z-axis points in the normal direction of the vessels orbit,
- and the axes rotate to match any changes to the prograde/normal/radial directions, for example when the prograde direction changes as the vessel continues on its orbit.

Vessel Surface Reference Frame

Another ex-

ample is

call-

ing

Vessel.reference_frame.

As with the previous example, it is fixed to the vessel (the origin moves with the vessel), however the orientation of the coordinate axes is different. They track the orientation of the vessel:

- The origin is at the center of mass of the vessel.
- the y-axis points in the same direction that the vessel is pointing,

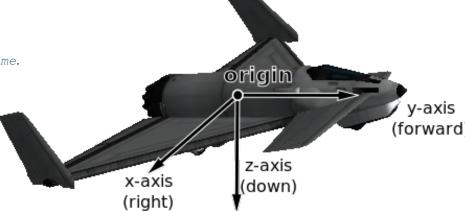


Fig. 2.3: The reference frame for an aircraft.

- the x-axis points out of the right side of the vessel,
- the z-axis points downwards out of the bottom of the vessel,
- and the axes rotate with any changes to the direction of the vessel.

Linear Velocity and Angular Velocity

Reference frames move and rotate relative to one another. For example, the reference frames discussed previously all have their origin position fixed to some object (such as a vessel or a planet). This means that they move and rotate to track the object, and so have a linear and angular velocity associated with them.

For example, the reference frame obtained by calling <code>CelestialBody.reference_frame</code> for Kerbin is fixed relative to Kerbin. This means the angular velocity of the reference frame is identical to Kerbin's angular velocity, and the linear velocity of the reference frame matches the current orbital velocity of Kerbin.



2.2.2 Available Reference Frames

kRPC provides the following reference frames:

- Vessel.reference frame
- Vessel.orbital_reference_frame
- Vessel.surface_reference_frame
- Vessel.surface_velocity_reference_frame
- CelestialBody.reference frame
- CelestialBody.non_rotating_reference_frame
- CelestialBody.orbital_reference_frame
- Node.reference_frame
- Node.orbital_reference_frame
- Part.reference frame
- DockingPort.reference_frame

2.2.3 Converting Between Reference Frames

kRPC provides a utility methods to convert positions, directions, rotations and velocities between the different reference frames:

- SpaceCenter.transform_position()
- SpaceCenter.transform_direction()
- SpaceCenter.transform_rotation()

2.2. Reference Frames

• SpaceCenter.transform_velocity()

2.2.4 Visual Debugging

References frames can be confusing, and choosing the correct one is a challenge in itself. To aid debugging, kRPC provides some methods with which you can draw direction vectors in-game.

SpaceCenter.draw_direction() will draw a direction vector, starting from the center of the active vessel. For example, the following code draws the direction of the current vessels velocity relative to the surface:

```
import krpc
conn = krpc.connect(name='Visual Debugging')
vessel = conn.space_center.active_vessel

ref_frame = vessel.orbit.body.reference_frame
velocity = vessel.flight(ref_frame).velocity
conn.space_center.draw_direction(velocity, ref_frame, (1,0,0))

while True:
pass
```

Note: The client must remain connected, otherwise kRPC will stop drawing the directions, hence the while loop at the end of this example.

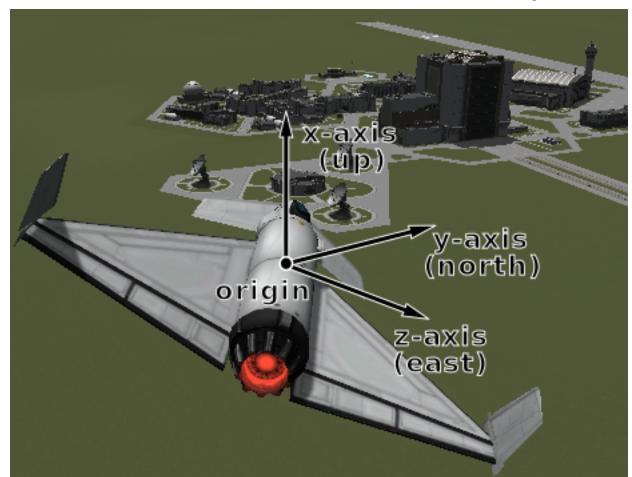
2.2.5 Examples

The following examples demonstrate various uses of reference frames.

Navball directions

This example demonstrates how to make the vessel point in various directions on the navball:

```
import krpc
   conn = krpc.connect(name='Navball directions')
   vessel = conn.space_center.active_vessel
   ap = vessel.auto_pilot
   ap.reference_frame = vessel.surface_reference_frame
   ap.engage()
   # Point the vessel north on the navball, with a pitch of 0 degrees
   ap.target\_direction = (0,1,0)
   ap.wait()
10
11
   # Point the vessel vertically upwards on the navball
12
   ap.target\_direction = (1,0,0)
13
   ap.wait()
   # Point the vessel west (heading of 270 degrees), with a pitch of 0 degrees
16
   ap.target_direction = (0, 0, -1)
17
   ap.wait()
18
   ap.disengage()
```



The code uses the vessel's surface reference frame (Vessel.surface_reference_frame), pictured below:

Line 9 instructs the auto-pilot to point in direction (0, 1, 0) (i.e. along the y-axis) in the vessel's surface reference frame. The y-axis of the reference frame points in the north direction, as required.

Line 13 instructs the auto-pilot to point in direction (1,0,0) (along the x-axis) in the vessel's surface reference frame. This x-axis of the reference frame points upwards (away from the planet) as required.

Line 17 instructs the auto-pilot to point in direction (0, 0, -1) (along the negative z axis). The z-axis of the reference frame points east, so the requested direction points west – as required.

Orbital directions

This example demonstrates how to make the vessel point in the various orbital directions, as seen on the navball when it is in 'orbit' mode. It uses <code>Vessel.orbital_reference_frame</code>.

```
import krpc
conn = krpc.connect(name='Orbital directions')
vessel = conn.space_center.active_vessel
ap = vessel.auto_pilot
ap.reference_frame = vessel.orbital_reference_frame
ap.engage()

# Point the vessel in the prograde direction
ap.target_direction = (0,1,0)
```

2.2. Reference Frames 15

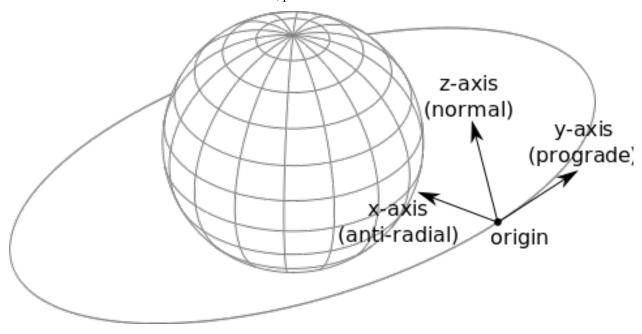
```
ap.wait()

# Point the vessel in the orbit normal direction
ap.target_direction = (0,0,1)
ap.wait()

# Point the vessel in the orbit radial direction
ap.target_direction = (-1,0,0)
ap.wait()

ap.disengage()
```

This code uses the vessel's orbital reference frame, pictured below:



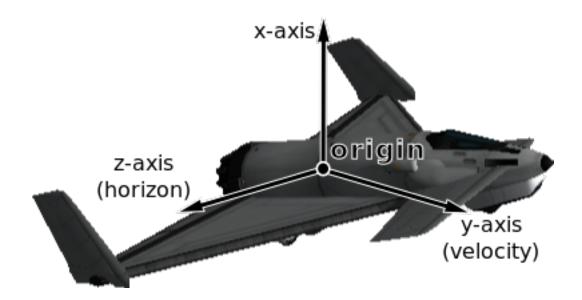
Surface 'prograde'

This example demonstrates how to point the vessel in the 'prograde' direction on the navball, when in 'surface' mode. This is the direction of the vessels velocity relative to the surface:

```
import krpc
conn = krpc.connect(name='Surface prograde')
vessel = conn.space_center.active_vessel
ap = vessel.auto_pilot

ap.reference_frame = vessel.surface_velocity_reference_frame
ap.target_direction = (0,1,0)
ap.engage()
ap.wait()
ap.disengage()
```

This code uses the <code>Vessel.surface_velocity_reference_frame</code>, pictured below:



Orbital speed

To compute the orbital speed of a vessel, you need to get the velocity relative to the planet's *non-rotating* reference frame (*CelestialBody.non_rotating_reference_frame*). This reference frame is fixed relative to the body, but does not rotate:

```
import krpc, time
conn = krpc.connect(name='Orbital speed')
vessel = conn.space_center.active_vessel

while True:

velocity = vessel.flight(vessel.orbit.body.non_rotating_reference_frame).velocity
print('Orbital velocity = (%.1f, %.1f, %.1f)' % velocity)

speed = vessel.flight(vessel.orbit.body.non_rotating_reference_frame).speed
print('Orbital speed = %.1f m/s' % speed)

time.sleep(1)
```

Surface speed

To compute the speed of a vessel relative to the surface of a planet/moon, you need to get the velocity relative to the planets reference frame (CelestialBody.reference_frame). This reference frame rotates with the body, therefore the rotational velocity of the body is taken into account when computing the velocity of the vessel:

```
import krpc, time
conn = krpc.connect(name='Surface speed')
vessel = conn.space_center.active_vessel

while True:

velocity = vessel.flight(vessel.orbit.body.reference_frame).velocity
print('Surface velocity = (%.1f, %.1f, %.1f)' % velocity)

speed = vessel.flight(vessel.orbit.body.reference_frame).speed
```

2.2. Reference Frames 17

```
print('Surface speed = %.1f m/s' % speed)
time.sleep(1)
```



Angle of attack

This example computes the angle between the direction the vessel is pointing in, and the direction that the vessel is moving in (relative to the surface):

```
import krpc, math, time
   conn = krpc.connect(name='Angle of attack')
   vessel = conn.space_center.active_vessel
   while True:
6
       d = vessel.direction(vessel.orbit.body.reference_frame)
       v = vessel.velocity(vessel.orbit.body.reference_frame)
       \# Compute the dot product of d and v
10
11
       dotprod = d[0] *v[0] + d[1] *v[1] + d[2] *v[2]
12
       # Compute the magnitude of v
13
       vmag = math.sqrt(v[0]**2 + v[1]**2 + v[2]**2)
14
       # Note: don't need to magnitude of d as it is a unit vector
15
16
        # Compute the angle between the vectors
       if dotprod == 0:
18
           angle = 0
19
       else:
20
           angle = abs(math.acos (dotprod / vmag) * (180. / math.pi))
21
22
       print('Angle of attack = %.1f' % angle)
23
       time.sleep(1)
```

Note that the orientation of the reference frame used to get the direction and velocity vectors (on lines 7 and 8) does not matter, as the angle between two vectors is the same regardless of the orientation of the axes. However, if we were to use a reference frame that moves with the vessel, line 8 would return (0,0,0). We therefore need a reference frame that is not fixed relative to the vessel. CelestialBody.reference_frame fits these requirements.

2.3 Launch into Orbit

This tutorial launches a two-stage rocket into a 150km circular orbit. The craft file for the rocket can be downloaded here and the entire python script from here.

The following code connects to the server, gets the active vessel, sets up a bunch streams to get flight telemetry then prepares the rocket for launch.

```
import krpc, time, math

turn_start_altitude = 250
turn_end_altitude = 45000
target_altitude = 150000

conn = krpc.connect(name='Launch into orbit')
```

```
vessel = conn.space_center.active_vessel
# Set up streams for telemetry
ut = conn.add_stream(getattr, conn.space_center, 'ut')
altitude = conn.add_stream(getattr, vessel.flight(), 'mean_altitude')
apoapsis = conn.add_stream(getattr, vessel.orbit, 'apoapsis_altitude')
periapsis = conn.add_stream(getattr, vessel.orbit, 'periapsis_altitude')
eccentricity = conn.add_stream(getattr, vessel.orbit, 'eccentricity')
stage_2_resources = vessel.resources_in_decouple_stage(stage=2, cumulative=False)
stage_3_resources = vessel.resources_in_decouple_stage(stage=3, cumulative=False)
srb_fuel = conn.add_stream(stage_3_resources.amount, 'SolidFuel')
launcher_fuel = conn.add_stream(stage_2_resources.amount, 'LiquidFuel')
# Pre-launch setup
vessel.control.sas = False
vessel.control.rcs = False
vessel.control.throttle = 1
# Countdown...
print('3...'); time.sleep(1)
print('2...'); time.sleep(1)
print('1...'); time.sleep(1)
print('Launch!')
```

The next part of the program launches the rocket. The main loop continuously updates the auto-pilot heading to gradually pitch the rocket towards the horizon. It also monitors the amount of solid fuel remaining in the boosters, separating them when they run dry. The loop exits when the rockets apoapsis is close to the target apoapsis.

```
# Activate the first stage
vessel.control.activate_next_stage()
vessel.auto_pilot.engage()
vessel.auto_pilot.target_pitch_and_heading(90, 90)
# Main ascent loop
srbs_separated = False
turn angle = 0
while True:
    # Gravity turn
    if altitude() > turn_start_altitude and altitude() < turn_end_altitude:</pre>
        frac = (altitude() - turn_start_altitude) / (turn_end_altitude - turn_start_altitude)
        new_turn_angle = frac * 90
        if abs(new_turn_angle - turn_angle) > 0.5:
            turn_angle = new_turn_angle
            vessel.auto_pilot.target_pitch_and_heading(90-turn_angle, 90)
    # Separate SRBs when finished
    if not srbs_separated:
        if srb_fuel() < 0.1:
            vessel.control.activate_next_stage()
            srbs_separated = True
            print('SRBs separated')
    # Decrease throttle when approaching target apoapsis
    if apoapsis() > target_altitude*0.9:
        print('Approaching target apoapsis')
        break
```

Next, the program fine tunes the apoapsis, using 10% thrust, then waits until the rocket has left Kerbin's atmosphere.

```
# Disable engines when target apoapsis is reached
vessel.control.throttle = 0.25
while apoapsis() < target_altitude:
    pass
print('Target apoapsis reached')
vessel.control.throttle = 0

# Wait until out of atmosphere
print('Coasting out of atmosphere')
while altitude() < 70500:
    pass</pre>
```

It is now time to plan the circularization burn. First, we calculate the delta-v required to circularize the orbit using the vis-viva equation. We then calculate the burn time needed to achieve this delta-v, using the Tsiolkovsky rocket equation.

```
# Plan circularization burn (using vis-viva equation)
print('Planning circularization burn')
mu = vessel.orbit.body.gravitational_parameter
r = vessel.orbit.apoapsis
a1 = vessel.orbit.semi_major_axis
v1 = math.sqrt(mu*((2./r)-(1./a1)))
v2 = math.sqrt(mu*((2./r)-(1./a2)))
delta_v = v2 - v1
node = vessel.control.add_node(ut() + vessel.orbit.time_to_apoapsis, prograde=delta_v)
# Calculate burn time (using rocket equation)
F = vessel.available_thrust
Isp = vessel.specific_impulse * 9.82
m0 = vessel.mass
m1 = m0 / math.exp(delta_v/Isp)
flow_rate = F / Isp
burn\_time = (m0 - m1) / flow\_rate
```

Next, we need to rotate the craft and wait until the circularization burn. We orientate the ship along the y-axis of the maneuver node's reference frame (i.e. in the direction of the burn) then time warp to 5 seconds before the burn.

```
# Orientate ship
print('Orientating ship for circularization burn')
vessel.auto_pilot.reference_frame = node.reference_frame
vessel.auto_pilot.target_direction = (0,1,0)
vessel.auto_pilot.wait()

# Wait until burn
print('Waiting until circularization burn')
burn_ut = ut() + vessel.orbit.time_to_apoapsis - (burn_time/2.)
lead_time = 5
conn.space_center.warp_to(burn_ut - lead_time)
```

This next part executes the burn. It sets maximum throttle, then throttles down to 5% approximately a tenth of a second before the predicted end of the burn. It then monitors the remaining delta-v until it flips around to point retrograde (at which point the node has been executed).

```
# Execute burn
print('Ready to execute burn')
time_to_apoapsis = conn.add_stream(getattr, vessel.orbit, 'time_to_apoapsis')
while time_to_apoapsis() - (burn_time/2.) > 0:
    pass
```

```
print('Executing burn')
vessel.control.throttle = 1
time.sleep(burn_time - 0.1)
print('Fine tuning')
vessel.control.throttle = 0.05
remaining_burn = conn.add_stream(node.remaining_burn_vector, node.reference_frame)
while remaining_burn()[1] > 0:
    pass
vessel.control.throttle = 0
node.remove()

print('Launch complete')
```

The rocket should now be in a circular 150km orbit above Kerbin.

2.4 Pitch, Heading and Roll

The following example calculates the pitch, heading and rolls angles of the active vessel once per second:

```
import krpc, math, time
conn = krpc.connect(name='Pitch/Heading/Roll')
vessel = conn.space_center.active_vessel
def cross_product(x, y):
    return (x[1]*y[2] - x[2]*y[1], x[2]*y[0] - x[0]*y[2], x[0]*y[1] - x[1]*y[0])
def dot_product(x, y):
    return x[0]*y[0] + x[1]*y[1] + x[2]*y[2]
def magnitude(x):
    return math.sqrt (x[0]**2 + x[1]**2 + x[2]**2)
def angle_between_vectors(x, y):
    """ Compute the angle between vector \boldsymbol{x} and \boldsymbol{y} """
    dp = dot_product(x, y)
   if dp == 0:
       return 0
   xm = magnitude(x)
   ym = magnitude(y)
   return math.acos(dp / (xm*ym)) * (180. / math.pi)
def angle_between_vector_and_plane(x, n):
    """ Compute the angle between a vector {\bf x} and plane with normal vector {\bf n} """
    dp = dot_product(x, n)
   if dp == 0:
        return 0
   xm = magnitude(x)
   nm = magnitude(n)
   return math.asin(dp / (xm*nm)) * (180. / math.pi)
while True:
   vessel_direction = vessel.direction(vessel.surface_reference_frame)
    # Get the direction of the vessel in the horizon plane
    horizon_direction = (0, vessel_direction[1], vessel_direction[2])
```

```
# Compute the pitch - the angle between the vessels direction and the direction in the horizon p.
pitch = angle_between_vectors(vessel_direction, horizon_direction)
if vessel_direction[0] < 0:</pre>
    pitch = -pitch
# Compute the heading - the angle between north and the direction in the horizon plane
north = (0,1,0)
heading = angle_between_vectors(north, horizon_direction)
if horizon_direction[2] < 0:</pre>
    heading = 360 - heading
# Compute the roll
# Compute the plane running through the vessels direction and the upwards direction
up = (1, 0, 0)
plane_normal = cross_product(vessel_direction, up)
# Compute the upwards direction of the vessel
vessel_up = conn.space_center.transform_direction(
    (0,0,-1), vessel.reference_frame, vessel.surface_reference_frame)
# Compute the angle between the upwards direction of the vessel and the plane
roll = angle_between_vector_and_plane(vessel_up, plane_normal)
# Adjust so that the angle is between -180 and 180 and
# rolling right is +ve and left is -ve
if vessel_up[0] > 0:
    roll *= -1
elif roll < 0:</pre>
    roll += 180
else:
    roll -= 180
print('pitch = % 5.1f, heading = % 5.1f, roll = % 5.1f' % (pitch, heading, roll))
time.sleep(1)
```

2.5 Interacting with Parts

The following examples demonstrate use of the *Parts* functionality to achieve various tasks. More details on specific topics can also be found in the API documentation:

- · Trees of Parts
- Attachment Modes
- Fuel Lines
- Staging

2.5.1 Deploying all Parachutes

Sometimes things go horribly wrong. The following script does its best to save your Kerbals by deploying all the parachutes:

```
import krpc
conn = krpc.connect()
vessel = conn.space_center.active_vessel
```

```
for parachute in vessel.parts.parachutes:
   parachute.deploy()
```

2.5.2 'Control From Here' for Docking Ports

The following example will find a standard sized Clamp-O-Tron docking port, and control the vessel from it:

```
import krpc
conn = krpc.connect()
vessel = conn.space_center.active_vessel

ports = vessel.parts.docking_ports
port = list(filter(lambda p: p.part.title == 'Clamp-O-Tron Docking Port', ports))[0]
part = port.part
vessel.parts.controlling = part
```

2.5.3 Combined Specific Impulse

The following script calculates the combined specific impulse of all currently active and fueled engines on a rocket. See here for a description of the maths: http://wiki.kerbalspaceprogram.com/wiki/Specific_impulse#Multiple_engines

```
import krpc
conn = krpc.connect()
vessel = conn.space_center.active_vessel

active_engines = filter(lambda e: e.active and e.has_fuel, vessel.parts.engines)

print('Active engines:')
for engine in active_engines:
    print(' %s in stage %d' % (engine.part.title, engine.part.stage))

thrust = sum(engine.thrust for engine in active_engines)
fuel_consumption = sum(engine.thrust / engine.specific_impulse for engine in active_engines)
isp = thrust / fuel_consumption

print('Combined vaccuum Isp = %d seconds' % isp)
```

2.6 Docking Guidance

The following script outputs docking guidance information. It waits until the vessel is being controlled from a docking port, and a docking port is set as the current target. It then prints out information about speeds and distances relative to the docking axis.

It uses numpy to do linear algebra on the vectors returned by kRPC – for example computing the dot product or length of a vector – and uses curses for terminal output.

```
import krpc, curses, time, sys
import numpy as np
import numpy.linalg as la

# Set up curses
stdscr = curses.initscr()
curses.nocbreak()
```

```
stdscr.keypad(1)
curses.noecho()
try:
    # Connect to kRPC
   conn = krpc.connect(name='Docking Guidance')
   vessel = conn.space_center.active_vessel
   current = None
   target = None
   while True:
        stdscr.clear()
        stdscr.addstr(0,0,'-- Docking Guidance --')
        current = conn.space_center.active_vessel.parts.controlling.docking_port
        target = conn.space_center.target_docking_port
        if current is None:
            stdscr.addstr(2,0,'Awaiting control from docking port...')
        elif target is None:
            stdscr.addstr(2,0,'Awaiting target docking port...')
            # Get positions, distances, velocities and speeds relative to the target docking port
            current_position = current.position(target.reference_frame)
            velocity = current.part.velocity(target.reference_frame)
            displacement = np.array(current_position)
            distance = la.norm(displacement)
            speed = la.norm(np.array(velocity))
            # Get speeds and distances relative to the docking axis
            # (the direction the target docking port is facing in)
            # Axial = along the docking axis
            axial_displacement = np.copy(displacement)
            axial\_displacement[0] = 0
            axial\_displacement[2] = 0
            axial_distance = axial_displacement[1]
            axial_velocity = np.copy(velocity)
            axial\_velocity[0] = 0
            axial\_velocity[2] = 0
            axial_speed = axial_velocity[1]
            if axial_distance > 0:
                axial\_speed *= -1
            # Radial = perpendicular to the docking axis
            radial_displacement = np.copy(displacement)
            radial_displacement[1] = 0
            radial_distance = la.norm(radial_displacement)
            radial_velocity = np.copy(velocity)
            radial\_velocity[1] = 0
            radial_speed = la.norm(radial_velocity)
            if np.dot(radial_velocity, radial_displacement) > 0:
                radial_speed \star = -1
```

```
# Get the docking port state
                              if current.state == conn.space_center.DockingPortState.ready:
                                         state = 'Ready to dock'
                               elif current.state == conn.space_center.DockingPortState.docked:
                                         state = 'Docked'
                               elif current.state == conn.space_center.DockingPortState.docking:
                                         state = 'Docking...'
                               else:
                                         state = 'Unknown'
                               # Output information
                               stdscr.addstr(2,0,'Current ship: {:30}'.format(current.part.vessel.name[:30]))
                               stdscr.addstr(3,0,'Current port: {:30}'.format(current.part.title[:30]))
                               stdscr.addstr(5,0,'Target ship: {:30}'.format(target.part.vessel.name[:30]))
                               stdscr.addstr(6,0,'Target port: {:30}'.format(target.part.title[:30]))
                               stdscr.addstr(8,0,'Status: {:10}'.format(state))
                               stdscr.addstr(10, 0, '
                               stdscr.addstr(11, 0, '
                                                                                                              | Distance | Speed |')
                               stdscr.addstr(12, 0, '+---
                                                                                                             stdscr.addstr(13, 0, '| | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m/s | {:>+6.2f} m/s |'.format distance, spentile stdscr.addstr(14, 0, '| Axial | {:>+6.2f} m/s | {:>+6.2f} m/s |'.format distance, spentile s
                               stdscr.addstr(15, 0, '| Radial | {:>+6.2f} m | {:>+6.2f} m/s |'.format radial_distar
                               stdscr.addstr(16, 0, '+------')
                    stdscr.refresh()
                    time.sleep(0.25)
finally:
       # Shutdown curses
         curses.nocbreak()
          stdscr.keypad(0)
          curses.echo()
         curses.endwin()
```

CHAPTER

THREE

C#

3.1 C# Client

The KRPC.Client.dll library provides functionality to interact with a kRPC server from C#.

3.1.1 Installing the Client

KRPC.Client.dll is available on NuGet. You also need to get Google.Protobuf.dll, which is also available on NuGet.

Note: The copy of Google.Protobuf.dll in the GameData folder shipped with kRPC should be *avoided*. It is a modified version to work within KSP. See here for more details.

3.1.2 Connecting to the Server

To connect to a server, create a KRPC.Client.Connection object. For example to connect to a server running on the local machine:

```
using KRPC.Client.Services.KRPC;

class Program
{
    public static void Main ()
    {
        var connection = new KRPC.Client.Connection (name : "Example");
        System.Console.WriteLine (connection.KRPC ().GetStatus ().Version);
    }
}
```

The class constructor also accepts arguments that specify what address and port numbers to connect to. For example:

```
using KRPC.Client.Services.KRPC;

class Program
{
    public static void Main ()
    {
        var connection = new KRPC.Client.Connection (name : "Example", address: "my.domain.name", rpc.
        System.Console.WriteLine (connection.KRPC ().GetStatus ().Version);
```

```
}
}
```

3.1.3 Interacting with the Server

Interaction with the server is performed via a connection object. Functionality for services are defined in the namespaces KRPC.Client.Services.*.

Before a service can be used it must first be instantiated. The following example connects to the server, instantiates the SpaceCenter service, and outputs the name of the active vessel:

```
using KRPC.Client.Services.SpaceCenter;

class Program
{
    public static void Main ()
    {
        var connection = new KRPC.Client.Connection (name : "Vessel Name");
        var sc = connection.SpaceCenter ();
        var vessel = sc.ActiveVessel;
        System.Console.WriteLine (vessel.Name);
    }
}
```

3.1.4 Streaming Data from the Server

A stream repeatedly executes a function on the server, with a fixed set of argument values. It provides a more efficient way of repeatedly getting the result of calling function on the server, without having to invoke it directly – which incurs communication overheads.

For example, consider the following loop that continuously prints out the position of the active vessel. This loop incurs significant communication overheads, as the <code>Vessel.Position</code> method is called repeatedly.

The following code achieves the same thing, but is far more efficient. It makes a single call to Connection. AddStream to create the stream, which avoids the communication overhead in the previous example.

Streams are created by calling Connection. AddStream and passing it a lambda expression. It returns an instance of the KRPC.Client.Stream class from which the latest value can be obtained by calling KRPC.Client.Stream.Get.

The lambda expression passed to Connection. AddStream must take zero arguments and be either a method call expression or a parameter call expression.

28 Chapter 3. C#

3.2 SpaceCenter API

3.2.1 SpaceCenter

class SpaceCenter

Provides functionality to interact with Kerbal Space Program. This includes controlling the active vessel, managing its resources, planning maneuver nodes and auto-piloting.

Vessel ActiveVessel { get; set; }

The currently active vessel.

IList<Vessel> Vessels { get; }

A list of all the vessels in the game.

IDictionary<string, CelestialBody> Bodies { get; }

A dictionary of all celestial bodies (planets, moons, etc.) in the game, keyed by the name of the body.

CelestialBody TargetBody { get; set; }

The currently targeted celestial body.

Vessel TargetVessel { get; set; }

The currently targeted vessel.

DockingPort TargetDockingPort { get; set; }

The currently targeted docking port.

void ClearTarget ()

Clears the current target.

void LaunchVesselFromVAB (string name)

Launch a new vessel from the VAB onto the launchpad.

Parameters

• name – Name of the vessel's craft file.

void LaunchVesselFromSPH (string name)

Launch a new vessel from the SPH onto the runway.

Parameters

• name – Name of the vessel's craft file.

double UT { get; }

The current universal time in seconds.

float G { get; }

The value of the gravitational constant G in $N(m/kg)^2$.

WarpMode WarpMode { get; }

The current time warp mode. Returns <code>WarpMode.None</code> if time warp is not active, <code>WarpMode.Rails</code> if regular "on-rails" time warp is active, or <code>WarpMode.Physics</code> if physical time warp is active.

float WarpRate { get; }

The current warp rate. This is the rate at which time is passing for either on-rails or physical time warp. For example, a value of 10 means time is passing 10x faster than normal. Returns 1 if time warp is not active.

float WarpFactor { get; }

The current warp factor. This is the index of the rate at which time is passing for either regular "on-rails" or physical time warp. Returns 0 if time warp is not active. When in on-rails time warp, this is equal to RailsWarpFactor, and in physics time warp, this is equal to PhysicsWarpFactor.

int RailsWarpFactor { get; set; }

The time warp rate, using regular "on-rails" time warp. A value between 0 and 7 inclusive. 0 means no time warp. Returns 0 if physical time warp is active. If requested time warp factor cannot be set, it will be set to the next lowest possible value. For example, if the vessel is too close to a planet. See the KSP wiki for details.

int PhysicsWarpFactor { get; set; }

The physical time warp rate. A value between 0 and 3 inclusive. 0 means no time warp. Returns 0 if regular "on-rails" time warp is active.

bool CanRailsWarpAt (int factor = 1)

Returns true if regular "on-rails" time warp can be used, at the specified warp *factor*. The maximum time warp rate is limited by various things, including how close the active vessel is to a planet. See the KSP wiki for details.

Parameters

• factor – The warp factor to check.

int MaximumRailsWarpFactor { get; }

The current maximum regular "on-rails" warp factor that can be set. A value between 0 and 7 inclusive. See the KSP wiki for details.

void WarpTo (double UT, float maxRailsRate = 100000.0, float maxPhysicsRate = 2.0)

Uses time acceleration to warp forward to a time in the future, specified by universal time *UT*. This call blocks until the desired time is reached. Uses regular "on-rails" or physical time warp as appropriate. For example, physical time warp is used when the active vessel is traveling through an atmosphere. When using regular "on-rails" time warp, the warp rate is limited by *maxRailsRate*, and when using physical time warp, the warp rate is limited by *maxPhysicsRate*.

Parameters

- **UT** The universal time to warp to, in seconds.
- maxRailsRate The maximum warp rate in regular "on-rails" time warp.
- maxPhysicsRate The maximum warp rate in physical time warp.

Returns When the time warp is complete.

Tuple<double, double, double> **TransformPosition** (Tuple<double, double, double> position, ReferenceFrame to)

Converts a position vector from one reference frame to another.

Parameters

- **position** Position vector in reference frame *from*.
- **from** The reference frame that the position vector is in.
- to The reference frame to covert the position vector to.

Returns The corresponding position vector in reference frame to.

Tuple<double, double, double> **TransformDirection** (Tuple<double, double, double> *direction*, ReferenceFrame from, ReferenceFrame to)

Converts a direction vector from one reference frame to another.

Parameters

- **direction** Direction vector in reference frame *from*.
- **from** The reference frame that the direction vector is in.
- to The reference frame to covert the direction vector to.

30 Chapter 3. C#

Returns The corresponding direction vector in reference frame to.

Tuple<double, double, rotation, ReferenceFrame from, ReferenceFrame to)

Converts a rotation from one reference frame to another.

Parameters

- **rotation** Rotation in reference frame *from*.
- from The reference frame that the rotation is in.
- to The corresponding rotation in reference frame to.

Returns The corresponding rotation in reference frame to.

Tuple<double, double, double> **TransformVelocity** (Tuple<double, double, double> *position*, Tuple<double, double, double> *velocity*, *ReferenceFrame from*, *ReferenceFrame to*)

Converts a velocity vector (acting at the specified position vector) from one reference frame to another. The position vector is required to take the relative angular velocity of the reference frames into account.

Parameters

- **position** Position vector in reference frame *from*.
- **velocity** Velocity vector in reference frame *from*.
- from The reference frame that the position and velocity vectors are in.
- to The reference frame to covert the velocity vector to.

Returns The corresponding velocity in reference frame *to*.

bool FARAvailable { get; }

Whether Ferram Aerospace Research is installed.

bool RemoteTechAvailable { get; }

Whether RemoteTech is installed.

void **DrawDirection** (Tuple<double, double, double> direction, ReferenceFrame referenceFrame, Tuple<double, double, double> color, float length = 10.0)

Draw a direction vector on the active vessel.

Parameters

- **direction** Direction to draw the line in.
- referenceFrame Reference frame that the direction is in.
- color The color to use for the line, as an RGB color.
- **length** The length of the line. Defaults to 10.

void DrawLine (Tuple<double, double, double> start, Tuple<double, double, double> end, Reference-Frame referenceFrame, Tuple<double, double, double> color)
Draw a line.

Parameters

- **start** Position of the start of the line.
- end Position of the end of the line.
- referenceFrame Reference frame that the position are in.
- color The color to use for the line, as an RGB color.

void ClearDrawing ()

Remove all directions and lines currently being drawn.

enum WarpMode

Returned by WarpMode

Rails

Time warp is active, and in regular "on-rails" mode.

Physics

Time warp is active, and in physical time warp mode.

None

Time warp is not active.

3.2.2 Vessel

class Vessel

These objects are used to interact with vessels in KSP. This includes getting orbital and flight data, manipulating control inputs and managing resources.

string Name { get; set; }

The name of the vessel.

VesselType Type { get; set; }

The type of the vessel.

VesselSituation Situation { get; }

The situation the vessel is in.

double MET { get; }

The mission elapsed time in seconds.

Flight Flight (ReferenceFrame referenceFrame = None)

Returns a Flight object that can be used to get flight telemetry for the vessel, in the specified reference frame.

Parameters

• **referenceFrame** – Reference frame. Defaults to the vessel's surface reference frame (Vessel.SurfaceReferenceFrame).

Vessel Target { get; set; }

The target vessel. null if there is no target. When setting the target, the target cannot be the current vessel.

Orbit Orbit { get; }

The current orbit of the vessel.

Control Control { get; }

Returns a *Control* object that can be used to manipulate the vessel's control inputs. For example, its pitch/yaw/roll controls, RCS and thrust.

AutoPilot AutoPilot { get; }

An AutoPilot object, that can be used to perform simple auto-piloting of the vessel.

Resources Resources { get; }

A Resources object, that can used to get information about resources stored in the vessel.

Resources ResourcesInDecoupleStage (int stage, bool cumulative = True)

Returns a Resources object, that can used to get information about resources stored in a given stage.

32 Chapter 3. C#

Parameters

- stage Get resources for parts that are decoupled in this stage.
- **cumulative** When false, returns the resources for parts decoupled in just the given stage. When true returns the resources decoupled in the given stage and all subsequent stages combined.

Note: For details on stage numbering, see the discussion on *Staging*.

Parts Parts { get; }

A Parts object, that can used to interact with the parts that make up this vessel.

Comms Comms { get; }

A Comms object, that can used to interact with RemoteTech for this vessel.

Note: Requires RemoteTech to be installed.

float Mass { get; }

The total mass of the vessel, including resources, in kg.

float DryMass { get; }

The total mass of the vessel, excluding resources, in kg.

float Thrust { get; }

The total thrust currently being produced by the vessel's engines, in Newtons. This is computed by summing *Engine.Thrust* for every engine in the vessel.

float AvailableThrust { get; }

Gets the total available thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine.AvailableThrust</code> for every active engine in the vessel.

float MaxThrust { get; }

The total maximum thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine.MaxThrust</code> for every active engine.

float MaxVacuumThrust { get; }

The total maximum thrust that can be produced by the vessel's active engines when the vessel is in a vacuum, in Newtons. This is computed by summing <code>Engine.MaxVacuumThrust</code> for every active engine.

float SpecificImpulse { get; }

The combined specific impulse of all active engines, in seconds. This is computed using the formula described here.

float VacuumSpecificImpulse { get; }

The combined vacuum specific impulse of all active engines, in seconds. This is computed using the formula described here.

float KerbinSeaLevelSpecificImpulse { get; }

The combined specific impulse of all active engines at sea level on Kerbin, in seconds. This is computed using the formula described here.

ReferenceFrame ReferenceFrame { get; }

The reference frame that is fixed relative to the vessel, and orientated with the vessel.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel.

- •The x-axis points out to the right of the vessel.
- •The y-axis points in the forward direction of the vessel.
- •The z-axis points out of the bottom off the vessel.

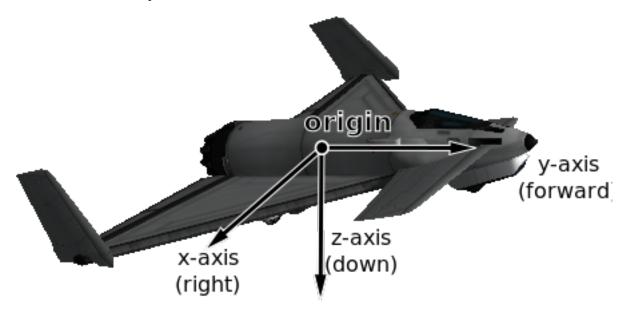


Fig. 3.1: Vessel reference frame origin and axes for the Aeris 3A aircraft

ReferenceFrame OrbitalReferenceFrame { get; }

The reference frame that is fixed relative to the vessel, and orientated with the vessels orbital prograde/normal/radial directions.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Note: Be careful not to confuse this with 'orbit' mode on the navball.

ReferenceFrame SurfaceReferenceFrame { get; }

The reference frame that is fixed relative to the vessel, and orientated with the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the north and up directions on the surface of the body.
- •The x-axis points in the zenith direction (upwards, normal to the body being orbited, from the center of the body towards the center of mass of the vessel).
- •The y-axis points northwards towards the astronomical horizon (north, and tangential to the surface of the body the direction in which a compass would point when on the surface).
- •The z-axis points eastwards towards the astronomical horizon (east, and tangential to the surface of the body east on a compass when on the surface).

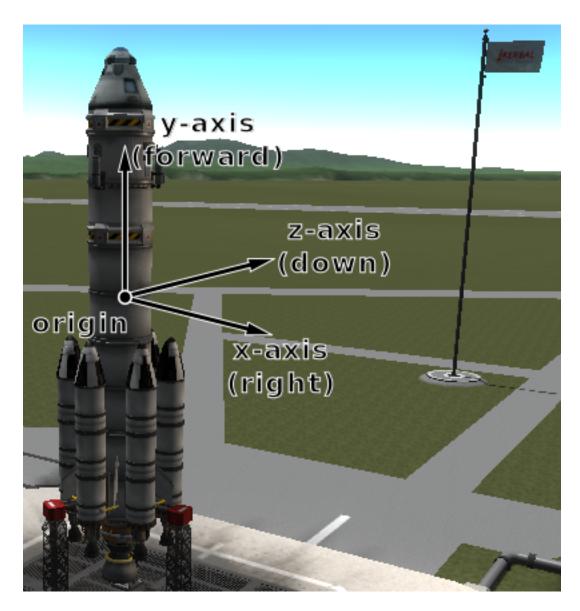


Fig. 3.2: Vessel reference frame origin and axes for the Kerbal-X rocket

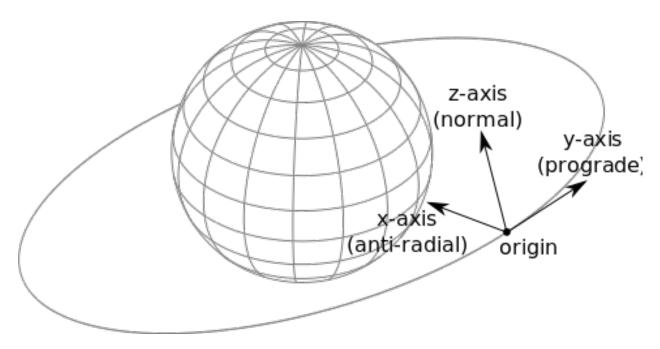


Fig. 3.3: Vessel orbital reference frame origin and axes

Note: Be careful not to confuse this with 'surface' mode on the navball.

ReferenceFrame SurfaceVelocityReferenceFrame { get; }

The reference frame that is fixed relative to the vessel, and orientated with the velocity vector of the vessel relative to the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel's velocity vector.
- •The y-axis points in the direction of the vessel's velocity vector, relative to the surface of the body being orbited.
- •The z-axis is in the plane of the astronomical horizon.
- •The x-axis is orthogonal to the other two axes.

Tuple<double, double, double> **Position** (*ReferenceFrame referenceFrame*)

Returns the position vector of the center of mass of the vessel in the given reference frame.

Parameters

Tuple<double, double> **Velocity** (*ReferenceFrame referenceFrame*)

Returns the velocity vector of the center of mass of the vessel in the given reference frame.

Parameters

Tuple<double, double, double> Rotation (ReferenceFrame referenceFrame)

Returns the rotation of the center of mass of the vessel in the given reference frame.

Parameters

Tuple<double, double> **Direction** (*ReferenceFrame referenceFrame*)

Returns the direction in which the vessel is pointing, as a unit vector, in the given reference frame.

Parameters

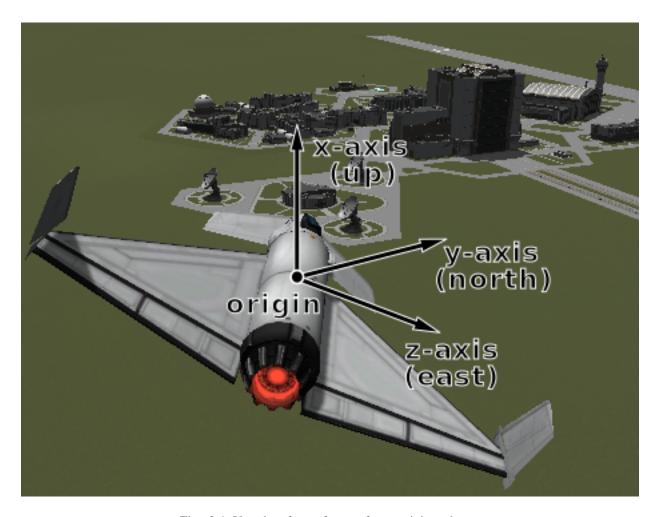


Fig. 3.4: Vessel surface reference frame origin and axes

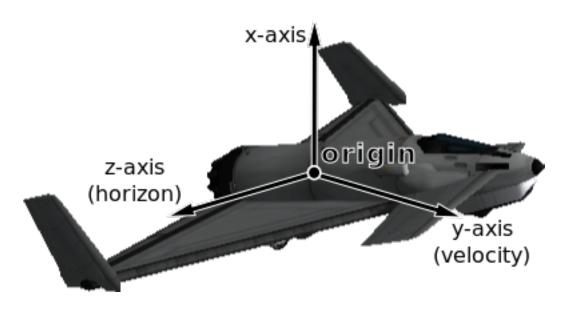


Fig. 3.5: Vessel surface velocity reference frame origin and axes

Tuple<double, double, double> AngularVelocity (ReferenceFrame referenceFrame)

Returns the angular velocity of the vessel in the given reference frame. The magnitude of the returned vector is the rotational speed in radians per second, and the direction of the vector indicates the axis of rotation (using the right hand rule).

Parameters

enum VesselType

See Vessel. Type.

Ship

Ship.

Station

Station.

Lander

Lander.

Probe

Probe.

Rover

Rover.

Base

Base.

Debris

Debris.

enum VesselSituation

See Vessel. Situation.

Docked

Vessel is docked to another.

Escaping

Escaping.

Flying

Vessel is flying through an atmosphere.

Landed

Vessel is landed on the surface of a body.

Orbiting

Vessel is orbiting a body.

PreLaunch

Vessel is awaiting launch.

Splashed

Vessel has splashed down in an ocean.

SubOrbital

Vessel is on a sub-orbital trajectory.

3.2.3 CelestialBody

class CelestialBody

Represents a celestial body (such as a planet or moon).

string Name { get; }

The name of the body.

IList<CelestialBody> Satellites { get; }

A list of celestial bodies that are in orbit around this celestial body.

Orbit Orbit { get; }

The orbit of the body.

float Mass { get; }

The mass of the body, in kilograms.

float GravitationalParameter { get; }

The standard gravitational parameter of the body in m^3s^{-2} .

float SurfaceGravity { get; }

The acceleration due to gravity at sea level (mean altitude) on the body, in m/s^2 .

float RotationalPeriod { get; }

The sidereal rotational period of the body, in seconds.

float RotationalSpeed { get; }

The rotational speed of the body, in radians per second.

float EquatorialRadius { get; }

The equatorial radius of the body, in meters.

double SurfaceHeight (double latitude, double longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water this is equal to 0.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees

double BedrockHeight (double latitude, double longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water, this is the height of the sea-bed and is therefore a negative value.

Parameters

- **latitude** Latitude in degrees
- **longitude** Longitude in degrees

Tuple<double, double, double> MSLPosition (double latitude, double longitude, ReferenceFrame referenceFrame)

The position at mean sea level at the given latitude and longitude, in the given reference frame.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees
- referenceFrame Reference frame for the returned position vector

Tuple<double, double, double> SurfacePosition (double latitude, double longitude, Reference-Frame referenceFrame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position of the surface of the water.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees
- referenceFrame Reference frame for the returned position vector

Tuple<double, double, double> **BedrockPosition** (double *latitude*, double *longitude*, *Reference-Frame*)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position at the bottom of the sea-bed.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees
- referenceFrame Reference frame for the returned position vector

float SphereOfInfluence { get; }

The radius of the sphere of influence of the body, in meters.

bool HasAtmosphere { get; }

true if the body has an atmosphere.

float AtmosphereDepth { get; }

The depth of the atmosphere, in meters.

bool HasAtmosphericOxygen { get; }

true if there is oxygen in the atmosphere, required for air-breathing engines.

ReferenceFrame ReferenceFrame { get; }

The reference frame that is fixed relative to the celestial body.

- •The origin is at the center of the body.
- •The axes rotate with the body.
- •The x-axis points from the center of the body towards the intersection of the prime meridian and equator (the position at 0° longitude, 0° latitude).
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points from the center of the body towards the equator at 90°E longitude.

ReferenceFrame NonRotatingReferenceFrame { get; }

The reference frame that is fixed relative to this celestial body, and orientated in a fixed direction (it does not rotate with the body).

- •The origin is at the center of the body.
- •The axes do not rotate.
- •The x-axis points in an arbitrary direction through the equator.
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points in an arbitrary direction through the equator.

ReferenceFrame OrbitalReferenceFrame { get; }

Gets the reference frame that is fixed relative to this celestial body, but orientated with the body's orbital prograde/normal/radial directions.

- •The origin is at the center of the body.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.

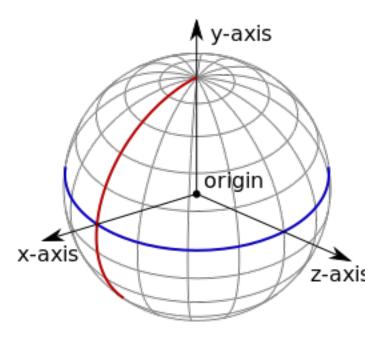


Fig. 3.6: Celestial body reference frame origin and axes. The equator is shown in blue, and the prime meridian in red.

- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Tuple<double, double> Position (ReferenceFrame referenceFrame)

Returns the position vector of the center of the body in the specified reference frame.

Parameters

Tuple<double, double, double> **Velocity** (*ReferenceFrame referenceFrame*)

Returns the velocity vector of the body in the specified reference frame.

Parameters

Tuple<double, double, double> Rotation (ReferenceFrame referenceFrame)

Returns the rotation of the body in the specified reference frame.

Parameters

Tuple<double, double> **Direction** (*ReferenceFrame referenceFrame*)

Returns the direction in which the north pole of the celestial body is pointing, as a unit vector, in the specified reference frame.

Parameters

Tuple < double, double > Angular Velocity (Reference Frame reference Frame)

Returns the angular velocity of the body in the specified reference frame. The magnitude of the vector is the rotational speed of the body, in radians per second, and the direction of the vector indicates the axis of rotation, using the right-hand rule.

Parameters

3.2.4 Flight

class Flight

Used to get flight telemetry for a vessel, by calling Vessel.Flight. All of the information returned by this

class is given in the reference frame passed to that method.

Note: To get orbital information, such as the apoapsis or inclination, see Orbit.

float GForce { get; }

The current G force acting on the vessel in m/s^2 .

double MeanAltitude { get; }

The altitude above sea level, in meters.

double SurfaceAltitude { get; }

The altitude above the surface of the body or sea level, whichever is closer, in meters.

double BedrockAltitude { get; }

The altitude above the surface of the body, in meters. When over water, this is the altitude above the sea floor.

double Elevation { get; }

The elevation of the terrain under the vessel, in meters. This is the height of the terrain above sea level, and is negative when the vessel is over the sea.

double Latitude { get; }

The latitude of the vessel for the body being orbited, in degrees.

double Longitude { get; }

The longitude of the vessel for the body being orbited, in degrees.

Tuple<double, double> Velocity { get; }

The velocity vector of the vessel. The magnitude of the vector is the speed of the vessel in meters per second. The direction of the vector is the direction of the vessels motion.

double Speed { get; }

The speed of the vessel in meters per second.

double HorizontalSpeed { get; }

The horizontal speed of the vessel in meters per second.

double VerticalSpeed { get; }

The vertical speed of the vessel in meters per second.

Tuple<double, double, double> CenterOfMass { get; }

The position of the center of mass of the vessel.

Tuple<double, double, double> Rotation { get; }

The rotation of the vessel.

Tuple<double, double> Direction { get; }

The direction vector that the vessel is pointing in.

float Pitch { get; }

The pitch angle of the vessel relative to the horizon, in degrees. A value between -90° and +90°.

float Heading { get; }

The heading angle of the vessel relative to north, in degrees. A value between 0° and 360°.

float Roll { get; }

The roll angle of the vessel relative to the horizon, in degrees. A value between -180° and +180°.

Tuple<double, double, double> Prograde { get; }

The unit direction vector pointing in the prograde direction.

Tuple<double, double> Retrograde { get; }

The unit direction vector pointing in the retrograde direction.

Tuple<double, double, double> Normal { get; }

The unit direction vector pointing in the normal direction.

Tuple<double, double> AntiNormal { get; }

The unit direction vector pointing in the anti-normal direction.

Tuple<double, double, double> Radial { get; }

The unit direction vector pointing in the radial direction.

Tuple<double, double> AntiRadial { get; }

The unit direction vector pointing in the anti-radial direction.

float AtmosphereDensity { get; }

The current density of the atmosphere around the vessel, in kg/m^3 .

float DynamicPressure { get; }

The dynamic pressure acting on the vessel, in Pascals. This is a measure of the strength of the aerodynamic forces. It is equal to $\frac{1}{2}$ air density velocity². It is commonly denoted as Q.

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

float StaticPressure { get; }

The static atmospheric pressure acting on the vessel, in Pascals.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

Tuple<double, double, double> AerodynamicForce { get; }

The total aerodynamic forces acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

Tuple<double, double, double> Lift { get; }

The aerodynamic lift currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

Tuple<double, double, double> Drag { get; }

The aerodynamic drag currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

float SpeedOfSound { get; }

The speed of sound, in the atmosphere around the vessel, in m/s.

Note: Not available when Ferram Aerospace Research is installed.

float Mach { get; }

The speed of the vessel, in multiples of the speed of sound.

Note: Not available when Ferram Aerospace Research is installed.

float EquivalentAirSpeed { get; }

The equivalent air speed of the vessel, in m/s.

Note: Not available when Ferram Aerospace Research is installed.

float TerminalVelocity { get; }

An estimate of the current terminal velocity of the vessel, in m/s. This is the speed at which the drag forces cancel out the force of gravity.

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

float AngleOfAttack { get; }

Gets the pitch angle between the orientation of the vessel and its velocity vector, in degrees.

float SideslipAngle { get; }

Gets the yaw angle between the orientation of the vessel and its velocity vector, in degrees.

float TotalAirTemperature { get; }

The total air temperature of the atmosphere around the vessel, in Kelvin. This temperature includes the Flight.StaticAirTemperature and the vessel's kinetic energy.

float StaticAirTemperature { get; }

The static (ambient) temperature of the atmosphere around the vessel, in Kelvin.

float StallFraction { get; }

Gets the current amount of stall, between 0 and 1. A value greater than 0.005 indicates a minor stall and a value greater than 0.5 indicates a large-scale stall.

Note: Requires Ferram Aerospace Research.

float DragCoefficient { get; }

Gets the coefficient of drag. This is the amount of drag produced by the vessel. It depends on air speed, air density and wing area.

Note: Requires Ferram Aerospace Research.

float LiftCoefficient { get; }

Gets the coefficient of lift. This is the amount of lift produced by the vessel, and depends on air speed, air density and wing area.

Note: Requires Ferram Aerospace Research.

float BallisticCoefficient { get; }

Gets the ballistic coefficient.

Note: Requires Ferram Aerospace Research.

float ThrustSpecificFuelConsumption { get; }

Gets the thrust specific fuel consumption for the jet engines on the vessel. This is a measure of the efficiency of the engines, with a lower value indicating a more efficient vessel. This value is the number of Newtons of fuel that are burned, per hour, to product one newton of thrust.

Note: Requires Ferram Aerospace Research.

3.2.5 Orbit

class Orbit

Describes an orbit. For example, the orbit of a vessel, obtained by calling <code>Vessel.Orbit</code>, or a celestial body, obtained by calling <code>CelestialBody.Orbit</code>.

CelestialBody Body { get; }

The celestial body (e.g. planet or moon) around which the object is orbiting.

double Apoapsis { get; }

Gets the apoapsis of the orbit, in meters, from the center of mass of the body being orbited.

Note: For the apoapsis altitude reported on the in-game map view, use Orbit.ApoapsisAltitude.

double Periapsis { get; }

The periapsis of the orbit, in meters, from the center of mass of the body being orbited.

Note: For the periapsis altitude reported on the in-game map view, use Orbit.PeriapsisAltitude.

double ApoapsisAltitude { get; }

The apoapsis of the orbit, in meters, above the sea level of the body being orbited.

Note: This is equal to Orbit . Apoapsis minus the equatorial radius of the body.

double PeriapsisAltitude { get; }

The periapsis of the orbit, in meters, above the sea level of the body being orbited.

Note: This is equal to *Orbit* . *Periapsis* minus the equatorial radius of the body.

double SemiMajorAxis { get; }

The semi-major axis of the orbit, in meters.

double SemiMinorAxis { get; }

The semi-minor axis of the orbit, in meters.

double Radius { get; }

The current radius of the orbit, in meters. This is the distance between the center of mass of the object in orbit, and the center of mass of the body around which it is orbiting.

Note: This value will change over time if the orbit is elliptical.

double Speed { get; }

The current orbital speed of the object in meters per second.

Note: This value will change over time if the orbit is elliptical.

double Period { get; }

The orbital period, in seconds.

double TimeToApoapsis { get; }

The time until the object reaches apoapsis, in seconds.

double TimeToPeriapsis { get; }

The time until the object reaches periapsis, in seconds.

double Eccentricity { get; }

The eccentricity of the orbit.

double Inclination { get; }

The inclination of the orbit, in radians.

double LongitudeOfAscendingNode { get; }

The longitude of the ascending node, in radians.

double ArgumentOfPeriapsis { get; }

The argument of periapsis, in radians.

double MeanAnomalyAtEpoch { get; }

The mean anomaly at epoch.

double Epoch { get; }

The time since the epoch (the point at which the mean anomaly at epoch was measured, in seconds.

double MeanAnomaly { get; }

The mean anomaly.

double EccentricAnomaly { get; }

The eccentric anomaly.

static Tuple<double, double, double> ReferencePlaneNormal (ReferenceFrame referenceFrame)

The unit direction vector that is normal to the orbits reference plane, in the given reference frame. The reference plane is the plane from which the orbits inclination is measured.

Parameters

static Tuple < double, double > ReferencePlaneDirection (ReferenceFrame reference-

The unit direction vector from which the orbits longitude of ascending node is measured, in the given reference frame.

Parameters

double TimeToSOIChange { get; }

The time until the object changes sphere of influence, in seconds. Returns NaN if the object is not going to change sphere of influence.

Orbit NextOrbit { get; }

If the object is going to change sphere of influence in the future, returns the new orbit after the change. Otherwise returns null.

3.2.6 Control

class Control

Used to manipulate the controls of a vessel. This includes adjusting the throttle, enabling/disabling systems such as SAS and RCS, or altering the direction in which the vessel is pointing.

Note: Control inputs (such as pitch, yaw and roll) are zeroed when all clients that have set one or more of these inputs are no longer connected.

bool SAS { get; set; }

The state of SAS.

Note: Equivalent to AutoPilot.SAS

SASMode SASMode { get; set; }

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Note: Equivalent to AutoPilot.SASMode

SpeedMode SpeedMode { get; set; }

The current SpeedMode of the navball. This is the mode displayed next to the speed at the top of the navball.

bool RCS { get; set; }

The state of RCS.

bool Gear { get; set; }

The state of the landing gear/legs.

bool Lights { get; set; }

The state of the lights.

bool Brakes { get; set; }

The state of the wheel brakes.

bool Abort { get; set; }

The state of the abort action group.

float Throttle { get; set; }

The state of the throttle. A value between 0 and 1.

float Pitch { get; set; }

The state of the pitch control. A value between -1 and 1. Equivalent to the w and s keys.

float Yaw { get; set; }

The state of the yaw control. A value between -1 and 1. Equivalent to the a and d keys.

float Roll { get; set; }

The state of the roll control. A value between -1 and 1. Equivalent to the q and e keys.

float Forward { get; set; }

The state of the forward translational control. A value between -1 and 1. Equivalent to the h and n keys.

float Up { get; set; }

The state of the up translational control. A value between -1 and 1. Equivalent to the i and k keys.

float Right { get; set; }

The state of the right translational control. A value between -1 and 1. Equivalent to the j and l keys.

float WheelThrottle { get; set; }

The state of the wheel throttle. A value between -1 and 1. A value of 1 rotates the wheels forwards, a value of -1 rotates the wheels backwards.

float WheelSteering { get; set; }

The state of the wheel steering. A value between -1 and 1. A value of 1 steers to the left, and a value of -1 steers to the right.

int CurrentStage { get; }

The current stage of the vessel. Corresponds to the stage number in the in-game UI.

IList<Vessel> ActivateNextStage ()

Activates the next stage. Equivalent to pressing the space bar in-game.

Returns A list of vessel objects that are jettisoned from the active vessel.

bool GetActionGroup (uint group)

Returns true if the given action group is enabled.

Parameters

• group – A number between 0 and 9 inclusive.

void SetActionGroup (uint group, bool state)

Sets the state of the given action group (a value between 0 and 9 inclusive).

Parameters

• group – A number between 0 and 9 inclusive.

void ToggleActionGroup (uint group)

Toggles the state of the given action group.

Parameters

• group – A number between 0 and 9 inclusive.

Node AddNode (double UT, float prograde = 0.0, float normal = 0.0, float radial = 0.0)

Creates a maneuver node at the given universal time, and returns a *Node* object that can be used to modify it. Optionally sets the magnitude of the delta-v for the maneuver node in the prograde, normal and radial directions.

Parameters

- UT Universal time of the maneuver node.
- **prograde** Delta-v in the prograde direction.
- normal Delta-v in the normal direction.
- radial Delta-v in the radial direction.

IList<Node> Nodes { get; }

Returns a list of all existing maneuver nodes, ordered by time from first to last.

void RemoveNodes ()

Remove all maneuver nodes.

enum SASMode

The behavior of the SAS auto-pilot. See AutoPilot.SASMode.

StabilityAssist

Stability assist mode. Dampen out any rotation.

Maneuver

Point in the burn direction of the next maneuver node.

Prograde

Point in the prograde direction.

Retrograde

Point in the retrograde direction.

Normal

Point in the orbit normal direction.

AntiNormal

Point in the orbit anti-normal direction.

Radial

Point in the orbit radial direction.

AntiRadial

Point in the orbit anti-radial direction.

Target

Point in the direction of the current target.

AntiTarget

Point away from the current target.

enum SpeedMode

See Control.SpeedMode.

Orbit

Speed is relative to the vessel's orbit.

Surface

Speed is relative to the surface of the body being orbited.

Target

Speed is relative to the current target.

3.2.7 Parts

The following classes allow interaction with a vessels individual parts.

- Parts
- Part
- Module
- Specific Types of Part
 - Decoupler
 - Docking Port
 - Engine
 - Landing Gear
 - Landing Leg
 - Launch Clamp
 - Light
 - Parachute
 - Radiator
 - Resource Converter
 - Resource Harvester
 - Reaction Wheel
 - Sensor
 - Solar Panel
- Trees of Parts
 - Traversing the Tree
 - Attachment Modes
- Fuel Lines
- Staging

Parts

class Parts

Instances of this class are used to interact with the parts of a vessel. An instance can be obtained by calling *Vessel.Parts*.

IList<Part> All { get; }

A list of all of the vessels parts.

Part Root { get; }

The vessels root part.

Note: See the discussion on *Trees of Parts*.

Part Controlling { get; set; }

The part from which the vessel is controlled.

IList<Part> WithName (string name)

A list of parts whose Part. Name is name.

Parameters

IList<Part> WithTitle (string title)

A list of all parts whose Part. Title is title.

Parameters

IList<Part> WithModule (string moduleName)

A list of all parts that contain a Module whose Module. Name is moduleName.

Parameters

IList<Part> InStage (int stage)

A list of all parts that are activated in the given stage.

Parameters

Note: See the discussion on *Staging*.

IList<Part> InDecoupleStage (int stage)

A list of all parts that are decoupled in the given stage.

Parameters

Note: See the discussion on *Staging*.

IList<Module> ModulesWithName (string moduleName)

A list of modules (combined across all parts in the vessel) whose Module. Name is moduleName.

Parameters

IList<Decoupler> Decouplers { get; }

A list of all decouplers in the vessel.

List<DockingPort> DockingPorts { get; }

A list of all docking ports in the vessel.

DockingPort DockingPortWithName (string name)

The first docking port in the vessel with the given port name, as returned by <code>DockingPort.Name</code>. Returns null if there are no such docking ports.

Parameters

IList<Engine> Engines { get; }

A list of all engines in the vessel.

IList<LandingGear> LandingGear { get; }

A list of all landing gear attached to the vessel.

IList<LandingLeg> LandingLegs { get; }

A list of all landing legs attached to the vessel.

IList<LaunchClamp> LaunchClamps { get; }

A list of all launch clamps attached to the vessel.

IList<Light> Lights { get; }

A list of all lights in the vessel.

IList<Parachute> Parachutes { get; }

A list of all parachutes in the vessel.

IList<Radiator> Radiators { get; }

A list of all radiators in the vessel.

List<ResourceConverter> ResourceConverters { get; }

A list of all resource converters in the vessel.

IList<ResourceHarvester> ResourceHarvesters { get; }

A list of all resource harvesters in the vessel.

IList<ReactionWheel> ReactionWheels { get; }

A list of all reaction wheels in the vessel.

IList<Sensor> Sensors { get; }

A list of all sensors in the vessel.

IList<SolarPanel> SolarPanels { get; }

A list of all solar panels in the vessel.

Part

class Part

Instances of this class represents a part. A vessel is made of multiple parts. Instances can be obtained by various methods in *Parts*.

string Name { get; }

Internal name of the part, as used in part cfg files. For example "Mark1-2Pod".

string Title { get; }

Title of the part, as shown when the part is right clicked in-game. For example "Mk1-2 Command Pod".

double Cost { get; }

The cost of the part, in units of funds.

Vessel Vessel { get; }

The vessel that contains this part.

Part Parent { get; }

The parts parent. Returns null if the part does not have a parent. This, in combination with <code>Part.Children</code>, can be used to traverse the vessels parts tree.

Note: See the discussion on *Trees of Parts*.

IList<Part> Children { get; }

The parts children. Returns an empty list if the part has no children. This, in combination with <code>Part.Parent</code>, can be used to traverse the vessels parts tree.

Note: See the discussion on *Trees of Parts*.

bool AxiallyAttached { get; }

Whether the part is axially attached to its parent, i.e. on the top or bottom of its parent. If the part has no parent, returns false.

Note: See the discussion on *Attachment Modes*.

bool RadiallyAttached { get; }

Whether the part is radially attached to its parent, i.e. on the side of its parent. If the part has no parent, returns false.

Note: See the discussion on Attachment Modes.

int Stage { get; }

The stage in which this part will be activated. Returns -1 if the part is not activated by staging.

Note: See the discussion on *Staging*.

int DecoupleStage { get; }

The stage in which this part will be decoupled. Returns -1 if the part is never decoupled from the vessel.

Note: See the discussion on *Staging*.

bool Massless { get; }

Whether the part is massless.

double Mass { get; }

The current mass of the part, including resources it contains, in kilograms. Returns zero if the part is massless.

double DryMass { get; }

The mass of the part, not including any resources it contains, in kilograms. Returns zero if the part is massless.

double ImpactTolerance { get; }

The impact tolerance of the part, in meters per second.

double Temperature { get; }

Temperature of the part, in Kelvin.

double SkinTemperature { get; }

Temperature of the skin of the part, in Kelvin.

double MaxTemperature { get; }

Maximum temperature that the part can survive, in Kelvin.

double MaxSkinTemperature { get; }

Maximum temperature that the skin of the part can survive, in Kelvin.

double ExternalTemperature { get; }

Temperature of the atmosphere/vacuum surrounding the part, in Kelvin. This does not include heating from direct sunlight.

double ThermalMass { get; }

How much it takes to heat up the part.

double ThermalSkinMass { get; }

How much it takes to heat up the part's skin.

double ThermalResourceMass { get; }

How much it takes to heat up resources in the part.

double ThermalConductionFlux { get; }

The speed that heat is conducting into or out of the part through contact with other parts. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double ThermalConvectionFlux { get; }

The speed that heat is convecting into or out of the part from the surrounding atmosphere. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double ThermalRadiationFlux { get; }

The speed that heat is radiating into or out of the part from the surrounding vacuum. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double ThermalInternalFlux { get; }

The speed that heat is generated by the part. For example, engines generate heat by burning fuel. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double ThermalSkinToInternalFlux { get; }

The speed that heat is conducting between the part's skin and its internals.

Resources Resources { get; }

A Resources object for the part.

bool Crossfeed { get; }

Whether this part is crossfeed capable.

bool IsFuelLine { get; }

Whether this part is a fuel line.

IList<Part> FuelLinesFrom { get; }

The parts that are connected to this part via fuel lines, where the direction of the fuel line is into this part.

Note: See the discussion on *Fuel Lines*.

IList<Part> FuelLinesTo { get; }

The parts that are connected to this part via fuel lines, where the direction of the fuel line is out of this part.

Note: See the discussion on *Fuel Lines*.

IList<Module> Modules { get; }

The modules for this part.

Decoupler Decoupler { get; }

A Decoupler if the part is a decoupler, otherwise null.

DockingPort DockingPort { get; }

A DockingPort if the part is a docking port, otherwise null.

Engine Engine { get; }

An *Engine* if the part is an engine, otherwise null.

LandingGear LandingGear { get; }

A LandingGear if the part is a landing gear, otherwise null.

LandingLeg LandingLeg { get; }

A LandingLeg if the part is a landing leg, otherwise null.

LaunchClamp LaunchClamp { get; }

A LaunchClamp if the part is a launch clamp, otherwise null.

Light Light { get; }

A Light if the part is a light, otherwise null.

Parachute Parachute { get; }

A Parachute if the part is a parachute, otherwise null.

Radiator Radiator { get; }

A Radiator if the part is a radiator, otherwise null.

ReactionWheel ReactionWheel { get; }

A ReactionWheel if the part is a reaction wheel, otherwise null.

ResourceConverter ResourceConverter { get; }

A ResourceConverter if the part is a resource converter, otherwise null.

ResourceHarvester ResourceHarvester { get; }

A ResourceHarvester if the part is a resource harvester, otherwise null.

Sensor { get; }

A Sensor if the part is a sensor, otherwise null.

SolarPanel { get; }

A SolarPanel if the part is a solar panel, otherwise null.

Tuple<double, double> Position (ReferenceFrame referenceFrame)

The position of the part in the given reference frame.

Parameters

Tuple<double, double> Direction (ReferenceFrame referenceFrame)

The direction of the part in the given reference frame.

Parameters

Tuple<double, double, double> **Velocity** (*ReferenceFrame referenceFrame*)

The velocity of the part in the given reference frame.

Parameters

Tuple<double, double, double> Rotation (ReferenceFrame referenceFrame)

The rotation of the part in the given reference frame.

Parameters

ReferenceFrame ReferenceFrame { get; }

The reference frame that is fixed relative to this part.

- •The origin is at the position of the part.
- •The axes rotate with the part.
- •The x, y and z axis directions depend on the design of the part.

Note: For docking port parts, this reference frame is not necessarily equivalent to the reference frame for the docking port, returned by <code>DockingPort.ReferenceFrame</code>.

Module

class Module

In KSP, each part has zero or more PartModules associated with it. Each one contains some of the functionality of the part. For example, an engine has a "ModuleEngines" PartModule that contains all the functionality of an engine. This class allows you to interact with KSPs PartModules, and any PartModules that have been added by other mods.

string Name { get; }

Name of the PartModule. For example, "ModuleEngines".

Part Part { get; }

The part that contains this module.

IDictionary<string, string> Fields { get; }

The modules field names and their associated values, as a dictionary. These are the values visible in the right-click menu of the part.

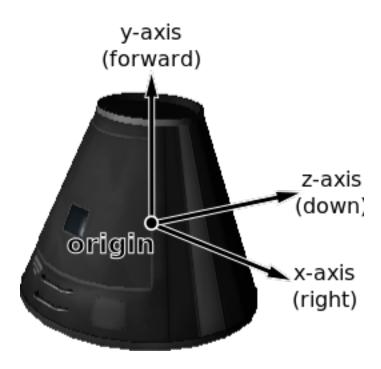


Fig. 3.7: Mk1 Command Pod reference frame origin and axes

bool HasField (string name)

Returns true if the module has a field with the given name.

Parameters

• name – Name of the field.

string GetField (string name)

Returns the value of a field.

Parameters

• name – Name of the field.

IList<string> Events { get; }

A list of the names of all of the modules events. Events are the clickable buttons visible in the right-click menu of the part.

bool HasEvent (string name)

true if the module has an event with the given name.

Parameters

void TriggerEvent (string name)

Trigger the named event. Equivalent to clicking the button in the right-click menu of the part.

Parameters

IList<string> Actions { get; }

A list of all the names of the modules actions. These are the parts actions that can be assigned to action groups in the in-game editor.

bool HasAction (string name)

true if the part has an action with the given name.

Parameters

```
void SetAction (string name, bool value = True)
```

Set the value of an action with the given name.

Parameters

Specific Types of Part

The following classes provide functionality for specific types of part.

- Decoupler
- · Docking Port
- Engine
- Landing Gear
- Landing Leg
- Launch Clamp
- Light
- Parachute
- Radiator
- Resource Converter
- Resource Harvester
- Reaction Wheel
- Sensor
- · Solar Panel

Decoupler

```
class Decoupler
```

```
Obtained by calling Part.Decoupler
```

```
Part Part { get; }
```

The part object for this decoupler.

void Decouple ()

Fires the decoupler. Has no effect if the decoupler has already fired.

bool Decoupled { get; }

Whether the decoupler has fired.

```
float Impulse { get; }
```

The impulse that the decoupler imparts when it is fired, in Newton seconds.

Docking Port

class DockingPort

```
Obtained by calling Part.DockingPort
```

```
Part Part { get; }
```

The part object for this docking port.

```
string Name { get; set; }
```

The port name of the docking port. This is the name of the port that can be set in the right click menu, when the Docking Port Alignment Indicator mod is installed. If this mod is not installed, returns the title of the part (Part. Title).

DockingPortState State { get; }

The current state of the docking port.

Part DockedPart { get; }

The part that this docking port is docked to. Returns null if this docking port is not docked to anything.

Vessel Undock ()

Undocks the docking port and returns the vessel that was undocked from. After undocking, the active vessel may change (ActiveVessel). This method can be called for either docking port in a docked pair - both calls will have the same effect. Returns null if the docking port is not docked to anything.

float ReengageDistance { get; }

The distance a docking port must move away when it undocks before it becomes ready to dock with another port, in meters.

bool HasShield { get; }

Whether the docking port has a shield.

bool Shielded { get; set; }

The state of the docking ports shield, if it has one. Returns true if the docking port has a shield, and the shield is closed. Otherwise returns false. When set to true, the shield is closed, and when set to false the shield is opened. If the docking port does not have a shield, setting this attribute has no effect.

Tuple<double, double, double> Position (ReferenceFrame referenceFrame)

The position of the docking port in the given reference frame.

Parameters

Tuple<double, double, double> **Direction** (*ReferenceFrame referenceFrame*)

The direction that docking port points in, in the given reference frame.

Parameters

Tuple<double, double, double> Rotation (ReferenceFrame referenceFrame)

The rotation of the docking port, in the given reference frame.

Parameters

ReferenceFrame ReferenceFrame { get; }

The reference frame that is fixed relative to this docking port, and oriented with the port.

- •The origin is at the position of the docking port.
- •The axes rotate with the docking port.
- •The x-axis points out to the right side of the docking port.
- •The y-axis points in the direction the docking port is facing.
- •The z-axis points out of the bottom off the docking port.

Note: This reference frame is not necessarily equivalent to the reference frame for the part, returned by <code>Part.ReferenceFrame</code>.

enum DockingPortState

See DockingPort.State.

Ready

The docking port is ready to dock to another docking port.

Docked

The docking port is docked to another docking port, or docked to another part (from the VAB/SPH).

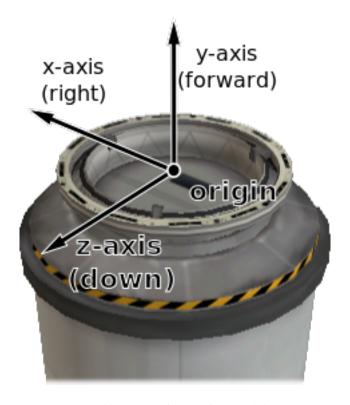


Fig. 3.8: Docking port reference frame origin and axes

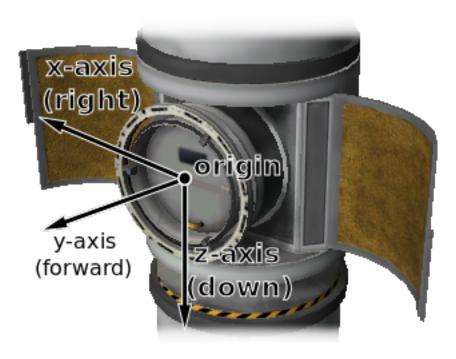


Fig. 3.9: Inline docking port reference frame origin and axes

Docking

The docking port is very close to another docking port, but has not docked. It is using magnetic force to acquire a solid dock.

Undocking

The docking port has just been undocked from another docking port, and is disabled until it moves away by a sufficient distance (DockingPort.ReengageDistance).

Shielded

The docking port has a shield, and the shield is closed.

Moving

The docking ports shield is currently opening/closing.

Engine

class Engine

Obtained by calling Part. Engine.

Part Part { get; }

The part object for this engine.

bool Active { get; set; }

Whether the engine is active. Setting this attribute may have no effect, depending on Engine.CanShutdown and Engine.CanRestart.

float Thrust { get; }

The current amount of thrust being produced by the engine, in Newtons. Returns zero if the engine is not active or if it has no fuel.

float AvailableThrust { get; }

The maximum available amount of thrust that can be produced by the engine, in Newtons. This takes *Engine.ThrustLimit* into account, and is the amount of thrust produced by the engine when activated and the main throttle is set to 100%. Returns zero if the engine does not have any fuel.

float MaxThrust { get; }

Gets the maximum amount of thrust that can be produced by the engine, in Newtons. This is the amount of thrust produced by the engine when activated, *Engine.ThrustLimit* is set to 100% and the main vessel's throttle is set to 100%.

float MaxVacuumThrust { get; }

The maximum amount of thrust that can be produced by the engine in a vacuum, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine.ThrustLimit</code> is set to 100%, the main vessel's throttle is set to 100% and the engine is in a vacuum.

float ThrustLimit { get; set; }

The thrust limiter of the engine. A value between 0 and 1. Setting this attribute may have no effect, for example the thrust limit for a solid rocket booster cannot be changed in flight.

float SpecificImpulse { get; }

The current specific impulse of the engine, in seconds. Returns zero if the engine is not active.

float VacuumSpecificImpulse { get; }

The vacuum specific impulse of the engine, in seconds.

float KerbinSeaLevelSpecificImpulse { get; }

The specific impulse of the engine at sea level on Kerbin, in seconds.

IList<string> Propellants { get; }

The names of resources that the engine consumes.

IDictionary<string, float> PropellantRatios { get; }

The ratios of resources that the engine consumes. A dictionary mapping resource names to the ratios at which they are consumed by the engine.

bool HasFuel { get; }

Whether the engine has run out of fuel (or flamed out).

float Throttle { get; }

The current throttle setting for the engine. A value between 0 and 1. This is not necessarily the same as the vessel's main throttle setting, as some engines take time to adjust their throttle (such as jet engines).

bool ThrottleLocked { get; }

Whether the <code>Control.Throttle</code> affects the engine. For example, this is true for liquid fueled rockets, and false for solid rocket boosters.

bool CanRestart { get; }

Whether the engine can be restarted once shutdown. If the engine cannot be shutdown, returns false. For example, this is true for liquid fueled rockets and false for solid rocket boosters.

bool CanShutdown { get; }

Gets whether the engine can be shutdown once activated. For example, this is true for liquid fueled rockets and false for solid rocket boosters.

bool Gimballed { get; }

Whether the engine nozzle is gimballed, i.e. can provide a turning force.

float GimbalRange { get; }

The range over which the gimbal can move, in degrees.

bool GimbalLocked { get; set; }

Whether the engines gimbal is locked in place. Setting this attribute has no effect if the engine is not gimballed.

float GimbalLimit { get; set; }

The gimbal limiter of the engine. A value between 0 and 1. Returns 0 if the gimbal is locked or the engine is not gimballed. Setting this attribute has no effect if the engine is not gimballed.

Landing Gear

class LandingGear

Obtained by calling Part.LandingGear.

Part Part { get; }

The part object for this landing gear.

LandingGearState State { get; }

Gets the current state of the landing gear.

bool Deployed { get; set; }

Whether the landing gear is deployed.

enum LandingGearState

See LandingGear. State.

Deployed

Landing gear is fully deployed.

Retracted

Landing gear is fully retracted.

Deploying Landing gear is being deployed. Retracting Landing gear is being retracted. **Landing Leg**

class LandingLeg

```
Obtained by calling Part.LandingLeg.
```

```
Part Part { get; }
```

The part object for this landing leg.

LandingLegState State { get; }

The current state of the landing leg.

bool Deployed { get; set; }

Whether the landing leg is deployed.

enum LandingLegState

See LandingLeg. State.

Deployed

Landing leg is fully deployed.

Retracted

Landing leg is fully retracted.

Deploying

Landing leg is being deployed.

Retracting

Landing leg is being retracted.

Broken

Landing leg is broken.

Repairing

Landing leg is being repaired.

Launch Clamp

class LaunchClamp

```
Obtained by calling Part.LaunchClamp.
```

```
Part Part { get; }
```

The part object for this launch clamp.

void Release ()

Releases the docking clamp. Has no effect if the clamp has already been released.

Light

class Light

Obtained by calling Part.Light.

```
Part Part { get; }
           The part object for this light.
     bool Active { get; set; }
           Whether the light is switched on.
     float PowerUsage { get; }
           The current power usage, in units of charge per second.
Parachute
class Parachute
     Obtained by calling Part. Parachute.
     Part Part { get; }
           The part object for this parachute.
     void Deploy ()
           Deploys the parachute. This has no effect if the parachute has already been deployed.
     bool Deployed { get; }
           Whether the parachute has been deployed.
     ParachuteState State { get; }
           The current state of the parachute.
     float DeployAltitude { get; set; }
           The altitude at which the parachute will full deploy, in meters.
     float DeployMinPressure { get; set; }
           The minimum pressure at which the parachute will semi-deploy, in atmospheres.
enum ParachuteState
     See Parachute. State.
     Stowed
           The parachute is safely tucked away inside its housing.
           The parachute is still stowed, but ready to semi-deploy.
     SemiDeployed
           The parachute has been deployed and is providing some drag, but is not fully deployed yet.
     Deployed
           The parachute is fully deployed.
     Cut
           The parachute has been cut.
Radiator
class Radiator
     Obtained by calling Part.Radiator.
     Part Part { get; }
           The part object for this radiator.
```

bool Deployed { get; set; }

Whether the radiator is extended.

RadiatorState State { get; }

The current state of the radiator.

enum RadiatorState

RadiatorState

Extended

Radiator is fully extended.

Retracted

Radiator is fully retracted.

Extending

Radiator is being extended.

Retracting

Radiator is being retracted.

Broken

Radiator is being broken.

Resource Converter

class ResourceConverter

 $Obtained \ by \ calling \ \textit{Part.ResourceConverter}.$

Part Part { get; }

The part object for this converter.

int Count { get; }

The number of converters in the part.

string Name (int index)

The name of the specified converter.

Parameters

• index – Index of the converter.

bool Active (int index)

True if the specified converter is active.

Parameters

• index – Index of the converter.

void Start (int index)

Start the specified converter.

Parameters

• index – Index of the converter.

void Stop (int index)

Stop the specified converter.

Parameters

• index – Index of the converter.

ResourceConverterState State (int index)

The state of the specified converter.

Parameters

• index – Index of the converter.

string StatusInfo (int index)

Status information for the specified converter. This is the full status message shown in the in-game UI.

Parameters

• index – Index of the converter.

IList<string> Inputs (int index)

List of the names of resources consumed by the specified converter.

Parameters

• index – Index of the converter.

IList<string> Outputs (int index)

List of the names of resources produced by the specified converter.

Parameters

• index – Index of the converter.

enum ResourceConverterState

See ResourceConverter.State.

Running

Converter is running.

Idle

Converter is idle.

MissingResource

Converter is missing a required resource.

StorageFull

No available storage for output resource.

Capacity

At preset resource capacity.

Unknown

Unknown state. Possible with modified resource converters. In this case, check ResourceConverter.StatusInfo for more information.

Resource Harvester

class ResourceHarvester

 $Obtained \ by \ calling \ \textit{Part.ResourceHarvester.}$

Part Part { get; }

The part object for this harvester.

ResourceHarvesterState State { get; }

The state of the harvester.

bool Deployed { get; set; }

Whether the harvester is deployed.

bool Active { get; set; }

Whether the harvester is actively drilling.

```
float ExtractionRate { get; }
    The rate at which the drill is extracting ore, in units per second.

float ThermalEfficiency { get; }
    The thermal efficiency of the drill, as a percentage of its maximum.

float CoreTemperature { get; }
    The core temperature of the drill, in Kelvin.

float OptimumCoreTemperature { get; }
    The core temperature at which the drill will operate with peak efficiency, in Kelvin.

Reaction Wheel

class ReactionWheel
    Obtained by calling Part.ReactionWheel.

Part Part { get; }
```

bool Broken { get; }

bool Active { get; set; }

Whether the reaction wheel is broken.

Whether the reaction wheel is active.

The part object for this reaction wheel.

float PitchTorque { get; }

The torque in the pitch axis, in Newton meters.

float YawTorque { get; }

The torque in the yaw axis, in Newton meters.

float RollTorque { get; }

The torque in the roll axis, in Newton meters.

Sensor

class Sensor

```
Obtained by calling Part. Sensor.
```

Part Part { get; }

The part object for this sensor.

bool Active { get; set; }

Whether the sensor is active.

string Value { get; }

The current value of the sensor.

float PowerUsage { get; }

The current power usage of the sensor, in units of charge per second.

Solar Panel

class SolarPanel

Obtained by calling Part.SolarPanel.

Part Part { get; }

The part object for this solar panel.

bool Deployed { get; set; }

Whether the solar panel is extended.

SolarPanelState State { get; }

The current state of the solar panel.

float EnergyFlow { get; }

The current amount of energy being generated by the solar panel, in units of charge per second.

float SunExposure { get; }

The current amount of sunlight that is incident on the solar panel, as a percentage. A value between 0 and 1.

enum SolarPanelState

See SolarPanel.State.

Extended

Solar panel is fully extended.

Retracted

Solar panel is fully retracted.

Extending

Solar panel is being extended.

Retracting

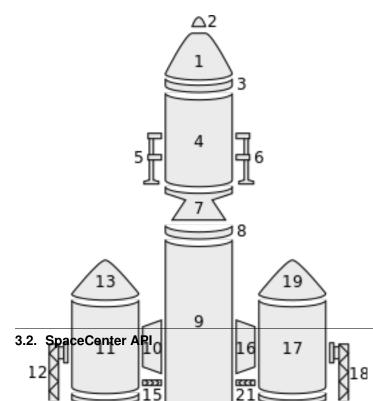
Solar panel is being retracted.

Broken

Solar panel is broken.

Trees of Parts

Vessels in KSP comprised of are number of parts, connected to one another in a a tree structure. An example vessel is shown in Figure 1, and the corresponding tree of The craft file downloaded here. parts in Figure for this example can also be 2.



Traversing the Tree

The tree of parts can be traversed using the attributes SpaceCenter.Parts.Root, SpaceCenter.Part.Parent and SpaceCenter.Part.Children.

The root of the tree is the same as the vessels root part (part number 1 in the example above) and can be obtained by calling SpaceCenter.Parts.Root. A parts children can be obtained by calling SpaceCenter.Part.Children. If the part does not have any children, SpaceCenter.Part.Children returns an empty list. A parts parent can be obtained by calling SpaceCenter.Part.Parent.

```
If the part does not have a parent (as is the case for the root part), SpaceCenter.Part.Parent returns null.
```

The following C# example uses these attributes to perform a depth-first traversal over all of the parts in a vessel:

```
using KRPC.Client;
using KRPC.Client.Services.SpaceCenter;
using System;
using System.Collections.Generic;
using System.Net;
class AttachmentModes
    public static void Main ()
        var connection = new KRPC.Client.Connect
        var vessel = connection.SpaceCenter ().
        var root = vessel.Parts.Root;
        var stack = new Stack<KRPC Client.Tuple</pre>
        stack.Push (new KRPC.Client.Tuple<Part,
        while (stack.Count > 0) {
            var item = stack.Pop ();
            Part part = item.Item1;
            int depth = item.Item2;
            Console.WriteLine (new String (' ',
            foreach (var child in part.Children)
                stack.Push (new KRPC.Client.Tup)
    }
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1
TR-18A Stack Decoupler
 FL-T400 Fuel Tank
  LV-909 Liquid Fuel Engine
   TR-18A Stack Decoupler
    FL-T800 Fuel Tank
     LV-909 Liquid Fuel Engine
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
        TT18-A Launch Stability Enhancer
        FTX-2 External Fuel Duct
       LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
        TT18-A Launch Stability Enhancer
        FTX-2 External Fuel Duct
        LV-909 Liquid Fuel Engine
        Aerodynamic Nose Cone
  LT-1 Landing Struts
```

```
LT-1 Landing Struts
Mk16 Parachute
```

Attachment Modes

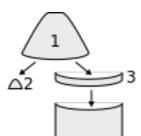
Parts can be attached to other parts either *radially* (on the side of the parent part) or *axially* (on the end of the parent part, to form a stack).

For example, in the vessel pictured above, the parachute (part 2) is *axially* connected to its

parent (the command pod – part 1), and the landing leg (part 5) is *radially* connected to its parent (the fuel tank – part 4).

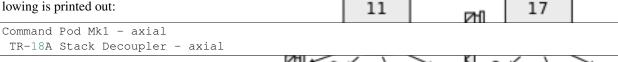
The root part of a vessel (for example the command pod – part 1) does not have a parent part, so does not have an attachment mode. However, the part is consider to be *axially* attached to nothing.

The following C# example does a depth-first traversal as before, but also prints out the attachment mode used by the part:



```
using KRPC.Client;
using KRPC.Client.Services.SpaceCenter;
using System;
using System.Collections.Generic;
using System.Net;
class AttachmentModes
   public static void Main ()
        var connection = new KRPC.Client.Connection ();
        var vessel = connection.SpaceCenter ().ActiveVessel;
        var root = vessel.Parts.Root;
        var stack = new Stack<KRPC.Client.Tuple<Part,int>> ();
        stack.Push (new KRPC.Client.Tuple<Part,int> (root, 0));
        while (stack.Count > 0) {
            var item = stack.Pop ();
           Part part = item.Item1;
            int depth = item.Item2;
            string attachMode = (part.AxiallyAttached ? "axial" : "radial");
            Console.WriteLine (new String (' ', depth) + part.Title + " - " + attachMode);
            foreach (var child in part.Children)
                stack.Push (new KRPC.Client.Tuple<Part,int> (child, depth + 1));
    }
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:



3.2. SpaceCenter API

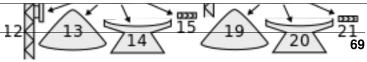


Fig. 3.11: **Figure 2** – Tree of parts for the vessel in Figure 1. Arrows point from the parent part to the child part.

```
FL-T400 Fuel Tank - axial
  LV-909 Liquid Fuel Engine - axial
   TR-18A Stack Decoupler - axial
    FL-T800 Fuel Tank - axial
     LV-909 Liquid Fuel Engine - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
  LT-1 Landing Struts - radial
  LT-1 Landing Struts - radial
Mk16 Parachute - axial
```

Fuel Lines

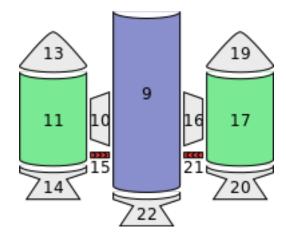
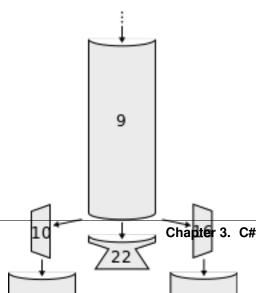


Fig. 3.12: **Figure 5** – Fuel lines from the example in Figure 1. Fuel flows from the parts highlighted in green, into the part highlighted in blue.

Fuel lines are considered parts, and are included in the parts tree (for example, as pictured in Figure 4). However, the parts tree does not contain information about which parts fuel lines connect to. The parent part of a fuel line is the part from which it will take fuel (as shown in Figure 4) however the part that it will send fuel to is not represented in the parts tree.

Figure 5 shows the fuel lines from the example vessel pictured earlier. Fuel line part 15 (in red) takes fuel from a fuel tank (part 11 – in green) and feeds it into another fuel tank (part 9 – in blue). The fuel line is therefore a child of part 11, but its connection to part 9 is not represented in the tree.

The attributes SpaceCenter.Part.FuelLinesFrom and



SpaceCenter.Part.FuelLinesTo can be used to discover these connections. In the example in Figure 5, when SpaceCenter.Part.FuelLinesTo is called on fuel tank part 11, it will return a list of parts containing just fuel tank part 9 (the blue part). When SpaceCenter.Part.FuelLinesFrom is called on fuel tank part 9, it will return a list containing fuel tank parts 11 and 17 (the parts colored green).

Staging

Each part has two staging numbers associated with it: the stage in which the part is *activated* and the stage in which the part is *decoupled*. These values can be obtained using SpaceCenter.Part.Stage and SpaceCenter.Part.DecoupleStage respectively. For parts that are not activated by staging, SpaceCenter.Part.Stage returns -1. For parts that are never decoupled, SpaceCenter.Part.DecoupleStage returns a value of -1.

Figure 6 shows an example staging sequence for a vessel. Figure 7 shows the stages in which each part of the vessel will be *activated*. Figure 8 shows the stages in which each part of the vessel will be *decoupled*.

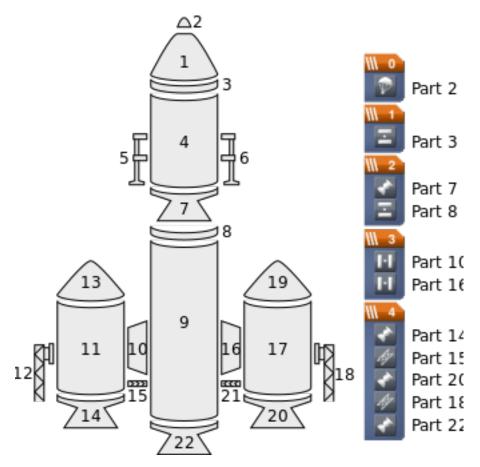


Fig. 3.14: **Figure 6** – Example vessel from Figure 1 with a staging sequence.

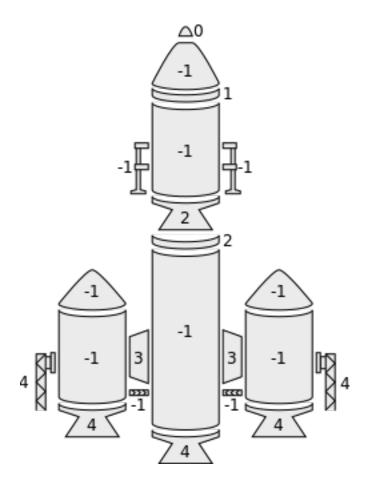


Fig. 3.15: **Figure 7** – The stage in which each part is *activated*.

3.2.8 Resources

class Resources

Created by calling Vessel.Resources, Vessel.ResourcesInDecoupleStage or Part.Resources.

IList<string> Names { get; }

A list of resource names that can be stored.

bool HasResource (string name)

Check whether the named resource can be stored.

Parameters

• name – The name of the resource.

float **Max** (string name)

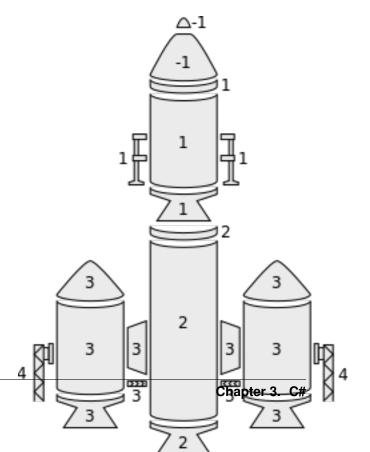
Returns the amount of a resource that can be stored.

Parameters

• name – The name of the resource.

float **Amount** (string name)

Returns the amount of a resource that is currently



stored.

Parameters

• name – The name of the resource.

static float Density (string name)

Returns the density of a resource, in kg/l.

Parameters

• name – The name of the resource.

static ResourceFlowMode FlowMode (string name)

Returns the flow mode of a resource.

Parameters

• name – The name of the resource.

enum ResourceFlowMode

See Resources.FlowMode.

Vessel

The resource flows to any part in the vessel. For example, electric charge.

Stage

The resource flows from parts in the first stage, followed by the second, and so on. For example, mono-propellant.

Adjacent

The resource flows between adjacent parts within the vessel. For example, liquid fuel or oxidizer.

None

The resource does not flow. For example, solid fuel.

3.2.9 Node

class Node

Represents a maneuver node. Can be created using Control.AddNode.

float Prograde { get; set; }

The magnitude of the maneuver nodes delta-v in the prograde direction, in meters per second.

float Normal { get; set; }

The magnitude of the maneuver nodes delta-v in the normal direction, in meters per second.

float Radial { get; set; }

The magnitude of the maneuver nodes delta-v in the radial direction, in meters per second.

float DeltaV { get; set; }

The delta-v of the maneuver node, in meters per second.

Note: Does not change when executing the maneuver node. See *Node.RemainingDeltaV*.

float RemainingDeltaV { get; }

Gets the remaining delta-v of the maneuver node, in meters per second. Changes as the node is executed. This is equivalent to the delta-v reported in-game.

Tuple<double, double, double> **BurnVector** (ReferenceFrame referenceFrame = None)

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s.

Parameters

Note: Does not change when executing the maneuver node. See *Node.RemainingBurnVector*.

Tuple < double, double, double > RemainingBurnVector (ReferenceFrame referenceFrame = None)

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s. The direction and magnitude change as the burn is executed.

Parameters

double UT { get; set; }

The universal time at which the maneuver will occur, in seconds.

double TimeTo { get; }

The time until the maneuver node will be encountered, in seconds.

Orbit Orbit { get; }

The orbit that results from executing the maneuver node.

void Remove ()

Removes the maneuver node.

ReferenceFrame ReferenceFrame { get; }

Gets the reference frame that is fixed relative to the maneuver node's burn.

- •The origin is at the position of the maneuver node.
- •The y-axis points in the direction of the burn.
- •The x-axis and z-axis point in arbitrary but fixed directions.

ReferenceFrame OrbitalReferenceFrame { get; }

Gets the reference frame that is fixed relative to the maneuver node, and orientated with the orbital prograde/normal/radial directions of the original orbit at the maneuver node's position.

- •The origin is at the position of the maneuver node.
- •The x-axis points in the orbital anti-radial direction of the original orbit, at the position of the maneuver node.
- •The y-axis points in the orbital prograde direction of the original orbit, at the position of the maneuver node.
- •The z-axis points in the orbital normal direction of the original orbit, at the position of the maneuver node.

Tuple < double, double > Position (Reference Frame reference Frame)

Returns the position vector of the maneuver node in the given reference frame.

Parameters

Tuple<double, double> **Direction** (*ReferenceFrame referenceFrame*)

Returns the unit direction vector of the maneuver nodes burn in the given reference frame.

Parameters

3.2.10 Comms

class Comms

Used to interact with RemoteTech. Created using a call to Vessel.Comms.

Note: This class requires RemoteTech to be installed.

bool HasLocalControl { get; }

Whether the vessel can be controlled locally.

bool HasFlightComputer { get; }

Whether the vessel has a RemoteTech flight computer on board.

bool HasConnection { get; }

Whether the vessel can receive commands from the KSC or a command station.

bool HasConnectionToGroundStation { get; }

Whether the vessel can transmit science data to a ground station.

double SignalDelay { get; }

The signal delay when sending commands to the vessel, in seconds.

double SignalDelayToGroundStation { get; }

The signal delay between the vessel and the closest ground station, in seconds.

double SignalDelayToVessel (Vessel other)

Returns the signal delay between the current vessel and another vessel, in seconds.

Parameters

3.2.11 ReferenceFrame

class ReferenceFrame

Represents a reference frame for positions, rotations and velocities. Contains:

- •The position of the origin.
- •The directions of the x, y and z axes.
- •The linear velocity of the frame.
- •The angular velocity of the frame.

Note: This class does not contain any properties or methods. It is only used as a parameter to other functions.

3.2.12 AutoPilot

class AutoPilot

Provides basic auto-piloting utilities for a vessel. Created by calling Vessel. AutoPilot.

void Engage ()

Engage the auto-pilot.

void Disengage ()

Disengage the auto-pilot.

void Wait ()

Blocks until the vessel is pointing in the target direction (if set) and has the target roll (if set).

float Error { get; }

The error, in degrees, between the direction the ship has been asked to point in and the direction it is pointing in. Returns zero if the auto-pilot has not been engaged, SAS is not enabled, SAS is in stability assist mode, or no target direction is set.

float RollError { get; }

The error, in degrees, between the roll the ship has been asked to be in and the actual roll. Returns zero if the auto-pilot has not been engaged or no target roll is set.

ReferenceFrame ReferenceFrame { get; set; }

The reference frame for the target direction (AutoPilot.TargetDirection).

Tuple<double, double, double> TargetDirection { get; set; }

The target direction. null if no target direction is set.

void TargetPitchAndHeading (float pitch, float heading)

Set (AutoPilot.TargetDirection) from a pitch and heading angle.

Parameters

- pitch Target pitch angle, in degrees between -90° and +90°.
- **heading** Target heading angle, in degrees between 0° and 360°.

float TargetRoll { get; set; }

The target roll, in degrees. NaN if no target roll is set

bool SAS { get; set; }

The state of SAS.

Note: Equivalent to Control. SAS

SASMode SASMode { get; set; }

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Note: Equivalent to Control.SASMode

float RotationSpeedMultiplier { get; set; }

Target rotation speed multiplier. Defaults to 1.

float MaxRotationSpeed { get; set; }

Maximum target rotation speed. Defaults to 1.

float RollSpeedMultiplier { get; set; }

Target roll speed multiplier. Defaults to 1.

float MaxRollSpeed { get; set; }

Maximum target roll speed. Defaults to 1.

void **SetPIDParameters** (float Kp = 1.0, float Ki = 0.0, float Kd = 0.0)

Sets the gains for the rotation rate PID controller.

Parameters

- **Kp** Proportional gain.
- Ki Integral gain.
- **Kd** Derivative gain.

3.2.13 Geometry Types

class Vector3

3-dimensional vectors are represented as a 3-tuple. For example:

```
using KRPC.Client.Services.SpaceCenter;
using System;
using System.Net;

class VectorExample
{
    public static void Main ()
    {
        var connection = new KRPC.Client.Connection ();
        var vessel = connection.SpaceCenter ().ActiveVessel;
        KRPC.Client.Tuple<double,double, v = vessel.Flight ().Prograde;
        Console.WriteLine (v.Item1 + "," + v.Item2 + "," + v.Item3);
    }
}</pre>
```

class Quaternion

Quaternions (rotations in 3-dimensional space) are encoded as a 4-tuple containing the x, y, z and w components. For example:

```
using KRPC.Client.Services.SpaceCenter;
using System;
using System.Net;

class QuaternionExample
{
    public static void Main ()
    {
       var connection = new KRPC.Client.Connection ();
       var vessel = connection.SpaceCenter ().ActiveVessel;
       KRPC.Client.Tuple<double,double,double> q = vessel.Flight ().Rotation;
       Console.WriteLine (q.Item1 + "," + q.Item2 + "," + q.Item3 + "," + q.Item4);
    }
}
```

3.3 InfernalRobotics API

Provides RPCs to interact with the InfernalRobotics mod. Provides the following classes:

3.3.1 InfernalRobotics

class InfernalRobotics

This service provides functionality to interact with the InfernalRobotics mod.

IList<ControlGroup> ServoGroups { get; }

A list of all the servo groups in the active vessel.

ControlGroup ServoGroupWithName (string name)

Returns the servo group with the given *name* or null if none exists. If multiple servo groups have the same name, only one of them is returned.

Parameters

• name – Name of servo group to find.

Servo ServoWithName (string name)

Returns the servo with the given *name*, from all servo groups, or null if none exists. If multiple servos have the same name, only one of them is returned.

Parameters

• name – Name of the servo to find.

3.3.2 ControlGroup

class ControlGroup

A group of servos, obtained by calling ServoGroups or ServoGroupWithName. Represents the "Servo Groups" in the Infernal-Robotics UI.

string Name { get; set; }

The name of the group.

string ForwardKey { get; set; }

The key assigned to be the "forward" key for the group.

string ReverseKey { get; set; }

The key assigned to be the "reverse" key for the group.

float Speed { get; set; }

The speed multiplier for the group.

bool Expanded { get; set; }

Whether the group is expanded in the Infernal-Robotics UI.

IList<Servo> Servos { get; }

The servos that are in the group.

Servo ServoWithName (string name)

Returns the servo with the given *name* from this group, or null if none exists.

Parameters

• name – Name of servo to find.

void MoveRight ()

Moves all of the servos in the group to the right.

void MoveLeft ()

Moves all of the servos in the group to the left.

void MoveCenter ()

Moves all of the servos in the group to the center.

void MoveNextPreset ()

Moves all of the servos in the group to the next preset.

void MovePrevPreset ()

Moves all of the servos in the group to the previous preset.

void Stop ()

Stops the servos in the group.

3.3.3 Servo

class Servo

Represents a servo. Obtained using ${\it ControlGroup.Servos}, {\it ControlGroup.ServoWithName}$ or ServoWithName.

string Name { get; set; }

The name of the servo.

bool Highlight { set; }

Whether the servo should be highlighted in-game.

float Position { get; }

The position of the servo.

float MinConfigPosition { get; }

The minimum position of the servo, specified by the part configuration.

float MaxConfigPosition { get; }

The maximum position of the servo, specified by the part configuration.

float MinPosition { get; set; }

The minimum position of the servo, specified by the in-game tweak menu.

float MaxPosition { get; set; }

The maximum position of the servo, specified by the in-game tweak menu.

float ConfigSpeed { get; }

The speed multiplier of the servo, specified by the part configuration.

float Speed { get; set; }

The speed multiplier of the servo, specified by the in-game tweak menu.

float CurrentSpeed { get; set; }

The current speed at which the servo is moving.

```
float Acceleration { get; set; }
     The current speed multiplier set in the UI.
bool IsMoving { get; }
     Whether the servo is moving.
bool IsFreeMoving { get; }
     Whether the servo is freely moving.
bool IsLocked { get; set; }
     Whether the servo is locked.
bool IsAxisInverted { get; set; }
     Whether the servos axis is inverted.
void MoveRight ()
     Moves the servo to the right.
void MoveLeft ()
     Moves the servo to the left.
void MoveCenter ()
     Moves the servo to the center.
void MoveNextPreset ()
     Moves the servo to the next preset.
void MovePrevPreset ()
     Moves the servo to the previous preset.
void MoveTo (float position, float speed)
     Moves the servo to position and sets the speed mul-
     tiplier to speed.
 Parameters
```

- **position** The position to move the servo to.
- **speed** Speed multiplier for the movement.

```
void {f Stop} ()
```

Stops the servo.

3.3.4 Example

The following example gets the control group named "MyGroup", prints out the names and positions of all of the servos in the group, then moves all of the servos to the right for 1 second.

```
using KRPC.Client.Services.InfernalRobotics;
using System;
using System.Threading;
using System.Net;

class InfernalRoboticsExample
{
    public static void Main ()
    {
        var connection = new KRPC.Client.Connection (name: "InfernalRobotics Example");
```

```
var ir = connection.InfernalRobotics ();

var group = ir.ServoGroupWithName ("MyGroup");
if (group == null) {
        Console.WriteLine ("Group not found");
        return;
}

foreach (var servo in group.Servos)
        Console.WriteLine (servo.Name + " " + servo.Position);

group.MoveRight ();
Thread.Sleep (1000);
group.Stop ();
}
```

3.4 Kerbal Alarm Clock API

Provides RPCs to interact with the Kerbal Alarm Clock mod. Provides the following classes:

3.4.1 KerbalAlarmClock

class KerbalAlarmClock

This service provides functionality to interact with the Kerbal Alarm Clock mod.

IList<Alarm> Alarms { get; }

A list of all the alarms.

Alarm AlarmWithName (string name)

Get the alarm with the given *name*, or null if no alarms have that name. If more than one alarm has the name, only returns one of them.

Parameters

• name – Name of the alarm to search for.

IList<Alarm> AlarmsWithType (AlarmType type)

Get a list of alarms of the specified *type*.

Parameters

• **type** – Type of alarm to return.

Alarm CreateAlarm (AlarmType type, string name, double ut)

Create a new alarm and return it.

Parameters

- **type** Type of the new alarm.
- name Name of the new alarm.
- ut Time at which the new alarm should trigger.

3.4.2 Alarm

class Alarm

Represents an alarm. Obtained by calling Alarms, AlarmWithName or AlarmsWithType.

AlarmAction Action { get; set; }

The action that the alarm triggers.

double Margin { get; set; }

The number of seconds before the event that the alarm will fire.

double Time { get; set; }

The time at which the alarm will fire.

AlarmType Type { get; }

The type of the alarm.

string ID { get; }

The unique identifier for the alarm.

string Name { get; set; }

The short name of the alarm.

string Notes { get; set; }

The long description of the alarm.

double Remaining { get; }

The number of seconds until the alarm will fire.

bool Repeat { get; set; }

Whether the alarm will be repeated after it has fired.

double RepeatPeriod { get; set; }

The time delay to automatically create an alarm after it has fired.

SpaceCenter.Vessel Vessel { get; set; }

The vessel that the alarm is attached to.

SpaceCenter.CelestialBody XferOriginBody { get; set; }

The celestial body the vessel is departing from.

SpaceCenter.CelestialBody XferTargetBody { get; set; }

The celestial body the vessel is arriving at.

void Remove ()

Removes the alarm.

3.4.3 AlarmType

enum AlarmType

The type of an alarm.

Raw

An alarm for a specific date/time or a specific period in the future.

Maneuver

An alarm based on the next maneuver node on the

current ships flight path. This node will be stored and can be restored when you come back to the ship.

ManeuverAuto

See AlarmType.Maneuver.

Apoapsis

An alarm for furthest part of the orbit from the planet.

Periapsis

An alarm for nearest part of the orbit from the planet.

AscendingNode

Ascending node for the targeted object, or equatorial ascending node.

DescendingNode

Descending node for the targeted object, or equatorial descending node.

Closest

An alarm based on the closest approach of this vessel to the targeted vessel, some number of orbits into the future.

Contract

An alarm based on the expiry or deadline of contracts in career modes.

ContractAuto

See AlarmType.Contract.

Crew

An alarm that is attached to a crew member.

Distance

An alarm that is triggered when a selected target comes within a chosen distance.

EarthTime

An alarm based on the time in the "Earth" alternative Universe (aka the Real World).

LaunchRendevous

An alarm that fires as your landed craft passes under the orbit of your target.

SOIChange

An alarm manually based on when the next SOI point is on the flight path or set to continually monitor the active flight path and add alarms as it detects SOI changes.

SOIChangeAuto

See AlarmType.SOIChange.

Transfer

An alarm based on Interplanetary Transfer Phase Angles, i.e. when should I launch to planet X?

Based on Kosmo Not's post and used in Olex's Calculator.

TransferModelled

See AlarmType. Transfer.

3.4.4 AlarmAction

enum AlarmAction

The action performed by an alarm when it fires.

DoNothing

Don't do anything at all...

DoNothingDeleteWhenPassed

Don't do anything, and delete the alarm.

KillWarp

Drop out of time warp.

KillWarpOnly

Drop out of time warp.

MessageOnly

Display a message.

PauseGame

Pause the game.

3.4.5 Example

The following example creates a new alarm for the active vessel. The alarm is set to trigger after 10 seconds have passed, and display a message.

CHAPTER

FOUR

C++

4.1 C++ Client

The krpc library provides functionality to interact with a kRPC server from C++.

4.1.1 Compiling and Installing the Library

The source archive can be downloaded from github.

To compile and install the library using the configure script, extract the archive and execute the following commands:

```
./configure
make
sudo make install
```

Alternatively, you can use CMake:

```
cmake .
make
sudo make install
```

Installing to a Custom Location

To install the library to a different location, pass the --prefix argument to the configure script. For example:

```
./configure --prefix=/install/path make make install
```

Or set CMAKE_INSTALL_PREFIX when using CMake:

```
cmake . -DCMAKE_INSTALL_PREFIX=/install/path
make
make install
```

Using the Library

To use the library, simply include the main krpc.hpp header file, and the header files for the services that you would like to use. For example, krpc/services/space_center.hpp. Then link against *libkrpc.so* when compiling your application.

4.1.2 Connecting to the Server

To connect to a server, use the *krpc::connect()* function. This returns a connection object through which you can interact with the server. For example to connect to a server running on the local machine:

```
#include <krpc.hpp>
#include <krpc/services/krpc.hpp>
#include <iostream>

using namespace krpc;

int main() {
   krpc::Client conn = krpc::connect("Example");
   krpc::services::KRPC krpc(&conn);
   std::cout << krpc.get_status().version() << std::endl;
}</pre>
```

This function also accepts arguments that specify what address and port numbers to connect to. For example:

```
#include <krpc.hpp>
#include <krpc/services/krpc.hpp>
#include <iostream>

using namespace krpc;

int main() {
   krpc::Client conn = krpc::connect("Remote example", "my.domain.name", 1000, 1001);
   krpc::services::KRPC krpc(&conn);
   std::cout << krpc.get_status().version() << std::endl;
}</pre>
```

4.1.3 Interacting with the Server

Interaction with the server is performed via a client object (of type krpc::Client) returned by calling krpc::connect().

Functionality for services are defined in the header files in krpc/services/.... For example, all of the functionality provided by the SpaceCenter service is contained in the header file krpc/services/space_center.hpp and the functionality provided by the InfernalRobotics service is contained in krpc/services/infernal_robotics.hpp.

Before a service can be used it must first be instantiated, and passed a copy of the krpc::Client object. Calling methods on the service are mapped to remote procedure calls and passed to the server by the client.

The following example connects to the server, instantiates the SpaceCenter service, and outputs the name of the active vessel:

```
#include <krpc.hpp>
#include <krpc/services/space_center.hpp>
#include <iostream>

using namespace krpc;

int main() {
   krpc::Client conn = krpc::connect("Vessel Name");
   krpc::services::SpaceCenter sc(&conn);
   krpc::services::SpaceCenter::Vessel vessel = sc.active_vessel();
```

```
std::cout << vessel.name() << std::endl;
}</pre>
```

4.1.4 Streaming Data from the Server

A stream repeatedly executes a function on the server, with a fixed set of argument values. It provides a more efficient way of repeatedly getting the result of calling function on the server, without having to invoke it directly – which incurs communication overheads.

For example, consider the following loop that continuously prints out the position of the active vessel. This loop incurs significant communication overheads, as the vessel.position() function is called repeatedly.

The following code achieves the same thing, but is far more efficient. It calls vessel.position_stream once at the start of the program to create a stream, and then repeatedly gets the position from the stream. This avoids the communication overhead in the previous example.

A stream can be created for a function call by adding _stream to the end of the function's name. This returns a stream object of type krpc::Stream, where T is the return type of the original function. The most recent value of the stream can be obtained by calling Stream<T>::operator(). A stream can be stopped by calling krpc::Stream<T>::remove() on the stream object. All streams are automatically stopped when the connection is terminated.

4.1.5 Reference

krpc::*Client* connect (const std::string &name = "", const std::string &address = "127.0.0.1", unsigned int rpc_port = 50000, unsigned int stream_port = 50001)

This function creates a connection to a kRPC server. It returns a krpc::Client object, through which the server can be communicated with.

Parameters

- name (std::string) A descriptive name for the connection. This is passed to the server and appears, for example, in the client connection dialog on the in-game server window.
- address (std::string) The address of the server to connect to. Can either be a hostname or an IP address in dotted decimal notation. Defaults to '127.0.0.1'.
- rpc_port (unsigned int) The port number of the RPC Server. Defaults to 50000.

4.1. C++ Client 89

• **stream_port** (*unsigned int*) – The port number of the Stream Server. Defaults to 50001. Set it to 0 to disable connection to the stream server.

class Client

This class provides the interface for communicating with the server. It is used by service class instances to invoke remote procedure calls. Instances of this class can be obtained by calling krpc::connect().

class KRPC

This class provides access to the basic server functionality provided by the KRPC service. Most of this functionality is used internally by the client (for example to create and remove streams) and therefore does not need to be used directly from application code. The only exception that may be useful is KRPC::get_status().

```
KRPC (krpc::Client *client)
```

Construct an instance of this service from the given krpc::Client object.

```
krpc::schema::Status get_status()
```

Gets a status message from the server containing information including the server's version string and performance statistics.

For example, the following prints out the version string for the server:

```
#include <krpc.hpp>
#include <krpc/services/krpc.hpp>
#include <iostream>

using namespace krpc;

int main() {
    krpc::Client conn = krpc::connect();
    krpc::services::KRPC krpc(&conn);
    std::cout << "Server version = " << krpc.get_status().version() << std::endl;
}</pre>
```

Or to get the rate at which the server is sending and receiving data over the network:

```
#include <krpc.hpp>
#include <krpc/services/krpc.hpp>
#include <iostream>

using namespace krpc;

int main() {
    krpc::Client conn = krpc::connect();
    krpc::services::KRPC krpc(&conn);
    krpc::schema::Status status = krpc.get_status();
    std::cout << "Data in = " << (status.bytes_read_rate()/1024.0) << " KB/s" << std::endl;
    std::cout << "Data out = " << (status.bytes_written_rate()/1024.0) << " KB/s" << std::endl;
}</pre>
```

class AddStream<T>

A stream object. Streams are created by calling a function with _stream appended to its name.

Toperator()()

Get the most recently received value from the stream.

void remove()

Remove the stream from the server.

4.2 SpaceCenter API

4.2.1 SpaceCenter

```
class SpaceCenter : public krpc::Service
     Provides functionality to interact with Kerbal Space Program. This includes controlling the active vessel, man-
     aging its resources, planning maneuver nodes and auto-piloting.
     SpaceCenter (krpc::Client *client)
          Construct an instance of this service.
     Vessel active_vessel()
     void set_active_vessel (Vessel value)
          The currently active vessel.
     std::vector<Vessel> vessels()
          A list of all the vessels in the game.
     std::map<std::string, CelestialBody> bodies ()
          A dictionary of all celestial bodies (planets, moons, etc.) in the game, keyed by the name of the body.
     CelestialBody target_body ()
     void set_target_body (CelestialBody value)
          The currently targeted celestial body.
     Vessel target_vessel()
     void set_target_vessel (Vessel value)
          The currently targeted vessel.
     DockingPort target_docking_port()
     void set_target_docking_port (DockingPort value)
          The currently targeted docking port.
     void clear_target()
          Clears the current target.
     void launch_vessel_from_vab (std::string name)
          Launch a new vessel from the VAB onto the launchpad.
              Parameters
                   • name – Name of the vessel's craft file.
     void launch_vessel_from_sph (std::string name)
          Launch a new vessel from the SPH onto the runway.
              Parameters
                   • name – Name of the vessel's craft file.
     double ut ()
          The current universal time in seconds.
          The value of the gravitational constant G in N(m/kg)^2.
      WarpMode warp_mode()
          The current time warp mode.
                                               Returns WarpMode::none if time warp is not active,
```

WarpMode::rails if regular "on-rails" time warp is active, or WarpMode::physics if physical time warp is active.

float warp_rate()

The current warp rate. This is the rate at which time is passing for either on-rails or physical time warp. For example, a value of 10 means time is passing 10x faster than normal. Returns 1 if time warp is not active.

float warp_factor()

The current warp factor. This is the index of the rate at which time is passing for either regular "on-rails" or physical time warp. Returns 0 if time warp is not active. When in on-rails time warp, this is equal to rails_warp_factor(), and in physics time warp, this is equal to physics_warp_factor().

int32 rails_warp_factor()

```
void set_rails_warp_factor (int32 value)
```

The time warp rate, using regular "on-rails" time warp. A value between 0 and 7 inclusive. 0 means no time warp. Returns 0 if physical time warp is active. If requested time warp factor cannot be set, it will be set to the next lowest possible value. For example, if the vessel is too close to a planet. See the KSP wiki for details.

```
int32 physics_warp_factor()
```

```
void set_physics_warp_factor (int32 value)
```

The physical time warp rate. A value between 0 and 3 inclusive. 0 means no time warp. Returns 0 if regular "on-rails" time warp is active.

```
bool can_rails_warp_at (int32 factor = 1)
```

Returns true if regular "on-rails" time warp can be used, at the specified warp *factor*. The maximum time warp rate is limited by various things, including how close the active vessel is to a planet. See the KSP wiki for details.

Parameters

• factor – The warp factor to check.

```
int32 maximum_rails_warp_factor()
```

The current maximum regular "on-rails" warp factor that can be set. A value between 0 and 7 inclusive. See the KSP wiki for details.

```
void warp_to (double ut, float max\_rails\_rate = 100000.0, float max\_physics\_rate = 2.0)
```

Uses time acceleration to warp forward to a time in the future, specified by universal time *ut*. This call blocks until the desired time is reached. Uses regular "on-rails" or physical time warp as appropriate. For example, physical time warp is used when the active vessel is traveling through an atmosphere. When using regular "on-rails" time warp, the warp rate is limited by *max_rails_rate*, and when using physical time warp, the warp rate is limited by *max_physics_rate*.

Parameters

- **ut** The universal time to warp to, in seconds.
- max_rails_rate The maximum warp rate in regular "on-rails" time warp.
- max_physics_rate The maximum warp rate in physical time warp.

Returns When the time warp is complete.

std::tuple<double, double, double> transform_position (std::tuple<double, double, double> position, ReferenceFrame from, ReferenceFrame to)

Converts a position vector from one reference frame to another.

Parameters

- **position** Position vector in reference frame *from*.
- from The reference frame that the position vector is in.
- to The reference frame to covert the position vector to.

Returns The corresponding position vector in reference frame to.

std::tuple<double, double, double> transform_direction (std::tuple<double, double, double> direction, ReferenceFrame from, ReferenceFrame to)

Converts a direction vector from one reference frame to another.

Parameters

- **direction** Direction vector in reference frame *from*.
- from The reference frame that the direction vector is in.
- **to** The reference frame to covert the direction vector to.

Returns The corresponding direction vector in reference frame to.

std::tuple<double, double, double> transform_rotation (std::tuple<double, double, double, double) ble, double> rotation, Reference-Frame from, ReferenceFrame to)

Converts a rotation from one reference frame to another.

Parameters

- rotation Rotation in reference frame from.
- from The reference frame that the rotation is in.
- to The corresponding rotation in reference frame to.

Returns The corresponding rotation in reference frame to.

std::tuple<double, double, double> transform_velocity (std::tuple<double, double, double> position, std::tuple<double, double, double> velocity, ReferenceFrame from, ReferenceFrame to)

Converts a velocity vector (acting at the specified position vector) from one reference frame to another. The position vector is required to take the relative angular velocity of the reference frames into account.

Parameters

- **position** Position vector in reference frame *from*.
- **velocity** Velocity vector in reference frame *from*.
- from The reference frame that the position and velocity vectors are in.
- to The reference frame to covert the velocity vector to.

Returns The corresponding velocity in reference frame *to*.

bool far_available()

Whether Ferram Aerospace Research is installed.

bool remote tech available()

Whether RemoteTech is installed.

void **draw_direction** (std::tuple<double, double> direction, ReferenceFrame reference_frame, std::tuple<double, double> color, float length = 10.0)

Draw a direction vector on the active vessel.

Parameters

- **direction** Direction to draw the line in.
- **reference_frame** Reference frame that the direction is in.
- color The color to use for the line, as an RGB color.
- length The length of the line. Defaults to 10.

void **draw_line** (std::tuple<double, double> *start*, std::tuple<double, double, double> *end*, *ReferenceFrame reference_frame*, std::tuple<double, double, double> *color*)

Draw a line.

Parameters

- start Position of the start of the line.
- end Position of the end of the line.
- reference_frame Reference frame that the position are in.
- color The color to use for the line, as an RGB color.

```
void clear_drawing()
```

Remove all directions and lines currently being drawn.

enum struct WarpMode

```
Returned by warp_mode()
```

enumerator rails

Time warp is active, and in regular "on-rails" mode.

enumerator physics

Time warp is active, and in physical time warp mode.

enumerator none

Time warp is not active.

4.2.2 Vessel

class Vessel

These objects are used to interact with vessels in KSP. This includes getting orbital and flight data, manipulating control inputs and managing resources.

```
std::string name ()
void set_name (std::string value)
    The name of the vessel.

VesselType type ()
void set_type (VesselType value)
    The type of the vessel.

VesselSituation situation ()
    The situation the vessel is in.
double met ()
    The mission elapsed time in seconds.

Flight flight (ReferenceFrame reference_frame = None)
```

Returns a Flight object that can be used to get flight telemetry for the vessel, in the specified reference frame.

Parameters

• reference_frame - Reference frame. Defaults to the vessel's surface reference frame (Vessel::surface_reference_frame()).

Vessel target()

void set_target (Vessel value)

The target vessel. NULL if there is no target. When setting the target, the target cannot be the current vessel.

Orbit orbit()

The current orbit of the vessel.

Control control ()

Returns a Control object that can be used to manipulate the vessel's control inputs. For example, its pitch/yaw/roll controls, RCS and thrust.

AutoPilot auto_pilot()

An AutoPilot object, that can be used to perform simple auto-piloting of the vessel.

Resources resources ()

A Resources object, that can used to get information about resources stored in the vessel.

Resources resources_in_decouple_stage (int32 stage, bool cumulative = True)

Returns a Resources object, that can used to get information about resources stored in a given stage.

Parameters

- **stage** Get resources for parts that are decoupled in this stage.
- **cumulative** When false, returns the resources for parts decoupled in just the given stage. When true returns the resources decoupled in the given stage and all subsequent stages combined.

Note: For details on stage numbering, see the discussion on *Staging*.

Parts parts ()

A Parts object, that can used to interact with the parts that make up this vessel.

Comms comms ()

A Comms object, that can used to interact with RemoteTech for this vessel.

Note: Requires RemoteTech to be installed.

float mass ()

The total mass of the vessel, including resources, in kg.

float dry_mass()

The total mass of the vessel, excluding resources, in kg.

float thrust ()

The total thrust currently being produced by the vessel's engines, in Newtons. This is computed by summing <code>Engine::thrust()</code> for every engine in the vessel.

float available_thrust()

Gets the total available thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine::available_thrust()</code> for every active engine in the vessel.

float max thrust ()

The total maximum thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine::max_thrust()</code> for every active engine.

float max_vacuum_thrust()

The total maximum thrust that can be produced by the vessel's active engines when the vessel is in a vacuum, in Newtons. This is computed by summing <code>Engine::max_vacuum_thrust()</code> for every active engine.

float specific_impulse()

The combined specific impulse of all active engines, in seconds. This is computed using the formula described here.

float vacuum_specific_impulse()

The combined vacuum specific impulse of all active engines, in seconds. This is computed using the formula described here.

float kerbin_sea_level_specific_impulse()

The combined specific impulse of all active engines at sea level on Kerbin, in seconds. This is computed using the formula described here.

ReferenceFrame reference_frame()

The reference frame that is fixed relative to the vessel, and orientated with the vessel.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel.
- •The x-axis points out to the right of the vessel.
- •The y-axis points in the forward direction of the vessel.
- •The z-axis points out of the bottom off the vessel.

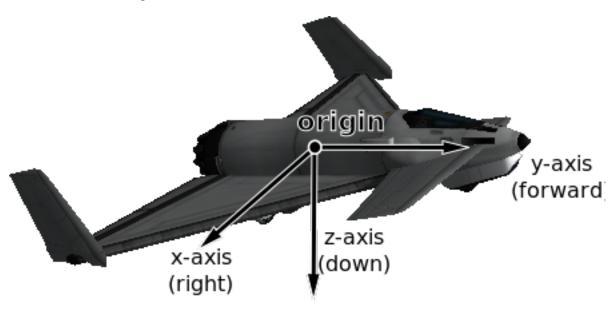


Fig. 4.1: Vessel reference frame origin and axes for the Aeris 3A aircraft

ReferenceFrame orbital_reference_frame()

The reference frame that is fixed relative to the vessel, and orientated with the vessels orbital prograde/normal/radial directions.

•The origin is at the center of mass of the vessel.

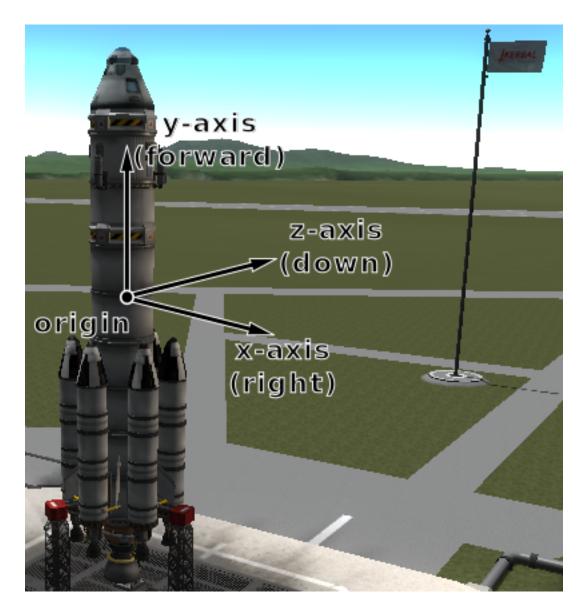


Fig. 4.2: Vessel reference frame origin and axes for the Kerbal-X rocket

- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Note: Be careful not to confuse this with 'orbit' mode on the navball.

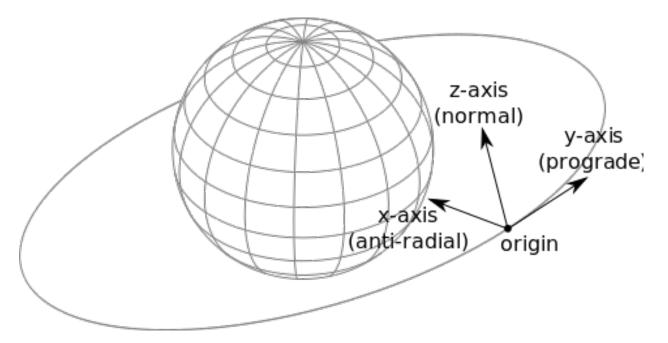


Fig. 4.3: Vessel orbital reference frame origin and axes

ReferenceFrame surface_reference_frame()

The reference frame that is fixed relative to the vessel, and orientated with the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the north and up directions on the surface of the body.
- •The x-axis points in the zenith direction (upwards, normal to the body being orbited, from the center of the body towards the center of mass of the vessel).
- •The y-axis points northwards towards the astronomical horizon (north, and tangential to the surface of the body the direction in which a compass would point when on the surface).
- •The z-axis points eastwards towards the astronomical horizon (east, and tangential to the surface of the body east on a compass when on the surface).

Note: Be careful not to confuse this with 'surface' mode on the navball.

ReferenceFrame surface_velocity_reference_frame()

The reference frame that is fixed relative to the vessel, and orientated with the velocity vector of the vessel relative to the surface of the body being orbited.

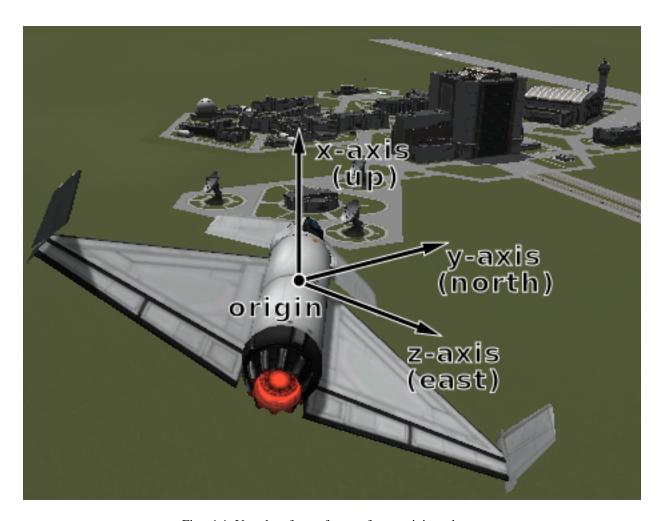


Fig. 4.4: Vessel surface reference frame origin and axes

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel's velocity vector.
- •The y-axis points in the direction of the vessel's velocity vector, relative to the surface of the body being orbited.
- •The z-axis is in the plane of the astronomical horizon.
- •The x-axis is orthogonal to the other two axes.

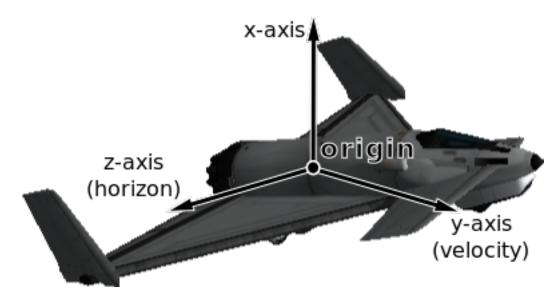


Fig. 4.5: Vessel surface velocity reference frame origin and axes

std::tuple<double, double> **position** (*ReferenceFrame reference_frame*)

Returns the position vector of the center of mass of the vessel in the given reference frame.

Parameters

std::tuple<double, double> **velocity** (*ReferenceFrame reference_frame*)

Returns the velocity vector of the center of mass of the vessel in the given reference frame.

Parameters

std::tuple<double, double, double> rotation (*ReferenceFrame reference_frame*)

Returns the rotation of the center of mass of the vessel in the given reference frame.

Parameters

std::tuple<double, double> **direction** (*ReferenceFrame reference_frame*)

Returns the direction in which the vessel is pointing, as a unit vector, in the given reference frame.

Parameters

std::tuple<double, double, double> angular_velocity (ReferenceFrame reference_frame)

Returns the angular velocity of the vessel in the given reference frame. The magnitude of the returned vector is the rotational speed in radians per second, and the direction of the vector indicates the axis of rotation (using the right hand rule).

Parameters

enum struct VesselType

See Vessel::type().

enumerator ship

Ship.

enumerator station

Station.

enumerator lander

Lander.

enumerator probe

Probe.

enumerator rover

Rover.

enumerator base

Base.

enumerator debris

Debris.

enum struct VesselSituation

See Vessel::situation().

enumerator docked

Vessel is docked to another.

enumerator escaping

Escaping.

enumerator flying

Vessel is flying through an atmosphere.

enumerator landed

Vessel is landed on the surface of a body.

enumerator orbiting

Vessel is orbiting a body.

enumerator pre_launch

Vessel is awaiting launch.

enumerator splashed

Vessel has splashed down in an ocean.

enumerator sub_orbital

Vessel is on a sub-orbital trajectory.

4.2.3 CelestialBody

class CelestialBody

Represents a celestial body (such as a planet or moon).

```
std::string name()
```

The name of the body.

std::vector<CelestialBody> satellites()

A list of celestial bodies that are in orbit around this celestial body.

Orbit orbit()

The orbit of the body.

float mass ()

The mass of the body, in kilograms.

float gravitational_parameter()

The standard gravitational parameter of the body in m^3s^{-2} .

float surface_gravity()

The acceleration due to gravity at sea level (mean altitude) on the body, in m/s^2 .

float rotational period()

The sidereal rotational period of the body, in seconds.

float rotational_speed()

The rotational speed of the body, in radians per second.

float equatorial_radius()

The equatorial radius of the body, in meters.

double surface_height (double latitude, double longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water this is equal to 0.

Parameters

- latitude Latitude in degrees
- **longitude** Longitude in degrees

double bedrock height (double latitude, double longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water, this is the height of the sea-bed and is therefore a negative value.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees

std::tuple<double, double, double>msl_position (double latitude, double longitude, ReferenceFrame reference_frame)

The position at mean sea level at the given latitude and longitude, in the given reference frame.

Parameters

- latitude Latitude in degrees
- **longitude** Longitude in degrees
- reference_frame Reference frame for the returned position vector

std::tuple<double, double, double> **surface_position** (double *latitude*, double *longitude*, *ReferenceFrame* reference frame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position of the surface of the water.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees
- reference_frame Reference frame for the returned position vector

std::tuple<double, double, double> **bedrock_position** (double *latitude*, double *longitude*, *ReferenceFrame*)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position at the bottom of the sea-bed.

Parameters

- latitude Latitude in degrees
- longitude Longitude in degrees
- reference frame Reference frame for the returned position vector

float sphere_of_influence()

The radius of the sphere of influence of the body, in meters.

bool has_atmosphere()

true if the body has an atmosphere.

float atmosphere_depth()

The depth of the atmosphere, in meters.

bool has_atmospheric_oxygen()

true if there is oxygen in the atmosphere, required for air-breathing engines.

ReferenceFrame reference_frame()

The reference frame that is fixed relative to the celestial body.

- •The origin is at the center of the body.
- •The axes rotate with the body.
- •The x-axis points from the center of the body towards the intersection of the prime meridian and equator (the position at 0° longitude, 0° latitude).
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points from the center of the body towards the equator at 90°E longitude.

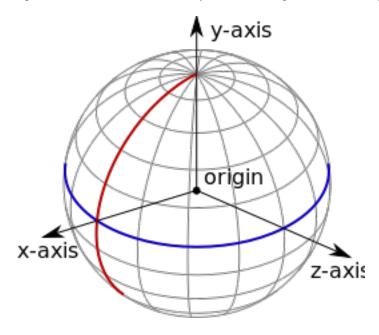


Fig. 4.6: Celestial body reference frame origin and axes. The equator is shown in blue, and the prime meridian in red.

ReferenceFrame non rotating reference frame()

The reference frame that is fixed relative to this celestial body, and orientated in a fixed direction (it does not rotate with the body).

- •The origin is at the center of the body.
- •The axes do not rotate.
- •The x-axis points in an arbitrary direction through the equator.
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points in an arbitrary direction through the equator.

ReferenceFrame orbital_reference_frame()

Gets the reference frame that is fixed relative to this celestial body, but orientated with the body's orbital prograde/normal/radial directions.

- •The origin is at the center of the body.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

std::tuple<double, double> position (*ReferenceFrame reference_frame*)

Returns the position vector of the center of the body in the specified reference frame.

Parameters

std::tuple<double, double, double> velocity (ReferenceFrame reference_frame)

Returns the velocity vector of the body in the specified reference frame.

Parameters

std::tuple<double, double, double, double> rotation (ReferenceFrame reference_frame)

Returns the rotation of the body in the specified reference frame.

Parameters

std::tuple<double, double, double> **direction** (*ReferenceFrame reference_frame*)

Returns the direction in which the north pole of the celestial body is pointing, as a unit vector, in the specified reference frame.

Parameters

std::tuple<double, double, double> angular_velocity (ReferenceFrame reference_frame)

Returns the angular velocity of the body in the specified reference frame. The magnitude of the vector is the rotational speed of the body, in radians per second, and the direction of the vector indicates the axis of rotation, using the right-hand rule.

Parameters

4.2.4 Flight

class Flight

Used to get flight telemetry for a vessel, by calling <code>Vessel::flight()</code>. All of the information returned by this class is given in the reference frame passed to that method.

Note: To get orbital information, such as the apoapsis or inclination, see *Orbit*.

float g_force()

The current G force acting on the vessel in m/s^2 .

double mean altitude()

The altitude above sea level, in meters.

double surface_altitude()

The altitude above the surface of the body or sea level, whichever is closer, in meters.

double bedrock_altitude()

The altitude above the surface of the body, in meters. When over water, this is the altitude above the sea floor.

double **elevation**()

The elevation of the terrain under the vessel, in meters. This is the height of the terrain above sea level, and is negative when the vessel is over the sea.

double latitude()

The latitude of the vessel for the body being orbited, in degrees.

double longitude()

The longitude of the vessel for the body being orbited, in degrees.

std::tuple<double, double, double> velocity()

The velocity vector of the vessel. The magnitude of the vector is the speed of the vessel in meters per second. The direction of the vector is the direction of the vessels motion.

double speed()

The speed of the vessel in meters per second.

double horizontal_speed()

The horizontal speed of the vessel in meters per second.

double vertical_speed()

The vertical speed of the vessel in meters per second.

std::tuple<double, double, double> center_of_mass()

The position of the center of mass of the vessel.

std::tuple<double, double, double> rotation()

The rotation of the vessel.

std::tuple<double, double> direction ()

The direction vector that the vessel is pointing in.

float pitch()

The pitch angle of the vessel relative to the horizon, in degrees. A value between -90° and +90°.

float heading()

The heading angle of the vessel relative to north, in degrees. A value between 0° and 360°.

float roll()

The roll angle of the vessel relative to the horizon, in degrees. A value between -180° and +180°.

std::tuple<double, double> prograde ()

The unit direction vector pointing in the prograde direction.

std::tuple < double, double > retrograde ()

The unit direction vector pointing in the retrograde direction.

std::tuple<double, double, double> normal()

The unit direction vector pointing in the normal direction.

std::tuple<double, double> anti_normal()

The unit direction vector pointing in the anti-normal direction.

std::tuple<double, double, double> radial ()

The unit direction vector pointing in the radial direction.

std::tuple<double, double> anti radial()

The unit direction vector pointing in the anti-radial direction.

float atmosphere_density()

The current density of the atmosphere around the vessel, in kg/m^3 .

float dynamic_pressure()

The dynamic pressure acting on the vessel, in Pascals. This is a measure of the strength of the aerodynamic forces. It is equal to $\frac{1}{2}$ air density velocity². It is commonly denoted as Q.

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

float static_pressure()

The static atmospheric pressure acting on the vessel, in Pascals.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

std::tuple<double, double> aerodynamic_force()

The total aerodynamic forces acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

std::tuple<double, double, double> lift()

The aerodynamic lift currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

std::tuple<double, double, double> drag()

The aerodynamic drag currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

float speed_of_sound()

The speed of sound, in the atmosphere around the vessel, in m/s.

Note: Not available when Ferram Aerospace Research is installed.

float mach ()

The speed of the vessel, in multiples of the speed of sound.

Note: Not available when Ferram Aerospace Research is installed.

float equivalent_air_speed()

The equivalent air speed of the vessel, in m/s.

Note: Not available when Ferram Aerospace Research is installed.

float terminal_velocity()

An estimate of the current terminal velocity of the vessel, in m/s. This is the speed at which the drag forces cancel out the force of gravity.

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

float angle_of_attack()

Gets the pitch angle between the orientation of the vessel and its velocity vector, in degrees.

float sideslip_angle()

Gets the yaw angle between the orientation of the vessel and its velocity vector, in degrees.

float total_air_temperature()

The total air temperature of the atmosphere around the vessel, in Kelvin. This temperature includes the Flight::static_air_temperature() and the vessel's kinetic energy.

float static_air_temperature()

The static (ambient) temperature of the atmosphere around the vessel, in Kelvin.

float stall_fraction()

Gets the current amount of stall, between 0 and 1. A value greater than 0.005 indicates a minor stall and a value greater than 0.5 indicates a large-scale stall.

Note: Requires Ferram Aerospace Research.

float drag coefficient()

Gets the coefficient of drag. This is the amount of drag produced by the vessel. It depends on air speed, air density and wing area.

Note: Requires Ferram Aerospace Research.

float lift_coefficient()

Gets the coefficient of lift. This is the amount of lift produced by the vessel, and depends on air speed, air density and wing area.

Note: Requires Ferram Aerospace Research.

float ballistic coefficient()

Gets the ballistic coefficient.

Note: Requires Ferram Aerospace Research.

float thrust specific fuel consumption()

Gets the thrust specific fuel consumption for the jet engines on the vessel. This is a measure of the efficiency of the engines, with a lower value indicating a more efficient vessel. This value is the number of Newtons of fuel that are burned, per hour, to product one newton of thrust.

Note: Requires Ferram Aerospace Research.

4.2.5 Orbit

class Orbit

Describes an orbit. For example, the orbit of a vessel, obtained by calling <code>Vessel::orbit()</code>, or a celestial body, obtained by calling <code>CelestialBody::orbit()</code>.

CelestialBody body ()

The celestial body (e.g. planet or moon) around which the object is orbiting.

double apoapsis()

Gets the apoapsis of the orbit, in meters, from the center of mass of the body being orbited.

Note: For the apoapsis altitude reported on the in-game map view, use <code>Orbit::apoapsis_altitude()</code>.

double **periapsis**()

The periapsis of the orbit, in meters, from the center of mass of the body being orbited.

Note: For the periapsis altitude reported on the in-game map view, use Orbit::periapsis_altitude().

double apoapsis_altitude()

The apoapsis of the orbit, in meters, above the sea level of the body being orbited.

Note: This is equal to Orbit::apoapsis() minus the equatorial radius of the body.

double periapsis_altitude()

The periapsis of the orbit, in meters, above the sea level of the body being orbited.

Note: This is equal to <code>Orbit::periapsis()</code> minus the equatorial radius of the body.

double semi_major_axis()

The semi-major axis of the orbit, in meters.

double semi_minor_axis()

The semi-minor axis of the orbit, in meters.

double radius ()

The current radius of the orbit, in meters. This is the distance between the center of mass of the object in orbit, and the center of mass of the body around which it is orbiting.

Note: This value will change over time if the orbit is elliptical.

double **speed**()

The current orbital speed of the object in meters per second.

Note: This value will change over time if the orbit is elliptical.

double **period**()

The orbital period, in seconds.

double time_to_apoapsis()

The time until the object reaches apoapsis, in seconds.

double time_to_periapsis()

The time until the object reaches periapsis, in seconds.

double eccentricity()

The eccentricity of the orbit.

double inclination()

The inclination of the orbit, in radians.

double longitude_of_ascending_node()

The longitude of the ascending node, in radians.

double argument_of_periapsis()

The argument of periapsis, in radians.

double mean_anomaly_at_epoch()

The mean anomaly at epoch.

double epoch ()

The time since the epoch (the point at which the mean anomaly at epoch was measured, in seconds.

double mean_anomaly()

The mean anomaly.

$double\ \textbf{eccentric_anomaly}\ (\)$

The eccentric anomaly.

static std::tuple<double, double> reference_plane_normal (ReferenceFrame

ence_frame)

The unit direction vector that is normal to the orbits reference plane, in the given reference frame. The reference plane is the plane from which the orbits inclination is measured.

Parameters

static std::tuple<double, double> reference_plane_direction (ReferenceFrame refer

ence frame)

The unit direction vector from which the orbits longitude of ascending node is measured, in the given reference frame.

Parameters

refer-

```
double time to soi change()
```

The time until the object changes sphere of influence, in seconds. Returns NaN if the object is not going to change sphere of influence.

```
Orbit next_orbit()
```

If the object is going to change sphere of influence in the future, returns the new orbit after the change. Otherwise returns NULL.

4.2.6 Control

class Control

Used to manipulate the controls of a vessel. This includes adjusting the throttle, enabling/disabling systems such as SAS and RCS, or altering the direction in which the vessel is pointing.

Note: Control inputs (such as pitch, yaw and roll) are zeroed when all clients that have set one or more of these inputs are no longer connected.

```
bool sas()
void set_sas(bool value)
The state of SAS.
```

Note: Equivalent to AutoPilot::sas()

```
SASMode sas_mode()
```

void set_sas_mode (SASMode value)

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Note: Equivalent to AutoPilot::sas_mode()

```
SpeedMode speed_mode()
```

```
void set_speed_mode (SpeedMode value)
```

The current SpeedMode of the navball. This is the mode displayed next to the speed at the top of the navball.

```
bool rcs()
void set_rcs(bool value)
    The state of RCS.
bool gear()
void set_gear(bool value)
    The state of the landing gear/legs.
bool lights()
void set_lights(bool value)
    The state of the lights.
bool brakes()
```

```
void set brakes (bool value)
     The state of the wheel brakes.
bool abort ()
void set abort (bool value)
     The state of the abort action group.
float throttle()
void set_throttle (float value)
     The state of the throttle. A value between 0 and 1.
float pitch()
void set_pitch (float value)
     The state of the pitch control. A value between -1 and 1. Equivalent to the w and s keys.
float yaw ()
void set_yaw (float value)
     The state of the yaw control. A value between -1 and 1. Equivalent to the a and d keys.
float roll()
void set_roll (float value)
     The state of the roll control. A value between -1 and 1. Equivalent to the q and e keys.
float forward()
void set forward (float value)
     The state of the forward translational control. A value between -1 and 1. Equivalent to the h and n keys.
float up ()
void set_up (float value)
     The state of the up translational control. A value between -1 and 1. Equivalent to the i and k keys.
float right ()
void set_right (float value)
     The state of the right translational control. A value between -1 and 1. Equivalent to the j and l keys.
float wheel throttle()
void set wheel throttle (float value)
     The state of the wheel throttle. A value between -1 and 1. A value of 1 rotates the wheels forwards, a value
     of -1 rotates the wheels backwards.
float wheel steering()
void set_wheel_steering (float value)
     The state of the wheel steering. A value between -1 and 1. A value of 1 steers to the left, and a value of -1
     steers to the right.
int32 current_stage()
     The current stage of the vessel. Corresponds to the stage number in the in-game UI.
std::vector<Vessel> activate_next_stage()
     Activates the next stage. Equivalent to pressing the space bar in-game.
         Returns A list of vessel objects that are jettisoned from the active vessel.
bool get_action_group (uint32 group)
     Returns true if the given action group is enabled.
```

Parameters

• group – A number between 0 and 9 inclusive.

void set_action_group (uint32 group, bool state)

Sets the state of the given action group (a value between 0 and 9 inclusive).

Parameters

• group – A number between 0 and 9 inclusive.

void toggle_action_group (uint32 group)

Toggles the state of the given action group.

Parameters

• group – A number between 0 and 9 inclusive.

Node add_node (double ut, float prograde = 0.0, float normal = 0.0, float radial = 0.0)

Creates a maneuver node at the given universal time, and returns a *Node* object that can be used to modify it. Optionally sets the magnitude of the delta-v for the maneuver node in the prograde, normal and radial directions.

Parameters

- **ut** Universal time of the maneuver node.
- **prograde** Delta-v in the prograde direction.
- **normal** Delta-v in the normal direction.
- radial Delta-v in the radial direction.

std::vector<*Node*> nodes()

Returns a list of all existing maneuver nodes, ordered by time from first to last.

void remove nodes()

Remove all maneuver nodes.

enum struct SASMode

The behavior of the SAS auto-pilot. See AutoPilot::sas_mode().

enumerator stability_assist

Stability assist mode. Dampen out any rotation.

enumerator maneuver

Point in the burn direction of the next maneuver node.

enumerator prograde

Point in the prograde direction.

enumerator retrograde

Point in the retrograde direction.

enumerator normal

Point in the orbit normal direction.

enumerator anti normal

Point in the orbit anti-normal direction.

enumerator radial

Point in the orbit radial direction.

enumerator anti radial

Point in the orbit anti-radial direction.

enumerator target

Point in the direction of the current target.

enumerator anti_target

Point away from the current target.

enum struct SpeedMode

```
See Control::speed_mode().
```

enumerator orbit

Speed is relative to the vessel's orbit.

enumerator surface

Speed is relative to the surface of the body being orbited.

enumerator target

Speed is relative to the current target.

4.2.7 Parts

The following classes allow interaction with a vessels individual parts.

- Parts
- Part
- Module
- Specific Types of Part
 - Decoupler
 - Docking Port
 - Engine
 - Landing Gear
 - Landing Leg
 - Launch Clamp
 - Light
 - Parachute
 - Radiator
 - Resource Converter
 - Resource Harvester
 - Reaction Wheel
 - Sensor
 - Solar Panel
- Trees of Parts
 - Traversing the Tree
 - Attachment Modes
- Fuel Lines
- Staging

Parts

class Parts

Instances of this class are used to interact with the parts of a vessel. An instance can be obtained by calling <code>Vessel::parts()</code>.

```
std::vector<Part> all()
```

A list of all of the vessels parts.

```
Part root()
     The vessels root part.
     Note: See the discussion on Trees of Parts.
Part controlling()
void set_controlling (Part value)
     The part from which the vessel is controlled.
std::vector<Part> with_name (std::string name)
     A list of parts whose Part::name() is name.
         Parameters
std::vector<Part> with_title (std::string title)
     A list of all parts whose Part::title() is title.
         Parameters
std::vector<Part> with_module (std::string module_name)
     A list of all parts that contain a Module whose Module::name() is module_name.
         Parameters
std::vector<Part> in_stage (int32 stage)
     A list of all parts that are activated in the given stage.
         Parameters
     Note: See the discussion on Staging.
std::vector<Part> in_decouple_stage (int32 stage)
     A list of all parts that are decoupled in the given stage.
         Parameters
     Note: See the discussion on Staging.
std::vector<Module> modules_with_name (std::string module_name)
     A list of modules (combined across all parts in the vessel) whose Module::name() is module_name.
         Parameters
std::vector<Decoupler> decouplers()
     A list of all decouplers in the vessel.
std::vector<DockingPort> docking_ports()
     A list of all docking ports in the vessel.
DockingPort docking_port_with_name (std::string name)
     The first docking port in the vessel with the given port name, as returned by DockingPort::name().
     Returns NULL if there are no such docking ports.
         Parameters
std::vector<Engine> engines()
     A list of all engines in the vessel.
```

```
std::vector<LandingGear> landing_gear()
     A list of all landing gear attached to the vessel.
std::vector<LandingLeg> landing_legs()
     A list of all landing legs attached to the vessel.
std::vector<LaunchClamp> launch_clamps ()
     A list of all launch clamps attached to the vessel.
std::vector<Light> lights()
     A list of all lights in the vessel.
std::vector<Parachute> parachutes ()
     A list of all parachutes in the vessel.
std::vector<Radiator> radiators()
     A list of all radiators in the vessel.
std::vector<ResourceConverter> resource_converters()
     A list of all resource converters in the vessel.
std::vector<ResourceHarvester> resource harvesters()
     A list of all resource harvesters in the vessel.
std::vector<ReactionWheel> reaction_wheels()
     A list of all reaction wheels in the vessel.
std::vector<Sensor> sensors()
     A list of all sensors in the vessel.
std::vector<SolarPanel> solar_panels ()
     A list of all solar panels in the vessel.
```

Part

class Part

Instances of this class represents a part. A vessel is made of multiple parts. Instances can be obtained by various methods in *Parts*.

std::string name()

Internal name of the part, as used in part cfg files. For example "Mark1-2Pod".

std::string title()

Title of the part, as shown when the part is right clicked in-game. For example "Mk1-2 Command Pod".

double cost ()

The cost of the part, in units of funds.

Vessel vessel()

The vessel that contains this part.

Part parent ()

The parts parent. Returns NULL if the part does not have a parent. This, in combination with <code>Part::children()</code>, can be used to traverse the vessels parts tree.

Note: See the discussion on *Trees of Parts*.

std::vector<*Part*> children()

The parts children. Returns an empty list if the part has no children. This, in combination with <code>Part::parent()</code>, can be used to traverse the vessels parts tree.

Note: See the discussion on *Trees of Parts*.

bool axially_attached()

Whether the part is axially attached to its parent, i.e. on the top or bottom of its parent. If the part has no parent, returns false.

Note: See the discussion on *Attachment Modes*.

bool radially_attached()

Whether the part is radially attached to its parent, i.e. on the side of its parent. If the part has no parent, returns false.

Note: See the discussion on *Attachment Modes*.

int32 stage()

The stage in which this part will be activated. Returns -1 if the part is not activated by staging.

Note: See the discussion on *Staging*.

int32 decouple_stage()

The stage in which this part will be decoupled. Returns -1 if the part is never decoupled from the vessel.

Note: See the discussion on Staging.

bool massless()

Whether the part is massless.

double mass ()

The current mass of the part, including resources it contains, in kilograms. Returns zero if the part is massless.

double dry_mass()

The mass of the part, not including any resources it contains, in kilograms. Returns zero if the part is massless.

double impact_tolerance()

The impact tolerance of the part, in meters per second.

double temperature ()

Temperature of the part, in Kelvin.

double skin_temperature()

Temperature of the skin of the part, in Kelvin.

double max_temperature()

Maximum temperature that the part can survive, in Kelvin.

double max_skin_temperature()

Maximum temperature that the skin of the part can survive, in Kelvin.

double external temperature()

Temperature of the atmosphere/vacuum surrounding the part, in Kelvin. This does not include heating from direct sunlight.

double thermal_mass()

How much it takes to heat up the part.

double thermal skin mass()

How much it takes to heat up the part's skin.

double thermal_resource_mass()

How much it takes to heat up resources in the part.

double thermal_conduction_flux()

The speed that heat is conducting into or out of the part through contact with other parts. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double thermal_convection_flux()

The speed that heat is convecting into or out of the part from the surrounding atmosphere. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double thermal_radiation_flux()

The speed that heat is radiating into or out of the part from the surrounding vacuum. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double thermal_internal_flux()

The speed that heat is generated by the part. For example, engines generate heat by burning fuel. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double thermal_skin_to_internal_flux()

The speed that heat is conducting between the part's skin and its internals.

Resources resources ()

A Resources object for the part.

bool crossfeed()

Whether this part is crossfeed capable.

bool is_fuel_line()

Whether this part is a fuel line.

std::vector<Part> fuel_lines_from()

The parts that are connected to this part via fuel lines, where the direction of the fuel line is into this part.

Note: See the discussion on *Fuel Lines*.

std::vector<*Part*> fuel_lines_to()

The parts that are connected to this part via fuel lines, where the direction of the fuel line is out of this part.

Note: See the discussion on *Fuel Lines*.

std::vector< Module > modules ()

The modules for this part.

Decoupler decoupler()

A Decoupler if the part is a decoupler, otherwise NULL.

DockingPort docking_port()

A *DockingPort* if the part is a docking port, otherwise NULL.

Engine engine () An *Engine* if the part is an engine, otherwise NULL. LandingGear landing_gear() A LandingGear if the part is a landing gear, otherwise NULL. LandingLeg landing leg() A LandingLeg if the part is a landing leg, otherwise NULL. LaunchClamp launch clamp() A LaunchClamp if the part is a launch clamp, otherwise NULL. Light light() A *Light* if the part is a light, otherwise NULL. Parachute parachute() A *Parachute* if the part is a parachute, otherwise NULL. Radiator radiator() A Radiator if the part is a radiator, otherwise NULL. ReactionWheel reaction wheel() A ReactionWheel if the part is a reaction wheel, otherwise NULL. ResourceConverter resource_converter() A ResourceConverter if the part is a resource converter, otherwise NULL. ResourceHarvester resource harvester() A ResourceHarvester if the part is a resource harvester, otherwise NULL. Sensor sensor() A Sensor if the part is a sensor, otherwise NULL. SolarPanel solar_panel() A SolarPanel if the part is a solar panel, otherwise NULL. std::tuple<double, double, double> position (ReferenceFrame reference_frame) The position of the part in the given reference frame. **Parameters**

std::tuple<double, double> direction (*ReferenceFrame reference_frame*)

The direction of the part in the given reference frame.

Parameters

std::tuple<double, double> **velocity** (*ReferenceFrame reference_frame*)

The velocity of the part in the given reference frame.

Parameters

std::tuple<double, double, double> **rotation** (*ReferenceFrame reference_frame*)

The rotation of the part in the given reference frame.

Parameters

ReferenceFrame reference_frame()

The reference frame that is fixed relative to this part.

- •The origin is at the position of the part.
- •The axes rotate with the part.
- •The x, y and z axis directions depend on the design of the part.

Note: For docking port parts, this reference frame is not necessarily equivalent to the reference frame for the docking port, returned by <code>DockingPort::reference_frame()</code>.

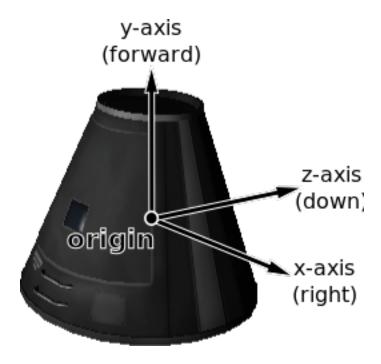


Fig. 4.7: Mk1 Command Pod reference frame origin and axes

Module

class Module

In KSP, each part has zero or more PartModules associated with it. Each one contains some of the functionality of the part. For example, an engine has a "ModuleEngines" PartModule that contains all the functionality of an engine. This class allows you to interact with KSPs PartModules, and any PartModules that have been added by other mods.

std::string name()

Name of the PartModule. For example, "ModuleEngines".

Part part ()

The part that contains this module.

std::map<std::string, std::string> fields ()

The modules field names and their associated values, as a dictionary. These are the values visible in the right-click menu of the part.

bool has field (std::string name)

Returns true if the module has a field with the given name.

Parameters

• name – Name of the field.

std::string get_field(std::string name)

Returns the value of a field.

Parameters

• name – Name of the field.

std::vector<std::string> events()

A list of the names of all of the modules events. Events are the clickable buttons visible in the right-click menu of the part.

bool has_event (std::string name)

true if the module has an event with the given name.

Parameters

```
void trigger_event (std::string name)
```

Trigger the named event. Equivalent to clicking the button in the right-click menu of the part.

Parameters

```
std::vector<std::string> actions()
```

A list of all the names of the modules actions. These are the parts actions that can be assigned to action groups in the in-game editor.

```
bool has_action (std::string name)
```

true if the part has an action with the given name.

Parameters

void set_action (std::string name, bool value = True)

Set the value of an action with the given name.

Parameters

Specific Types of Part

The following classes provide functionality for specific types of part.

- Decoupler
- Docking Port
- Engine
- Landing Gear
- Landing Leg
- Launch Clamp
- Light
- Parachute
- Radiator
- Resource Converter
- Resource Harvester
- Reaction Wheel
- Sensor
- Solar Panel

Decoupler

class Decoupler

Obtained by calling Part::decoupler()

Part part ()

The part object for this decoupler.

void decouple()

Fires the decoupler. Has no effect if the decoupler has already fired.

bool decoupled()

Whether the decoupler has fired.

float impulse()

The impulse that the decoupler imparts when it is fired, in Newton seconds.

Docking Port

class DockingPort

Obtained by calling Part::docking_port()

Part part ()

The part object for this docking port.

```
std::string name ()
```

```
void set_name (std::string value)
```

The port name of the docking port. This is the name of the port that can be set in the right click menu, when the Docking Port Alignment Indicator mod is installed. If this mod is not installed, returns the title of the part (Part::title()).

DockingPortState state()

The current state of the docking port.

Part docked_part()

The part that this docking port is docked to. Returns NULL if this docking port is not docked to anything.

Vessel undock ()

Undocks the docking port and returns the vessel that was undocked from. After undocking, the active vessel may change (active_vessel()). This method can be called for either docking port in a docked pair - both calls will have the same effect. Returns NULL if the docking port is not docked to anything.

float reengage_distance()

The distance a docking port must move away when it undocks before it becomes ready to dock with another port, in meters.

bool has_shield()

Whether the docking port has a shield.

bool shielded()

void set_shielded (bool value)

The state of the docking ports shield, if it has one. Returns true if the docking port has a shield, and the shield is closed. Otherwise returns false. When set to true, the shield is closed, and when set to false the shield is opened. If the docking port does not have a shield, setting this attribute has no effect.

std::tuple<double, double, double> position (ReferenceFrame reference_frame)

The position of the docking port in the given reference frame.

Parameters

std::tuple<double, double, double> direction (ReferenceFrame reference_frame)

The direction that docking port points in, in the given reference frame.

Parameters

std::tuple<double, double, double> rotation (*ReferenceFrame reference_frame*)

The rotation of the docking port, in the given reference frame.

Parameters

ReferenceFrame reference_frame()

The reference frame that is fixed relative to this docking port, and oriented with the port.

- •The origin is at the position of the docking port.
- •The axes rotate with the docking port.
- •The x-axis points out to the right side of the docking port.
- •The y-axis points in the direction the docking port is facing.
- •The z-axis points out of the bottom off the docking port.

Note: This reference frame is not necessarily equivalent to the reference frame for the part, returned by $Part::reference_frame()$.

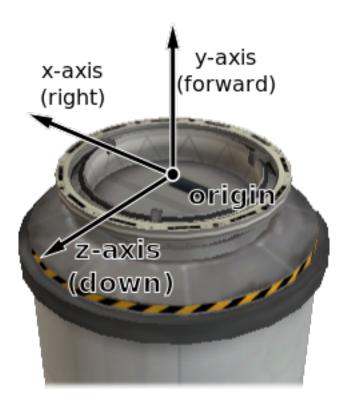


Fig. 4.8: Docking port reference frame origin and axes

enum struct DockingPortState

See DockingPort::state().

enumerator ready

The docking port is ready to dock to another docking port.

enumerator docked

The docking port is docked to another docking port, or docked to another part (from the VAB/SPH).

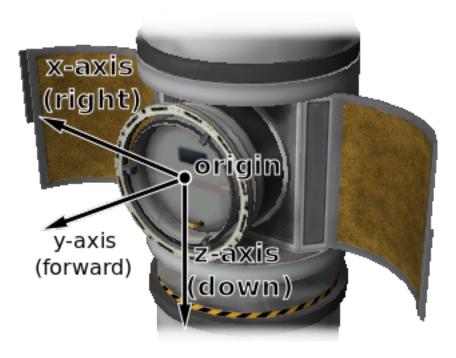


Fig. 4.9: Inline docking port reference frame origin and axes

enumerator docking

The docking port is very close to another docking port, but has not docked. It is using magnetic force to acquire a solid dock.

enumerator undocking

The docking port has just been undocked from another docking port, and is disabled until it moves away by a sufficient distance (DockingPort::reengage_distance()).

enumerator shielded

The docking port has a shield, and the shield is closed.

enumerator moving

The docking ports shield is currently opening/closing.

Engine

class Engine

Obtained by calling Part::engine().

Part part ()

The part object for this engine.

bool active()

void set_active (bool value)

Whether the engine is active. Setting this attribute may have no effect, depending on <code>Engine::can_shutdown()</code> and <code>Engine::can_restart()</code>.

float thrust ()

The current amount of thrust being produced by the engine, in Newtons. Returns zero if the engine is not active or if it has no fuel.

float available thrust()

The maximum available amount of thrust that can be produced by the engine, in Newtons. This takes <code>Engine::thrust_limit()</code> into account, and is the amount of thrust produced by the engine when activated and the main throttle is set to 100%. Returns zero if the engine does not have any fuel.

float max thrust ()

Gets the maximum amount of thrust that can be produced by the engine, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine::thrust_limit()</code> is set to 100% and the main vessel's throttle is set to 100%.

float max_vacuum_thrust()

The maximum amount of thrust that can be produced by the engine in a vacuum, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine::thrust_limit()</code> is set to 100%, the main vessel's throttle is set to 100% and the engine is in a vacuum.

float thrust limit()

void set_thrust_limit (float value)

The thrust limiter of the engine. A value between 0 and 1. Setting this attribute may have no effect, for example the thrust limit for a solid rocket booster cannot be changed in flight.

float specific_impulse()

The current specific impulse of the engine, in seconds. Returns zero if the engine is not active.

float vacuum specific impulse()

The vacuum specific impulse of the engine, in seconds.

float kerbin_sea_level_specific_impulse()

The specific impulse of the engine at sea level on Kerbin, in seconds.

std::vector<std::string> propellants ()

The names of resources that the engine consumes.

std::map<std::string, float> propellant_ratios ()

The ratios of resources that the engine consumes. A dictionary mapping resource names to the ratios at which they are consumed by the engine.

bool has_fuel()

Whether the engine has run out of fuel (or flamed out).

float throttle()

The current throttle setting for the engine. A value between 0 and 1. This is not necessarily the same as the vessel's main throttle setting, as some engines take time to adjust their throttle (such as jet engines).

bool throttle locked()

Whether the <code>Control::throttle()</code> affects the engine. For example, this is true for liquid fueled rockets, and false for solid rocket boosters.

bool can_restart()

Whether the engine can be restarted once shutdown. If the engine cannot be shutdown, returns false. For example, this is true for liquid fueled rockets and false for solid rocket boosters.

bool can_shutdown()

Gets whether the engine can be shutdown once activated. For example, this is true for liquid fueled rockets and false for solid rocket boosters.

bool gimballed()

Whether the engine nozzle is gimballed, i.e. can provide a turning force.

float gimbal_range()

The range over which the gimbal can move, in degrees.

```
bool gimbal_locked()
     void set_gimbal_locked (bool value)
          Whether the engines gimbal is locked in place. Setting this attribute has no effect if the engine is not
          gimballed.
     float gimbal_limit()
     void set_gimbal_limit (float value)
          The gimbal limiter of the engine. A value between 0 and 1. Returns 0 if the gimbal is locked or the engine
          is not gimballed. Setting this attribute has no effect if the engine is not gimballed.
Landing Gear
class LandingGear
     Obtained by calling Part::landing_gear().
     Part part ()
          The part object for this landing gear.
     LandingGearState state()
          Gets the current state of the landing gear.
     bool deployed()
     void set_deployed (bool value)
          Whether the landing gear is deployed.
enum struct LandingGearState
     See LandingGear::state().
     enumerator deployed
          Landing gear is fully deployed.
     enumerator retracted
          Landing gear is fully retracted.
     enumerator deploying
          Landing gear is being deployed.
     enumerator retracting
          Landing gear is being retracted.
Landing Leg
class LandingLeg
     Obtained by calling Part::landing_leg().
     Part part ()
          The part object for this landing leg.
     LandingLegState state()
          The current state of the landing leg.
     bool deployed()
     void set_deployed (bool value)
          Whether the landing leg is deployed.
enum struct LandingLegState
```

See LandingLeg::state().

```
enumerator deployed
          Landing leg is fully deployed.
     enumerator retracted
          Landing leg is fully retracted.
     enumerator deploying
          Landing leg is being deployed.
     enumerator retracting
          Landing leg is being retracted.
     enumerator broken
          Landing leg is broken.
     enumerator repairing
          Landing leg is being repaired.
Launch Clamp
class LaunchClamp
     Obtained by calling Part::launch_clamp().
     Part part ()
          The part object for this launch clamp.
     void release()
          Releases the docking clamp. Has no effect if the clamp has already been released.
Light
class Light
     Obtained by calling Part::light().
     Part part ()
          The part object for this light.
     bool active()
     void set_active (bool value)
          Whether the light is switched on.
     float power_usage()
          The current power usage, in units of charge per second.
Parachute
class Parachute
     Obtained by calling Part::parachute().
     Part part ()
          The part object for this parachute.
     void deploy()
          Deploys the parachute. This has no effect if the parachute has already been deployed.
     bool deployed()
          Whether the parachute has been deployed.
```

```
ParachuteState state()
          The current state of the parachute.
     float deploy_altitude()
     void set_deploy_altitude (float value)
          The altitude at which the parachute will full deploy, in meters.
     float deploy_min_pressure()
     void set_deploy_min_pressure (float value)
          The minimum pressure at which the parachute will semi-deploy, in atmospheres.
enum struct ParachuteState
     See Parachute::state().
     enumerator stowed
          The parachute is safely tucked away inside its housing.
     enumerator active
          The parachute is still stowed, but ready to semi-deploy.
     enumerator semi_deployed
          The parachute has been deployed and is providing some drag, but is not fully deployed yet.
     enumerator deployed
          The parachute is fully deployed.
     enumerator cut
          The parachute has been cut.
Radiator
class Radiator
     Obtained by calling Part::radiator().
     Part part ()
          The part object for this radiator.
     bool deployed()
     void set_deployed (bool value)
          Whether the radiator is extended.
     RadiatorState state()
          The current state of the radiator.
enum struct RadiatorState
     RadiatorState
     enumerator extended
          Radiator is fully extended.
     enumerator retracted
          Radiator is fully retracted.
     enumerator extending
          Radiator is being extended.
```

enumerator retracting

Radiator is being retracted.

enumerator broken

Radiator is being broken.

Resource Converter

class ResourceConverter

Obtained by calling Part::resource_converter().

Part part ()

The part object for this converter.

int32 count ()

The number of converters in the part.

std::string name (int32 index)

The name of the specified converter.

Parameters

• index – Index of the converter.

bool active (int32 index)

True if the specified converter is active.

Parameters

• index – Index of the converter.

void start (int32 index)

Start the specified converter.

Parameters

• index – Index of the converter.

void stop (int32 index)

Stop the specified converter.

Parameters

• index – Index of the converter.

ResourceConverterState state (int32 index)

The state of the specified converter.

Parameters

• index – Index of the converter.

std::string status_info (int32 index)

Status information for the specified converter. This is the full status message shown in the in-game UI.

Parameters

• index – Index of the converter.

std::vector<std::string> inputs (int32 index)

List of the names of resources consumed by the specified converter.

Parameters

• index – Index of the converter.

std::vector<std::string> outputs (int32 index)

List of the names of resources produced by the specified converter.

Parameters

• index – Index of the converter.

enum struct ResourceConverterState

See ResourceConverter::state().

enumerator running

Converter is running.

enumerator idle

Converter is idle.

enumerator missing_resource

Converter is missing a required resource.

enumerator storage_full

No available storage for output resource.

enumerator capacity

At preset resource capacity.

enumerator unknown

Unknown state. Possible with modified resource converters. In this case, check ResourceConverter::status_info() for more information.

Resource Harvester

class ResourceHarvester

Obtained by calling Part::resource_harvester().

Part part ()

The part object for this harvester.

ResourceHarvesterState state()

The state of the harvester.

bool deployed()

void set_deployed (bool value)

Whether the harvester is deployed.

bool active()

void set_active (bool value)

Whether the harvester is actively drilling.

float extraction_rate()

The rate at which the drill is extracting ore, in units per second.

float thermal_efficiency()

The thermal efficiency of the drill, as a percentage of its maximum.

float core_temperature()

The core temperature of the drill, in Kelvin.

float optimum_core_temperature()

The core temperature at which the drill will operate with peak efficiency, in Kelvin.

Reaction Wheel

```
class ReactionWheel
     Obtained by calling Part::reaction_wheel().
     Part part ()
          The part object for this reaction wheel.
     bool active()
     void set active (bool value)
          Whether the reaction wheel is active.
     bool broken()
          Whether the reaction wheel is broken.
     float pitch_torque()
          The torque in the pitch axis, in Newton meters.
     float yaw_torque()
          The torque in the yaw axis, in Newton meters.
     float roll_torque()
          The torque in the roll axis, in Newton meters.
Sensor
class Sensor
     Obtained by calling Part::sensor().
     Part part()
          The part object for this sensor.
     bool active()
     void set_active (bool value)
          Whether the sensor is active.
     std::string value()
          The current value of the sensor.
     float power_usage()
          The current power usage of the sensor, in units of charge per second.
Solar Panel
class SolarPanel
     Obtained by calling Part::solar_panel().
     Part part ()
          The part object for this solar panel.
     bool deployed()
     void set_deployed (bool value)
          Whether the solar panel is extended.
     SolarPanelState state()
          The current state of the solar panel.
```

float energy_flow()

The current amount of energy being generated by the solar panel, in units of charge per second.

float sun_exposure()

The current amount of sunlight that is incident on the solar panel, as a percentage. A value between 0 and 1

enum struct SolarPanelState

See SolarPanel::state().

enumerator extended

Solar panel is fully extended.

enumerator retracted

Solar panel is fully retracted.

enumerator extending

Solar panel is being extended.

enumerator retracting

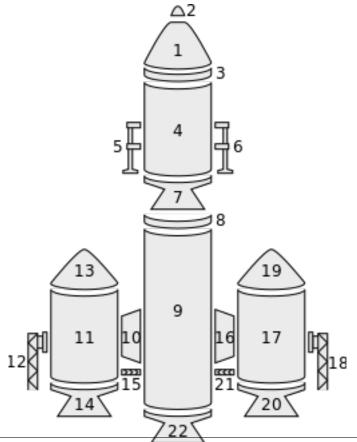
Solar panel is being retracted.

enumerator broken

Solar panel is broken.

Trees of Parts

Vessels in KSP are comprised of a number of parts, connected to one another in a tree structure. An example vessel is shown in Figure 1, and the corresponding tree of parts in Figure 2. The craft file for this example can also be downloaded here.



Traversing the Tree

The tree of parts can be traversed using the attributes SpaceCenter::Parts::root(), SpaceCenter::Part::parent() and SpaceCenter::Part::children().

The root of the tree is the same as the vessels root part (part number 1 in the example above) and can be obtained by calling SpaceCenter::Parts::root(). parts children can be obtained by calling SpaceCenter::Part::children(). If the part does not have any children, SpaceCenter::Part::children() returns an empty list. A parts can be obtained by calling SpaceCenter::Part::parent(). If the part does not have a parent (as is the case for the root part), SpaceCenter::Part::parent() returns NULL.

4.2. SpaceCenter API

131

Fig. 4.10: **Figure 1** – Example parts making up a vessel.

The following C++ example uses these attributes to perform a depth-first traversal over all of the parts in a vessel:

```
#include <krpc.hpp>
#include <krpc/services/space_center.hpp>
#include <iostream>
#include <stack>
using namespace krpc;
using namespace krpc::services;
int main() {
  Client conn = krpc::connect("");
  SpaceCenter sc(&conn);
  auto vessel = sc.active_vessel();
  auto root = vessel.parts().root();
  std::stack<std::pair<SpaceCenter::Part,int> >
  stack.push(std::pair<SpaceCenter::Part,int>(red)
  while (stack.size() > 0) {
   auto part = stack.top().first;
    auto depth = stack.top().second;
    stack.pop();
    std::cout << std::string(depth,</pre>
                                     ' ') << part
    auto children = part.children();
    for (std::vector<SpaceCenter::Part>::iterate
      stack.push(std::pair<SpaceCenter::Part, in
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1
TR-18A Stack Decoupler
 FL-T400 Fuel Tank
  LV-909 Liquid Fuel Engine
   TR-18A Stack Decoupler
    FL-T800 Fuel Tank
     LV-909 Liquid Fuel Engine
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
        TT18-A Launch Stability Enhancer
       FTX-2 External Fuel Duct
       LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
       TT18-A Launch Stability Enhancer
       FTX-2 External Fuel Duct
        LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
  LT-1 Landing Struts
  LT-1 Landing Struts
Mk16 Parachute
```

Attachment Modes

Parts can be attached to other parts either *radially* (on the side of the parent part) or *axially* (on the end of the parent part, to form a stack).

For example, in the vessel pictured above, the parachute (part 2) is *axially* connected to its parent (the command pod – part 1), and the landing leg (part 5) is *radially* connected to its parent (the fuel tank – part 4).

The root part of a vessel (for example the command pod – part 1) does not have a parent part, so does not have an attachment mode. However, the part is consider to be *axially* attached to nothing.

The following C++ example does a depth-first traversal as before, but also prints out the attachment mode used by the part:

```
1
Δ2
3
```

```
#include <krpc.hpp>
#include <krpc/services/space_center.hpp>
#include <iostream>
#include <stack>
using namespace krpc;
using namespace krpc::services;
int main() {
 Client conn = krpc::connect("");
  SpaceCenter sc(&conn);
  auto vessel = sc.active_vessel();
  auto root = vessel.parts().root();
  std::stack<std::pair<SpaceCenter::Part,int> > stack;
  stack.push(std::pair<SpaceCenter::Part,int>(root, 0));
  while (stack.size() > 0) {
   auto part = stack.top().first;
   auto depth = stack.top().second;
   stack.pop();
   std::string attach_mode;
   if (part.axially_attached()) {
      attach_mode = "axial";
    } else { // radially_attached
      attach_mode = "radial";
    std::cout << std::string(depth, ' ') << part.title() << " - " << attach_mode << std::endl;</pre>
    auto children = part.children();
    for (std::vector<SpaceCenter::Part>::iterator child = children.begin(); child != children.end();
      stack.push(std::pair<SpaceCenter::Part,int>(*child, depth+1));
  }
```

point from the parent part to the child part.

When this code is execute using the craft file for the example vessel pictured above, the fol-

lowing is printed out:

```
Command Pod Mk1 - axial
TR-18A Stack Decoupler - axial
 FL-T400 Fuel Tank - axial
  LV-909 Liquid Fuel Engine - axial
   TR-18A Stack Decoupler - axial
    FL-T800 Fuel Tank - axial
     LV-909 Liquid Fuel Engine - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
  LT-1 Landing Struts - radial
  LT-1 Landing Struts - radial
Mk16 Parachute - axial
```

Fuel Lines

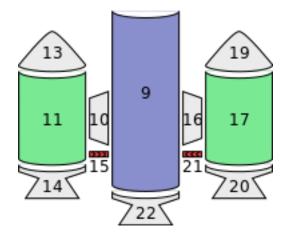
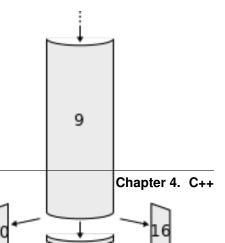


Fig. 4.12: **Figure 5** – Fuel lines from the example in Figure 1. Fuel flows from the parts highlighted in green, into the part highlighted in blue.

Fuel lines are considered parts, and are included in the parts tree (for example, as pictured in Figure 4). However, the parts tree does not contain information about which parts fuel lines connect to. The parent part of a fuel line is the part from which it will take fuel (as shown in Figure 4) however the part that it will send fuel to is not represented in the parts tree.

Figure 5 shows the fuel lines from the example vessel pictured earlier. Fuel line part 15 (in red) takes fuel from a fuel tank (part 11 – in green) and feeds it into another fuel tank (part 9 – in blue).



The fuel line is therefore a child of part 11, but its connection to part 9 is not represented in the tree.

The attributes <code>SpaceCenter::Part::fuel_lines_from()</code> and <code>SpaceCenter::Part::fuel_lines_to()</code> can be used to discover these connections. In the example in Figure 5, when <code>SpaceCenter::Part::fuel_lines_to()</code> is called on fuel tank part 11, it will return a list of parts containing just fuel tank part 9 (the blue part). When <code>SpaceCenter::Part::fuel_lines_from()</code> is called on fuel tank part 9, it will return a list containing fuel tank parts 11 and 17 (the parts colored green).

Staging

Each part has two staging numbers associated with it: the stage in which the part is activated and the stage in which the part is decoupled. These values can be obtained SpaceCenter::Part::stage() using and SpaceCenter::Part::decouple_stage() respectively. For parts that are not activated by staging, SpaceCenter::Part::stage() turns -1. For parts that are never decoupled, SpaceCenter::Part::decouple_stage() a value of -1.

Figure 6 shows an example staging sequence for a vessel. Figure 7 shows the stages in which each part of the vessel will be *activated*. Figure 8 shows the stages in which each part of the vessel will be *decoupled*.

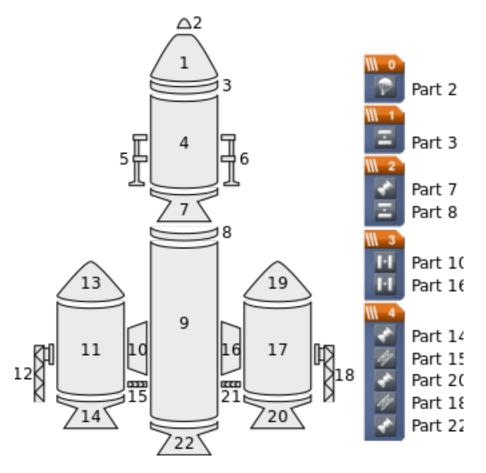


Fig. 4.14: **Figure 6** – Example vessel from Figure 1 with a staging sequence **135**

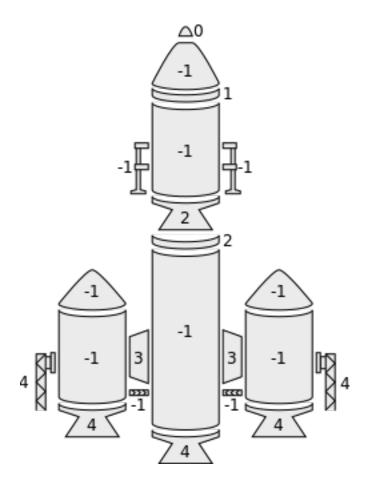


Fig. 4.15: **Figure 7** – The stage in which each part is *activated*.

4.2.8 Resources

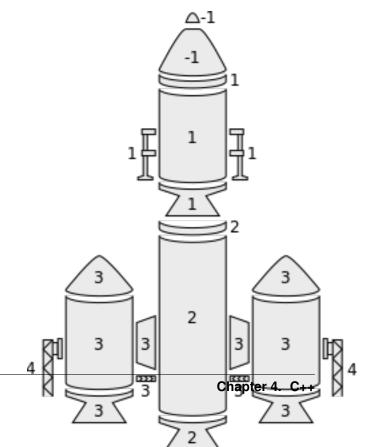
Created by calling Vessel::resources(), Vessel::resources_in_decouple_stage() or Part::resources(). std::vector<std::string> names() A list of resource names that can be stored. bool has_resource (std::string name) Check whether the named resource can be stored. Parameters • name – The name of the resource.

float max (std::string name)

Returns the amount of a resource that can be stored.

Parameters

• name – The name of the resource.



float amount (std::string name)

Returns the amount of a resource that is currently stored.

Parameters

• name – The name of the resource.

static float density (std::string name)

Returns the density of a resource, in kg/l.

Parameters

• name – The name of the resource.

static ResourceFlowMode flow_mode (std::string name)

Returns the flow mode of a resource.

Parameters

• name – The name of the resource.

enum struct ResourceFlowMode

See Resources::flow_mode().

enumerator vessel

The resource flows to any part in the vessel. For example, electric charge.

enumerator stage

The resource flows from parts in the first stage, followed by the second, and so on. For example, mono-propellant.

enumerator adjacent

The resource flows between adjacent parts within the vessel. For example, liquid fuel or oxidizer.

enumerator none

The resource does not flow. For example, solid fuel.

4.2.9 Node

class Node

Represents a maneuver node. Can be created using Control::add_node().

```
float prograde()
```

void set_prograde (float value)

The magnitude of the maneuver nodes delta-v in the prograde direction, in meters per second.

float normal()

void set_normal (float value)

The magnitude of the maneuver nodes delta-v in the normal direction, in meters per second.

```
float radial()
void set_radial (float value)
     The magnitude of the maneuver nodes delta-v in the
     radial direction, in meters per second.
float delta v()
void set_delta_v (float value)
     The delta-v of the maneuver node, in meters per
     second.
     Note: Does not change when executing the maneu-
     ver node. See Node::remaining_delta_v().
float remaining_delta_v()
     Gets the remaining delta-v of the maneuver node, in
     meters per second. Changes as the node is executed.
     This is equivalent to the delta-v reported in-game.
```

std::tuple<double, double, double> burn_vector (ReferenceFrame reference_frame = None) Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s.

Parameters

Note: Does not change when executing the maneuver node. See Node::remaining_burn_vector().

std::tuple<double, double, double> remaining_burn_vector (ReferenceFrame reference_frame = None)

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s. The direction and magnitude change as the burn is executed.

Parameters

node.

```
double ut ()
void set_ut (double value)
     The universal time at which the maneuver will occur,
     in seconds.
double time_to()
     The time until the maneuver node will be encoun-
     tered, in seconds.
Orbit orbit ()
```

The orbit that results from executing the maneuver

void remove()

Removes the maneuver node.

ReferenceFrame reference_frame()

Gets the reference frame that is fixed relative to the maneuver node's burn.

- •The origin is at the position of the maneuver node.
- •The y-axis points in the direction of the burn.
- The x-axis and z-axis point in arbitrary but fixed directions.

ReferenceFrame orbital_reference_frame()

Gets the reference frame that is fixed relative to the maneuver node, and orientated with the orbital prograde/normal/radial directions of the original orbit at the maneuver node's position.

- •The origin is at the position of the maneuver node.
- •The x-axis points in the orbital anti-radial direction of the original orbit, at the position of the maneuver node.
- •The y-axis points in the orbital prograde direction of the original orbit, at the position of the maneuver node.
- •The z-axis points in the orbital normal direction of the original orbit, at the position of the maneuver node.

std::tuple<double, double, double> position (ReferenceFrame reference_frame)

Returns the position vector of the maneuver node in the given reference frame.

Parameters

std::tuple<double, double> direction (ReferenceFrame reference_frame)

Returns the unit direction vector of the maneuver nodes burn in the given reference frame.

Parameters

4.2.10 Comms

class Comms

Used to interact with RemoteTech. Created using a call to Vessel::comms().

Note: This class requires RemoteTech to be installed.

bool has local control()

Whether the vessel can be controlled locally.

bool has_flight_computer()

Whether the vessel has a RemoteTech flight computer on board.

bool has_connection()

Whether the vessel can receive commands from the KSC or a command station.

bool has_connection_to_ground_station()

Whether the vessel can transmit science data to a ground station.

double signal_delay()

The signal delay when sending commands to the vessel, in seconds.

double signal_delay_to_ground_station()

The signal delay between the vessel and the closest ground station, in seconds.

double signal_delay_to_vessel (Vessel other)

Returns the signal delay between the current vessel and another vessel, in seconds.

Parameters

4.2.11 ReferenceFrame

class ReferenceFrame

Represents a reference frame for positions, rotations and velocities. Contains:

- •The position of the origin.
- •The directions of the x, y and z axes.
- •The linear velocity of the frame.
- •The angular velocity of the frame.

Note: This class does not contain any properties or methods. It is only used as a parameter to other functions.

4.2.12 AutoPilot

class AutoPilot

Provides basic auto-piloting utilities for a vessel. Created by calling <code>Vessel::auto_pilot()</code>.

void engage ()

Engage the auto-pilot.

void disengage()

Disengage the auto-pilot.

```
void wait()
     Blocks until the vessel is pointing in the target di-
     rection (if set) and has the target roll (if set).
float error ()
     The error, in degrees, between the direction the
     ship has been asked to point in and the direction
     it is pointing in. Returns zero if the auto-pilot has
     not been engaged, SAS is not enabled, SAS is in
     stability assist mode, or no target direction is set.
float roll_error()
     The error, in degrees, between the roll the ship has
     been asked to be in and the actual roll. Returns zero
     if the auto-pilot has not been engaged or no target
     roll is set.
ReferenceFrame reference_frame()
void set_reference_frame (ReferenceFrame value)
     The reference frame for the target direction
     (AutoPilot::target_direction()).
std::tuple<double, double> target_direction()
void set_target_direction (std::tuple<double, double, double> value)
     The target direction. NULL if no target direction is
void target_pitch_and_heading (float pitch, float heading)
           (AutoPilot::target direction())
     from a pitch and heading angle.
 Parameters
   • pitch – Target pitch angle, in degrees between -90°
   • heading – Target heading angle, in degrees between
     0^{\circ} and 360^{\circ}.
float target_roll()
void set target roll (float value)
     The target roll, in degrees. NaN if no target roll is
bool sas()
void set sas (bool value)
     The state of SAS.
     Note: Equivalent to Control::sas()
```

SASMode sas mode()

```
void set sas mode (SASMode value)
     The current SASMode. These modes are equivalent
     to the mode buttons to the left of the navball that
     appear when SAS is enabled.
     Note: Equivalent to Control::sas_mode()
float rotation_speed_multiplier()
void set_rotation_speed_multiplier (float value)
     Target rotation speed multiplier. Defaults to 1.
float max_rotation_speed()
void set_max_rotation_speed (float value)
     Maximum target rotation speed. Defaults to 1.
float roll_speed_multiplier()
void set_roll_speed_multiplier (float value)
     Target roll speed multiplier. Defaults to 1.
float max roll speed()
void set_max_roll_speed (float value)
     Maximum target roll speed. Defaults to 1.
void set_pid_parameters (float kp = 1.0, float ki = 0.0, float kd = 0.0)
     Sets the gains for the rotation rate PID controller.
 Parameters
   • kp – Proportional gain.
   • ki – Integral gain.
   • kd – Derivative gain.
     4.2.13 Geometry Types
class Vector3
     3-dimensional vectors are represented as a 3-tuple.
     For example:
           #include <krpc.hpp>
```

```
#include <krpc.hpp>
#include <krpc/services/space_center.hpp>
#include <iostream>

using namespace krpc;
using namespace krpc::services;

int main() {
   Client conn = krpc::connect();
   SpaceCenter sc(&conn);
   auto v = sc.active_vessel().flight().prograde();
```

142 Chapter 4. C++

```
std::cout << v.get<0>() << " " << v.get<1>() << " " << v.get<2>() << std::endl;
}</pre>
```

class Quaternion

Quaternions (rotations in 3-dimensional space) are encoded as a 4-tuple containing the x, y, z and w components. For example:

```
#include <krpc.hpp>
#include <krpc/services/space_center.hpp>
#include <iostream>

using namespace krpc;
using namespace krpc::services;

int main() {
   Client conn = krpc::connect();
   SpaceCenter sc(&conn);
   auto q = sc.active_vessel().flight().rotation();
   std::cout << q.get<0>() << " " << q.get<1>() << " " << q.get<2>() << " " << q.get<3>() << std:
}</pre>
```

4.3 InfernalRobotics API

Provides RPCs to interact with the InfernalRobotics mod. Provides the following classes:

4.3.1 InfernalRobotics

class InfernalRobotics : public krpc::Service

This service provides functionality to interact with the Infernal Robotics mod.

InfernalRobotics (krpc::Client *client)

Construct an instance of this service.

```
std::vector<ControlGroup> servo_groups ()
```

A list of all the servo groups in the active vessel.

ControlGroup servo_group_with_name (std::string name)

Returns the servo group with the given *name* or NULL if none exists. If multiple servo groups have the same name, only one of them is returned.

Parameters

• name – Name of servo group to find.

Servo servo_with_name (std::string name)

Returns the servo with the given *name*, from all servo groups, or NULL if none exists. If multiple servos have the same name, only one of them is returned.

Parameters

• name – Name of the servo to find.

4.3.2 ControlGroup

```
class ControlGroup
                                      obtained
     Α
           group
                           servos,
                                                   by
     calling
                      servo_groups()
                                                   or
                                           Represents
     servo_group_with_name().
     the "Servo Groups" in the InfernalRobotics UI.
std::string name ()
void set_name (std::string value)
     The name of the group.
std::string forward_key()
void set_forward_key (std::string value)
     The key assigned to be the "forward" key for the
     group.
std::string reverse_key()
void set_reverse_key (std::string value)
     The key assigned to be the "reverse" key for the
     group.
float speed()
void set_speed (float value)
     The speed multiplier for the group.
bool expanded()
void set_expanded (bool value)
     Whether the group is expanded in the Infernal-
     Robotics UI.
std::vector<Servo> servos()
     The servos that are in the group.
Servo servo_with_name (std::string name)
     Returns the servo with the given name from this
     group, or NULL if none exists.
 Parameters
   • name – Name of servo to find.
void move_right()
     Moves all of the servos in the group to the right.
void move left()
     Moves all of the servos in the group to the left.
void move_center()
     Moves all of the servos in the group to the center.
```

144 Chapter 4. C++

void move_next_preset()

Moves all of the servos in the group to the next preset.

void move_prev_preset ()

Moves all of the servos in the group to the previous preset.

void stop()

Stops the servos in the group.

4.3.3 Servo

class Servo

std::string name()

void set_name (std::string value)

The name of the servo.

void set_highlight (bool value)

Whether the servo should be highlighted in-game.

float position()

The position of the servo.

float min_config_position()

The minimum position of the servo, specified by the part configuration.

float max_config_position()

The maximum position of the servo, specified by the part configuration.

float min position ()

void set_min_position (float value)

The minimum position of the servo, specified by the in-game tweak menu.

float max_position()

void set_max_position (float value)

The maximum position of the servo, specified by the in-game tweak menu.

float config_speed()

The speed multiplier of the servo, specified by the part configuration.

float speed()

```
void set speed (float value)
     The speed multiplier of the servo, specified by the
     in-game tweak menu.
float current_speed()
void set_current_speed (float value)
     The current speed at which the servo is moving.
float acceleration()
void set_acceleration (float value)
     The current speed multiplier set in the UI.
bool is_moving()
     Whether the servo is moving.
bool is_free_moving()
     Whether the servo is freely moving.
bool is_locked()
void set_is_locked (bool value)
     Whether the servo is locked.
bool is_axis_inverted()
void set_is_axis_inverted(bool value)
     Whether the servos axis is inverted.
void move right()
     Moves the servo to the right.
void move_left()
    Moves the servo to the left.
void move_center()
    Moves the servo to the center.
void move next preset()
    Moves the servo to the next preset.
void move_prev_preset ()
    Moves the servo to the previous preset.
void move_to (float position, float speed)
     Moves the servo to position and sets the speed mul-
    tiplier to speed.
 Parameters
   • position – The position to move the servo to.
   • speed – Speed multiplier for the movement.
void stop()
    Stops the servo.
```

146 Chapter 4. C++

4.3.4 Example

The following example gets the control group named "MyGroup", prints out the names and positions of all of the servos in the group, then moves all of the servos to the right for 1 second.

```
#include <krpc.hpp>
#include <krpc/services/infernal_robotics.hpp>
#include <iostream>
#include <vector>
using namespace krpc;
using namespace krpc::services;
int main() {
  Client conn = krpc::connect("InfernalRobotics Example");
  InfernalRobotics infernal_robotics(&conn);
  InfernalRobotics::ControlGroup group = infernal_robotics.servo_group_with_name("MyGroup");
  if (group == InfernalRobotics::ControlGroup()) {
    std::cout << "Group not found" << std::endl;</pre>
    return 1;
  std::vector<InfernalRobotics::Servo> servos = group.servos();
  for (std::vector<InfernalRobotics::Servo>::iterator i = servos.begin();
       i != servos.end(); i++) {
    std::cout << i->name() << " " << i->position() << std::endl;
  group.move_right();
  sleep(1); //Note: platform dependent
  group.stop();
```

4.4 Kerbal Alarm Clock API

Provides RPCs to interact with the Kerbal Alarm Clock mod. Provides the following classes:

4.4.1 KerbalAlarmClock

```
class KerbalAlarmClock : public krpc::Service
    This service provides functionality to interact with
    the Kerbal Alarm Clock mod.

KerbalAlarmClock (krpc::Client *client)
    Construct an instance of this service.

std::vector<Alarm> alarms()
    A list of all the alarms.

Alarm alarm_with_name(std::string name)
    Get the alarm with the given name, or NULL if no
```

alarms have that name. If more than one alarm has the name, only returns one of them.

Parameters

• name – Name of the alarm to search for.

```
std::vector<Alarm> alarms_with_type (AlarmType type)
Get a list of alarms of the specified type.
```

Parameters

• **type** – Type of alarm to return.

```
Alarm create_alarm (AlarmType type, std::string name, double ut)

Create a new alarm and return it.
```

Parameters

- **type** Type of the new alarm.
- name Name of the new alarm.
- ut Time at which the new alarm should trigger.

4.4.2 Alarm

```
class Alarm
     Represents an alarm.
                                 Obtained by call-
     ing alarms(), alarm_with_name() or
     alarms_with_type().
AlarmAction action()
void set_action (AlarmAction value)
     The action that the alarm triggers.
double margin ()
void set_margin (double value)
     The number of seconds before the event that the
     alarm will fire.
double time()
void set time (double value)
     The time at which the alarm will fire.
AlarmType type()
     The type of the alarm.
std::string id()
     The unique identifier for the alarm.
std::string name()
void set_name (std::string value)
```

The short name of the alarm.

148 Chapter 4. C++

```
std::string notes()
void set_notes (std::string value)
     The long description of the alarm.
double remaining()
     The number of seconds until the alarm will fire.
bool repeat ()
void set_repeat (bool value)
     Whether the alarm will be repeated after it has fired.
double repeat_period()
void set_repeat_period (double value)
     The time delay to automatically create an alarm
     after it has fired.
SpaceCenter::Vessel vessel ()
void set_vessel (SpaceCenter::Vessel value)
     The vessel that the alarm is attached to.
SpaceCenter::CelestialBody xfer_origin_body()
void set_xfer_origin_body (SpaceCenter::CelestialBody value)
     The celestial body the vessel is departing from.
SpaceCenter::CelestialBody xfer_target_body()
void set_xfer_target_body (SpaceCenter::CelestialBody value)
     The celestial body the vessel is arriving at.
void remove()
     Removes the alarm.
     4.4.3 AlarmType
enum struct AlarmType
     The type of an alarm.
enumerator raw
     An alarm for a specific date/time or a specific period
     in the future.
enumerator maneuver
     An alarm based on the next maneuver node on the
```

4.4. Kerbal Alarm Clock API

See AlarmType::maneuver.

enumerator maneuver_auto

current ships flight path. This node will be stored and can be restored when you come back to the ship.

enumerator apoapsis

An alarm for furthest part of the orbit from the planet.

enumerator periapsis

An alarm for nearest part of the orbit from the planet.

enumerator ascending_node

Ascending node for the targeted object, or equatorial ascending node.

enumerator descending_node

Descending node for the targeted object, or equatorial descending node.

enumerator closest

An alarm based on the closest approach of this vessel to the targeted vessel, some number of orbits into the future.

enumerator contract

An alarm based on the expiry or deadline of contracts in career modes.

enumerator contract auto

See AlarmType::contract.

enumerator crew

An alarm that is attached to a crew member.

enumerator distance

An alarm that is triggered when a selected target comes within a chosen distance.

enumerator earth time

An alarm based on the time in the "Earth" alternative Universe (aka the Real World).

enumerator launch_rendevous

An alarm that fires as your landed craft passes under the orbit of your target.

enumerator soi_change

An alarm manually based on when the next SOI point is on the flight path or set to continually monitor the active flight path and add alarms as it detects SOI changes.

enumerator soi_change_auto

See AlarmType::soi_change.

enumerator transfer

An alarm based on Interplanetary Transfer Phase Angles, i.e. when should I launch to planet X? Based on Kosmo Not's post and used in Olex's Calculator.

enumerator transfer_modelled

See AlarmType::transfer.

150 Chapter 4. C++

4.4.4 AlarmAction

enum struct AlarmAction

The action performed by an alarm when it fires.

enumerator do_nothing

Don't do anything at all...

enumerator do_nothing_delete_when_passed

Don't do anything, and delete the alarm.

enumerator kill_warp

Drop out of time warp.

enumerator kill_warp_only

Drop out of time warp.

enumerator message_only

Display a message.

enumerator pause_game

Pause the game.

4.4.5 Example

The following example creates a new alarm for the active vessel. The alarm is set to trigger after 10 seconds have passed, and display a message.

152 Chapter 4. C++

CHAPTER

FIVE

JAVA

5.1 Java Client

The krpc.client package provides functionality to interact with a kRPC server from Java.

5.1.1 Connecting to the Server

To connect to a server, use the *Connection.newInstance()* function. This returns a connection object through which you can interact with the server. For example to connect to a server running on the local machine:

```
Connection conn = Connection.newInstance("Example");
System.out.println(KRPC.newInstance(conn).getStatus().getVersion());
```

This function also accepts arguments that specify what address and port numbers to connect to. For example:

```
Connection conn = Connection.newInstance("Remote example", "my.domain.name", 1000, 1001);
System.out.println(KRPC.newInstance(conn).getStatus().getVersion());
```

5.1.2 Interacting with the Server

Interaction with the server is performed via a connection object. Functionality for services are defined in the package krpc.client.services.*.

Before a service can be used it must first be instantiated. The following example connects to the server, instantiates the SpaceCenter service, and outputs the name of the active vessel:

5.1.3 Streaming Data from the Server

A stream repeatedly executes a function on the server, with a fixed set of argument values. It provides a more efficient way of repeatedly getting the result of calling function on the server, without having to invoke it directly – which incurs communication overheads.

For example, consider the following loop that continuously prints out the position of the active vessel. This loop incurs significant communication overheads, as the vessel.position() function is called repeatedly.

```
Vessel vessel = spaceCenter.getActiveVessel();
ReferenceFrame refframe = vessel.getOrbit().getBody().getReferenceFrame();
while (true)
    System.out.println(vessel.position(refframe));
```

The following code achieves the same thing, but is far more efficient. It makes a single call to Connection.addStream to create the stream, which avoids the communication overhead in the previous example.

Streams are created by calling <code>Connection.addStream</code> and passing it information about which method to stream. The example above passes a remote object, the name of the method to call, followed by the arguments to pass to the method (if any). The most recent value for the stream can be obtained by calling <code>Stream.get</code>.

Streams can also be added for static methods as follows:

```
Stream<Double> time_stream = connection.addStream(SpaceCenter.class, "getUt");
```

A stream can be removed by calling Stream.remove().

5.1.4 Reference

class Connection

This class provides the interface for communicating with the server.

```
static Connection newInstance (String name)
static Connection newInstance (String name, String address)
static Connection newInstance (String name, String address, int rpcPort, int streamPort)
static Connection newInstance (String name, InetAddress address)
static Connection newInstance (String name, InetAddress address, int rpcPort, int stream-
Port)
Create a connection to the server, using the given connection details.
```

Parameters

- name (String) A descriptive name for the connection. This is passed to the server and appears, for example, in the client connection dialog on the in-game server window.
- address (String) The address of the server to connect to. Can either be a hostname, an IP address as a string or an InetAddress object. Defaults to '127.0.0.1'.

```
    rpc_port (int) - The port number of the RPC Server. Defaults to 50000.
    stream_port (int) - The port number of the Stream Server. Defaults to 50001.
    void close ()
        Close the connection.
    Stream<T> addStream (Class<?> clazz, String method, Object... args)
        Create a stream for a static method call to the given class.
    Stream<T> addStream (RemoteObject instance, String method, Object... args)
        Create a stream for a method call to the given remote object.
    class Stream<T>
        A stream object.
    T get ()
        Get the most recent value for the stream.
    void remove ()
        Remove the stream from the server.
    abstract class RemoteObject
```

5.2 SpaceCenter API

The abstract base class for all remote objects.

5.2.1 SpaceCenter

public class SpaceCenter

Provides functionality to interact with Kerbal Space Program. This includes controlling the active vessel, managing its resources, planning maneuver nodes and auto-piloting.

```
Vessel getActiveVessel()
void setActiveVessel (Vessel value)
    The currently active vessel.
java.util.List<Vessel> getVessels()
    A list of all the vessels in the game.
java.util.Map<String, CelestialBody> getBodies ()
    A dictionary of all celestial bodies (planets, moons, etc.) in the game, keyed by the name of the body.
CelestialBody getTargetBody()
void setTargetBody (CelestialBody value)
    The currently targeted celestial body.
Vessel getTargetVessel()
void setTargetVessel (Vessel value)
    The currently targeted vessel.
DockingPort getTargetDockingPort()
void setTargetDockingPort (DockingPort value)
    The currently targeted docking port.
void ClearTarget ()
```

Clears the current target.

void LaunchVesselFromVAB (String name)

Launch a new vessel from the VAB onto the launchpad.

Parameters

• name (String) - Name of the vessel's craft file.

void LaunchVesselFromSPH (String name)

Launch a new vessel from the SPH onto the runway.

Parameters

• name (String) – Name of the vessel's craft file.

double getUT()

The current universal time in seconds.

float getG()

The value of the gravitational constant G in $N(m/kg)^2$.

WarpMode getWarpMode()

The current time warp mode. Returns <code>WarpMode.None</code> if time warp is not active, <code>WarpMode.Rails</code> if regular "on-rails" time warp is active, or <code>WarpMode.Physics</code> if physical time warp is active.

float getWarpRate()

The current warp rate. This is the rate at which time is passing for either on-rails or physical time warp. For example, a value of 10 means time is passing 10x faster than normal. Returns 1 if time warp is not active.

float getWarpFactor()

The current warp factor. This is the index of the rate at which time is passing for either regular "on-rails" or physical time warp. Returns 0 if time warp is not active. When in on-rails time warp, this is equal to RailsWarpFactor, and in physics time warp, this is equal to PhysicsWarpFactor.

int getRailsWarpFactor()

void setRailsWarpFactor (int value)

The time warp rate, using regular "on-rails" time warp. A value between 0 and 7 inclusive. 0 means no time warp. Returns 0 if physical time warp is active. If requested time warp factor cannot be set, it will be set to the next lowest possible value. For example, if the vessel is too close to a planet. See the KSP wiki for details.

int getPhysicsWarpFactor()

void setPhysicsWarpFactor (int value)

The physical time warp rate. A value between 0 and 3 inclusive. 0 means no time warp. Returns 0 if regular "on-rails" time warp is active.

boolean CanRailsWarpAt (int factor)

Returns true if regular "on-rails" time warp can be used, at the specified warp *factor*. The maximum time warp rate is limited by various things, including how close the active vessel is to a planet. See the KSP wiki for details.

Parameters

• **factor** (*int*) – The warp factor to check.

int getMaximumRailsWarpFactor()

The current maximum regular "on-rails" warp factor that can be set. A value between 0 and 7 inclusive. See the KSP wiki for details.

void WarpTo (double UT, float maxRailsRate, float maxPhysicsRate)

Uses time acceleration to warp forward to a time in the future, specified by universal time UT. This call

blocks until the desired time is reached. Uses regular "on-rails" or physical time warp as appropriate. For example, physical time warp is used when the active vessel is traveling through an atmosphere. When using regular "on-rails" time warp, the warp rate is limited by *maxRailsRate*, and when using physical time warp, the warp rate is limited by *maxPhysicsRate*.

Parameters

- **UT** (double) The universal time to warp to, in seconds.
- maxRailsRate (float) The maximum warp rate in regular "on-rails" time warp.
- maxPhysicsRate (float) The maximum warp rate in physical time warp.

Returns When the time warp is complete.

org.javatuples.Triplet<Double, Double, Double> **TransformPosition** (org.javatuples.Triplet<Double, Double, Double, Double> position, ReferenceFrame from, ReferenceFrame to)

Converts a position vector from one reference frame to another.

Parameters

- position (org. javatuples. Triplet < Double, Double, Double>) Position vector in reference frame from.
- **from** (ReferenceFrame) The reference frame that the position vector is in.
- to (ReferenceFrame) The reference frame to covert the position vector to.

Returns The corresponding position vector in reference frame *to*.

org.javatuples.Triplet<Double, Double> **TransformDirection** (org.javatuples.Triplet<Double, Double, Double, Double> direction, ReferenceFrame from, ReferenceFrame to)

Converts a direction vector from one reference frame to another.

Parameters

- direction (org.javatuples.Triplet<Double, Double, Double>) Direction vector in reference frame from.
- from (ReferenceFrame) The reference frame that the direction vector is in.
- to (ReferenceFrame) The reference frame to covert the direction vector to.

Returns The corresponding direction vector in reference frame to.

org.javatuples.Quartet<Double, Double, Double, Double>TransformRotation(org.javatuples.Quartet<Double,

Double, Double, Double> rotation, ReferenceFrame from, Reference-Frame to)

Converts a rotation from one reference frame to another.

Parameters

- rotation(org.javatuples.Quartet<Double, Double, Double, Double>)
 Rotation in reference frame from.
- **from** (ReferenceFrame) The reference frame that the rotation is in.
- to (ReferenceFrame) The corresponding rotation in reference frame to.

Returns The corresponding rotation in reference frame to.

org.javatuples.Triplet<Double, Double, Double> TransformVelocity (org.javatuples.Triplet<Double, Double> position, org.javatuples.Triplet<Double, Double, Double> velocity, ReferenceFrame from, ReferenceFrame to)

Converts a velocity vector (acting at the specified position vector) from one reference frame to another. The position vector is required to take the relative angular velocity of the reference frames into account.

Parameters

- **position** (org.javatuples.Triplet<Double, Double, Double>) Position vector in reference frame from.
- **velocity** (org. javatuples. Triplet < Double, Double, Double>) **Veloc**ity vector in reference frame *from*.
- **from** (ReferenceFrame) The reference frame that the position and velocity vectors are in.
- to (ReferenceFrame) The reference frame to covert the velocity vector to.

Returns The corresponding velocity in reference frame *to*.

boolean getFARAvailable()

Whether Ferram Aerospace Research is installed.

boolean getRemoteTechAvailable()

Whether RemoteTech is installed.

void **DrawDirection** (org.javatuples.Triplet<Double, Double, Double> direction, ReferenceFrame referenceFrame, org.javatuples.Triplet<Double, Double, Double> color, float length)

Draw a direction vector on the active vessel.

Parameters

- direction (org. javatuples. Triplet < Double, Double, Double>) Direction to draw the line in.
- referenceFrame (ReferenceFrame) Reference frame that the direction is in.
- **color** (org.javatuples.Triplet<Double, Double, Double>) The color to use for the line, as an RGB color.
- **length** (float) The length of the line. Defaults to 10.

void **DrawLine** (org.javatuples.Triplet<Double, Double, Double start, org.javatuples.Triplet<Double, Double, Double end, ReferenceFrame referenceFrame, org.javatuples.Triplet<Double, Double, Double color)

Draw a line.

Parameters

- start (org. javatuples. Triplet < Double, Double, Double>) Position of the start of the line.
- end (org. javatuples. Triplet < Double, Double, Double>) Position of the end of the line.
- referenceFrame (ReferenceFrame) Reference frame that the position are in.

• color (org. javatuples. Triplet < Double, Double, Double>) - The color to use for the line, as an RGB color.

void ClearDrawing()

Remove all directions and lines currently being drawn.

public enum WarpMode

```
Returned by WarpMode
```

```
public WarpMode Rails
```

Time warp is active, and in regular "on-rails" mode.

public WarpMode Physics

Time warp is active, and in physical time warp mode.

public WarpMode None

Time warp is not active.

5.2.2 Vessel

public class Vessel

These objects are used to interact with vessels in KSP. This includes getting orbital and flight data, manipulating control inputs and managing resources.

```
String getName()
```

void setName (String value)

The name of the vessel.

VesselType getType()

void setType (VesselType value)

The type of the vessel.

VesselSituation getSituation()

The situation the vessel is in.

double getMET()

The mission elapsed time in seconds.

Flight Flight (ReferenceFrame referenceFrame)

Returns a Flight object that can be used to get flight telemetry for the vessel, in the specified reference frame.

Parameters

• referenceFrame (ReferenceFrame) - Reference frame. Defaults to the vessel's surface reference frame (Vessel.SurfaceReferenceFrame).

Vessel getTarget()

void setTarget (Vessel value)

The target vessel. null if there is no target. When setting the target, the target cannot be the current vessel.

Orbit getOrbit()

The current orbit of the vessel.

Control getControl()

Returns a *Control* object that can be used to manipulate the vessel's control inputs. For example, its pitch/yaw/roll controls, RCS and thrust.

AutoPilot getAutoPilot()

An AutoPilot object, that can be used to perform simple auto-piloting of the vessel.

Resources getResources()

A Resources object, that can used to get information about resources stored in the vessel.

Resources InDecoupleStage (int stage, boolean cumulative)

Returns a Resources object, that can used to get information about resources stored in a given stage.

Parameters

- **stage** (*int*) Get resources for parts that are decoupled in this stage.
- **cumulative** (boolean) When false, returns the resources for parts decoupled in just the given stage. When true returns the resources decoupled in the given stage and all subsequent stages combined.

Note: For details on stage numbering, see the discussion on <code>java-api-parts-staging</code>.

Parts getParts()

A Parts object, that can used to interact with the parts that make up this vessel.

Comms getComms ()

A Comms object, that can used to interact with RemoteTech for this vessel.

Note: Requires RemoteTech to be installed.

float getMass()

The total mass of the vessel, including resources, in kg.

float getDryMass()

The total mass of the vessel, excluding resources, in kg.

float getThrust()

The total thrust currently being produced by the vessel's engines, in Newtons. This is computed by summing Engine. Thrust for every engine in the vessel.

float getAvailableThrust()

Gets the total available thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing Engine.AvailableThrust for every active engine in the vessel.

float getMaxThrust()

The total maximum thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing Engine. MaxThrust for every active engine.

float getMaxVacuumThrust()

The total maximum thrust that can be produced by the vessel's active engines when the vessel is in a vacuum, in Newtons. This is computed by summing Engine. MaxVacuumThrust for every active engine.

float getSpecificImpulse()

The combined specific impulse of all active engines, in seconds. This is computed using the formula described here.

float getVacuumSpecificImpulse()

The combined vacuum specific impulse of all active engines, in seconds. This is computed using the formula described here.

float getKerbinSeaLevelSpecificImpulse()

The combined specific impulse of all active engines at sea level on Kerbin, in seconds. This is computed using the formula described here.

ReferenceFrame getReferenceFrame()

The reference frame that is fixed relative to the vessel, and orientated with the vessel.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel.
- •The x-axis points out to the right of the vessel.
- •The y-axis points in the forward direction of the vessel.
- •The z-axis points out of the bottom off the vessel.

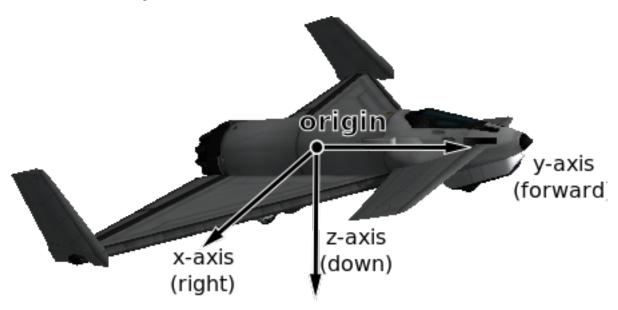


Fig. 5.1: Vessel reference frame origin and axes for the Aeris 3A aircraft

ReferenceFrame getOrbitalReferenceFrame()

The reference frame that is fixed relative to the vessel, and orientated with the vessels orbital prograde/normal/radial directions.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Note: Be careful not to confuse this with 'orbit' mode on the navball.

ReferenceFrame getSurfaceReferenceFrame()

The reference frame that is fixed relative to the vessel, and orientated with the surface of the body being orbited.

•The origin is at the center of mass of the vessel.

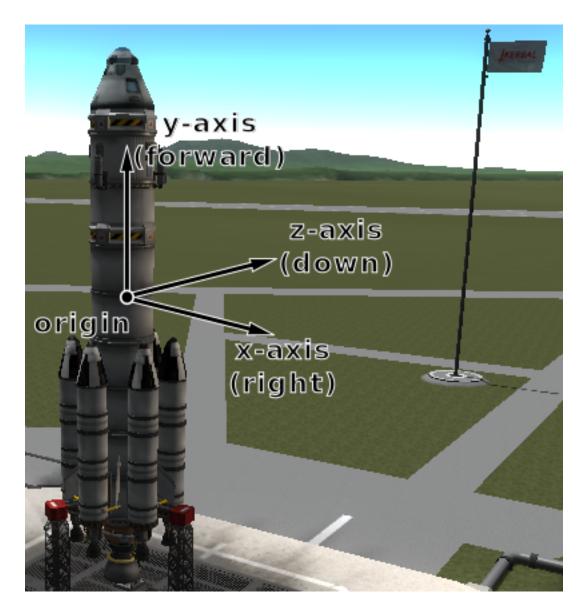


Fig. 5.2: Vessel reference frame origin and axes for the Kerbal-X rocket

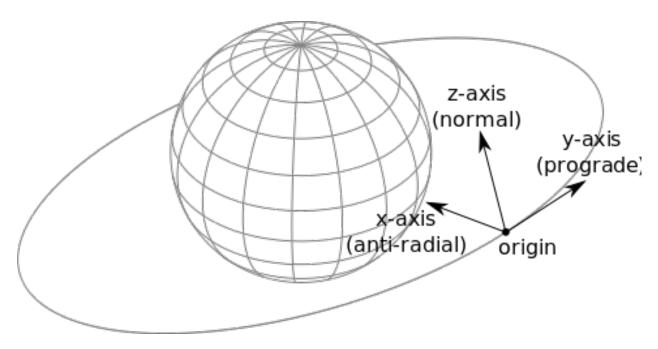


Fig. 5.3: Vessel orbital reference frame origin and axes

- •The axes rotate with the north and up directions on the surface of the body.
- •The x-axis points in the zenith direction (upwards, normal to the body being orbited, from the center of the body towards the center of mass of the vessel).
- •The y-axis points northwards towards the astronomical horizon (north, and tangential to the surface of the body the direction in which a compass would point when on the surface).
- •The z-axis points eastwards towards the astronomical horizon (east, and tangential to the surface of the body east on a compass when on the surface).

Note: Be careful not to confuse this with 'surface' mode on the navball.

ReferenceFrame getSurfaceVelocityReferenceFrame()

The reference frame that is fixed relative to the vessel, and orientated with the velocity vector of the vessel relative to the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel's velocity vector.
- •The y-axis points in the direction of the vessel's velocity vector, relative to the surface of the body being orbited.
- •The z-axis is in the plane of the astronomical horizon.
- •The x-axis is orthogonal to the other two axes.

org.javatuples.Triplet<Double, Double, Double> **Position** (*ReferenceFrame referenceFrame*)

Returns the position vector of the center of mass of the vessel in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

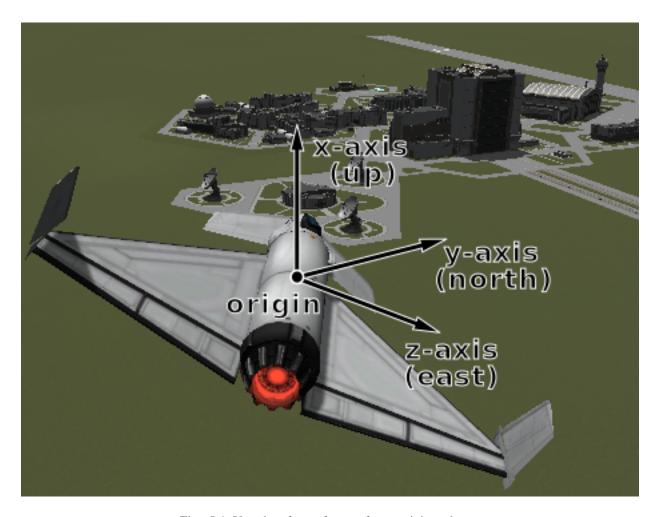


Fig. 5.4: Vessel surface reference frame origin and axes

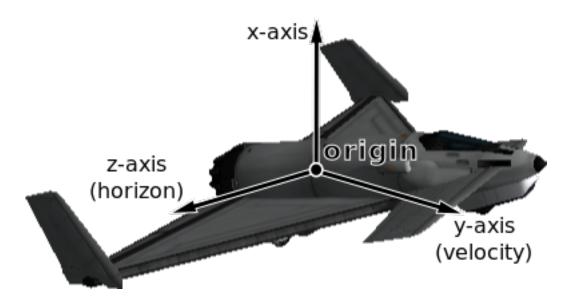


Fig. 5.5: Vessel surface velocity reference frame origin and axes

org.javatuples.Triplet<Double, Double, Double> **Velocity** (*ReferenceFrame referenceFrame*)
Returns the velocity vector of the center of mass of the vessel in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Quartet<Double, Double, Double, Double>Rotation (ReferenceFrame reference-

Frame)

Returns the rotation of the center of mass of the vessel in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> **Direction** (*ReferenceFrame referenceFrame*)

Returns the direction in which the vessel is pointing, as a unit vector, in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> AngularVelocity (ReferenceFrame reference-Frame)

Returns the angular velocity of the vessel in the given reference frame. The magnitude of the returned vector is the rotational speed in radians per second, and the direction of the vector indicates the axis of rotation (using the right hand rule).

Parameters

• referenceFrame (ReferenceFrame) -

```
public enum VesselType
     See Vessel. Type.
     public VesselType Ship
          Ship.
     public VesselType Station
          Station.
     public VesselType Lander
          Lander.
     public VesselType Probe
          Probe.
     public VesselType Rover
          Rover.
     public VesselType Base
          Base.
     public VesselType Debris
          Debris.
public enum VesselSituation
     See Vessel. Situation.
     public VesselSituation Docked
          Vessel is docked to another.
     public VesselSituation Escaping
```

Escaping.

```
public VesselSituation Flying
           Vessel is flying through an atmosphere.
     public VesselSituation Landed
           Vessel is landed on the surface of a body.
     public VesselSituation Orbiting
           Vessel is orbiting a body.
     public VesselSituation PreLaunch
           Vessel is awaiting launch.
     public VesselSituation Splashed
           Vessel has splashed down in an ocean.
     public VesselSituation SubOrbital
           Vessel is on a sub-orbital trajectory.
5.2.3 CelestialBody
public class CelestialBody
     Represents a celestial body (such as a planet or moon).
      String getName()
           The name of the body.
      java.util.List<CelestialBody> getSatellites()
           A list of celestial bodies that are in orbit around this celestial body.
      Orbit getOrbit()
           The orbit of the body.
      float getMass()
           The mass of the body, in kilograms.
      float getGravitationalParameter()
           The standard gravitational parameter of the body in m^3s^{-2}.
      float getSurfaceGravity()
           The acceleration due to gravity at sea level (mean altitude) on the body, in m/s^2.
      float getRotationalPeriod()
           The sidereal rotational period of the body, in seconds.
      float getRotationalSpeed()
           The rotational speed of the body, in radians per second.
      float getEquatorialRadius()
           The equatorial radius of the body, in meters.
      double SurfaceHeight (double latitude, double longitude)
           The height of the surface relative to mean sea level at the given position, in meters. When over water this
           is equal to 0.
```

Parameters

- latitude (double) Latitude in degrees
- longitude (double) Longitude in degrees

double BedrockHeight (double latitude, double longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water, this is the height of the sea-bed and is therefore a negative value.

Parameters

- latitude (double) Latitude in degrees
- longitude (double) Longitude in degrees

org.javatuples.Triplet<Double, Double, Double> MSLPosition (double latitude, double longitude, ReferenceFrame referenceFrame)

The position at mean sea level at the given latitude and longitude, in the given reference frame.

Parameters

- latitude (double) Latitude in degrees
- longitude (double) Longitude in degrees
- referenceFrame (ReferenceFrame) Reference frame for the returned position vector

org.javatuples.Triplet<Double, Double, Double> SurfacePosition (double latitude, double longitude, ReferenceFrame referenceFrame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position of the surface of the water.

Parameters

- latitude (double) Latitude in degrees
- longitude (double) Longitude in degrees
- referenceFrame (ReferenceFrame) Reference frame for the returned position vector

org.javatuples.Triplet<Double, Double, Double> **BedrockPosition** (double latitude, double lon-gitude, ReferenceFrame referenceFrame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position at the bottom of the sea-bed.

Parameters

- latitude (double) Latitude in degrees
- **longitude** (*double*) **Longitude** in degrees
- referenceFrame (ReferenceFrame) Reference frame for the returned position vector

float getSphereOfInfluence()

The radius of the sphere of influence of the body, in meters.

boolean getHasAtmosphere()

true if the body has an atmosphere.

float getAtmosphereDepth()

The depth of the atmosphere, in meters.

boolean getHasAtmosphericOxygen()

true if there is oxygen in the atmosphere, required for air-breathing engines.

ReferenceFrame getReferenceFrame()

The reference frame that is fixed relative to the celestial body.

- •The origin is at the center of the body.
- •The axes rotate with the body.

- •The x-axis points from the center of the body towards the intersection of the prime meridian and equator (the position at 0° longitude, 0° latitude).
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points from the center of the body towards the equator at 90°E longitude.

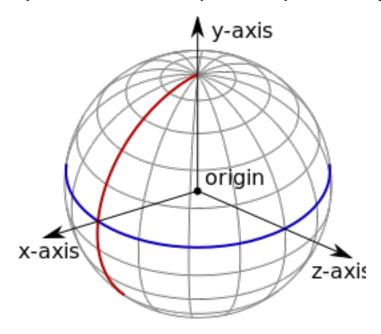


Fig. 5.6: Celestial body reference frame origin and axes. The equator is shown in blue, and the prime meridian in red.

ReferenceFrame getNonRotatingReferenceFrame()

The reference frame that is fixed relative to this celestial body, and orientated in a fixed direction (it does not rotate with the body).

- •The origin is at the center of the body.
- •The axes do not rotate.
- •The x-axis points in an arbitrary direction through the equator.
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points in an arbitrary direction through the equator.

ReferenceFrame getOrbitalReferenceFrame()

Gets the reference frame that is fixed relative to this celestial body, but orientated with the body's orbital prograde/normal/radial directions.

- •The origin is at the center of the body.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

org.javatuples.Triplet<Double, Double, Double> Position (ReferenceFrame referenceFrame)

Returns the position vector of the center of the body in the specified reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> **Velocity** (*ReferenceFrame referenceFrame*)

Returns the velocity vector of the body in the specified reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Quartet<Double, Double, Double, Double>Rotation (ReferenceFrame reference-

Frame)

Returns the rotation of the body in the specified reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> **Direction** (*ReferenceFrame referenceFrame*)

Returns the direction in which the north pole of the celestial body is pointing, as a unit vector, in the specified reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> AngularVelocity (ReferenceFrame reference-

Frame

Returns the angular velocity of the body in the specified reference frame. The magnitude of the vector is the rotational speed of the body, in radians per second, and the direction of the vector indicates the axis of rotation, using the right-hand rule.

Parameters

• referenceFrame (ReferenceFrame) -

5.2.4 Flight

public class Flight

Used to get flight telemetry for a vessel, by calling Vessel.Flight. All of the information returned by this class is given in the reference frame passed to that method.

Note: To get orbital information, such as the apoapsis or inclination, see Orbit.

float getGForce()

The current G force acting on the vessel in m/s^2 .

double getMeanAltitude()

The altitude above sea level, in meters.

double getSurfaceAltitude()

The altitude above the surface of the body or sea level, whichever is closer, in meters.

double getBedrockAltitude()

The altitude above the surface of the body, in meters. When over water, this is the altitude above the sea floor.

double **getElevation**()

The elevation of the terrain under the vessel, in meters. This is the height of the terrain above sea level, and is negative when the vessel is over the sea.

```
double getLatitude()
    The latitude of the vessel for the body being orbited, in degrees.
double getLongitude()
    The longitude of the vessel for the body being orbited, in degrees.
org.javatuples.Triplet<Double, Double, Double> getVelocity()
    The velocity vector of the vessel. The magnitude of the vector is the speed of the vessel in meters per
    second. The direction of the vector is the direction of the vessels motion.
double getSpeed()
    The speed of the vessel in meters per second.
double getHorizontalSpeed()
    The horizontal speed of the vessel in meters per second.
double getVerticalSpeed()
    The vertical speed of the vessel in meters per second.
org.javatuples.Triplet<Double, Double, Double> getCenterOfMass()
    The position of the center of mass of the vessel.
org.javatuples.Quartet<Double, Double, Double, Double> getRotation()
    The rotation of the vessel.
org.javatuples.Triplet<Double, Double, Double> getDirection()
    The direction vector that the vessel is pointing in.
float getPitch()
    The pitch angle of the vessel relative to the horizon, in degrees. A value between -90° and +90°.
float getHeading()
    The heading angle of the vessel relative to north, in degrees. A value between 0° and 360°.
float getRoll()
    The roll angle of the vessel relative to the horizon, in degrees. A value between -180° and +180°.
org.javatuples.Triplet<Double, Double, Double> getPrograde()
    The unit direction vector pointing in the prograde direction.
org.javatuples.Triplet<Double, Double, Double> getRetrograde()
    The unit direction vector pointing in the retrograde direction.
org.javatuples.Triplet<Double, Double, Double> getNormal()
    The unit direction vector pointing in the normal direction.
org.javatuples.Triplet<Double, Double, Double> getAntiNormal()
    The unit direction vector pointing in the anti-normal direction.
org.javatuples.Triplet<Double, Double, Double> getRadial()
    The unit direction vector pointing in the radial direction.
org.javatuples.Triplet<Double, Double, Double> getAntiRadial()
    The unit direction vector pointing in the anti-radial direction.
float getAtmosphereDensity()
    The current density of the atmosphere around the vessel, in kg/m^3.
float getDynamicPressure()
    The dynamic pressure acting on the vessel, in Pascals. This is a measure of the strength of the aerodynamic
    forces. It is equal to \frac{1}{2} air density velocity<sup>2</sup>. It is commonly denoted as Q.
```

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

float getStaticPressure()

The static atmospheric pressure acting on the vessel, in Pascals.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

org.javatuples.Triplet<Double, Double, Double> getAerodynamicForce()

The total aerodynamic forces acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

org.javatuples.Triplet<Double, Double, Double> getLift()

The aerodynamic lift currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

org.javatuples.Triplet<Double, Double, Double> getDrag()

The aerodynamic drag currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

float getSpeedOfSound()

The speed of sound, in the atmosphere around the vessel, in m/s.

Note: Not available when Ferram Aerospace Research is installed.

float getMach()

The speed of the vessel, in multiples of the speed of sound.

Note: Not available when Ferram Aerospace Research is installed.

float getEquivalentAirSpeed()

The equivalent air speed of the vessel, in m/s.

Note: Not available when Ferram Aerospace Research is installed.

float getTerminalVelocity()

An estimate of the current terminal velocity of the vessel, in m/s. This is the speed at which the drag forces cancel out the force of gravity.

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

float getAngleOfAttack()

Gets the pitch angle between the orientation of the vessel and its velocity vector, in degrees.

float getSideslipAngle()

Gets the yaw angle between the orientation of the vessel and its velocity vector, in degrees.

float getTotalAirTemperature()

The total air temperature of the atmosphere around the vessel, in Kelvin. This temperature includes the Flight.StaticAirTemperature and the vessel's kinetic energy.

float getStaticAirTemperature()

The static (ambient) temperature of the atmosphere around the vessel, in Kelvin.

float getStallFraction()

Gets the current amount of stall, between 0 and 1. A value greater than 0.005 indicates a minor stall and a value greater than 0.5 indicates a large-scale stall.

Note: Requires Ferram Aerospace Research.

float getDragCoefficient()

Gets the coefficient of drag. This is the amount of drag produced by the vessel. It depends on air speed, air density and wing area.

Note: Requires Ferram Aerospace Research.

float getLiftCoefficient()

Gets the coefficient of lift. This is the amount of lift produced by the vessel, and depends on air speed, air density and wing area.

Note: Requires Ferram Aerospace Research.

float getBallisticCoefficient()

Gets the ballistic coefficient.

Note: Requires Ferram Aerospace Research.

float getThrustSpecificFuelConsumption()

Gets the thrust specific fuel consumption for the jet engines on the vessel. This is a measure of the efficiency of the engines, with a lower value indicating a more efficient vessel. This value is the number of Newtons of fuel that are burned, per hour, to product one newton of thrust.

Note: Requires Ferram Aerospace Research.

5.2.5 Orbit

public class Orbit

Describes an orbit. For example, the orbit of a vessel, obtained by calling Vessel.Orbit, or a celestial body, obtained by calling CelestialBody.Orbit.

CelestialBody getBody()

The celestial body (e.g. planet or moon) around which the object is orbiting.

double getApoapsis()

Gets the apoapsis of the orbit, in meters, from the center of mass of the body being orbited.

Note: For the apoapsis altitude reported on the in-game map view, use Orbit. ApoapsisAltitude.

double getPeriapsis()

The periapsis of the orbit, in meters, from the center of mass of the body being orbited.

Note: For the periapsis altitude reported on the in-game map view, use Orbit.PeriapsisAltitude.

double getApoapsisAltitude()

The apoapsis of the orbit, in meters, above the sea level of the body being orbited.

Note: This is equal to Orbit . Apoapsis minus the equatorial radius of the body.

double getPeriapsisAltitude()

The periapsis of the orbit, in meters, above the sea level of the body being orbited.

Note: This is equal to Orbit . Periapsis minus the equatorial radius of the body.

$double \ {\tt getSemiMajorAxis}\ (\,)$

The semi-major axis of the orbit, in meters.

double getSemiMinorAxis()

The semi-minor axis of the orbit, in meters.

double getRadius()

The current radius of the orbit, in meters. This is the distance between the center of mass of the object in orbit, and the center of mass of the body around which it is orbiting.

Note: This value will change over time if the orbit is elliptical.

double getSpeed()

The current orbital speed of the object in meters per second.

Note: This value will change over time if the orbit is elliptical.

double getPeriod()

The orbital period, in seconds.

```
double getTimeToApoapsis()
    The time until the object reaches apoapsis, in seconds.
double getTimeToPeriapsis()
    The time until the object reaches periapsis, in seconds.
double getEccentricity()
    The eccentricity of the orbit.
double getInclination()
    The inclination of the orbit, in radians.
double getLongitudeOfAscendingNode()
    The longitude of the ascending node, in radians.
double getArgumentOfPeriapsis()
    The argument of periapsis, in radians.
double getMeanAnomalyAtEpoch()
    The mean anomaly at epoch.
double getEpoch()
    The time since the epoch (the point at which the mean anomaly at epoch was measured, in seconds.
double getMeanAnomaly()
    The mean anomaly.
double getEccentricAnomaly()
    The eccentric anomaly.
org.javatuples.Triplet<Double, Double, Double>ReferencePlaneNormal(ReferenceFrame refer-
                                                                        enceFrame)
    The unit direction vector that is normal to the orbits reference plane, in the given reference frame. The
    reference plane is the plane from which the orbits inclination is measured.
        Parameters
            • referenceFrame (ReferenceFrame) -
org.javatuples.Triplet<Double, Double, Double>ReferencePlaneDirection (ReferenceFrame
                                                                            referenceFrame)
    The unit direction vector from which the orbits longitude of ascending node is measured, in the given
    reference frame.
        Parameters
            • referenceFrame (ReferenceFrame) -
```

$double \ {\tt getTimeToSOIChange}\ (\)$

The time until the object changes sphere of influence, in seconds. Returns NaN if the object is not going to change sphere of influence.

Orbit getNextOrbit()

If the object is going to change sphere of influence in the future, returns the new orbit after the change. Otherwise returns null.

5.2.6 Control

public class Control

Used to manipulate the controls of a vessel. This includes adjusting the throttle, enabling/disabling systems such as SAS and RCS, or altering the direction in which the vessel is pointing.

Note: Control inputs (such as pitch, yaw and roll) are zeroed when all clients that have set one or more of these inputs are no longer connected.

```
boolean getSAS()
```

void **setSAS** (boolean *value*)

The state of SAS.

Note: Equivalent to AutoPilot.SAS

SASMode getSASMode ()

void setSASMode (SASMode value)

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Note: Equivalent to AutoPilot.SASMode

SpeedMode getSpeedMode()

void setSpeedMode (SpeedMode value)

The current SpeedMode of the navball. This is the mode displayed next to the speed at the top of the navball.

```
boolean getRCS()
```

void setRCS (boolean value)

The state of RCS.

boolean getGear()

void setGear (boolean value)

The state of the landing gear/legs.

boolean getLights()

void setLights (boolean value)

The state of the lights.

boolean **getBrakes** ()

void setBrakes (boolean value)

The state of the wheel brakes.

boolean getAbort ()

void setAbort (boolean value)

The state of the abort action group.

float getThrottle()

void **setThrottle** (float *value*)

The state of the throttle. A value between 0 and 1.

float getPitch()

void setPitch (float value)

The state of the pitch control. A value between -1 and 1. Equivalent to the w and s keys.

```
float getYaw()
void setYaw (float value)
     The state of the yaw control. A value between -1 and 1. Equivalent to the a and d keys.
float getRoll()
void setRoll (float value)
     The state of the roll control. A value between -1 and 1. Equivalent to the q and e keys.
float getForward()
void setForward (float value)
     The state of the forward translational control. A value between -1 and 1. Equivalent to the h and n keys.
float getUp()
void setUp (float value)
     The state of the up translational control. A value between -1 and 1. Equivalent to the i and k keys.
float getRight()
void setRight (float value)
     The state of the right translational control. A value between -1 and 1. Equivalent to the j and l keys.
float getWheelThrottle()
void setWheelThrottle (float value)
     The state of the wheel throttle. A value between -1 and 1. A value of 1 rotates the wheels forwards, a value
     of -1 rotates the wheels backwards.
float getWheelSteering()
void setWheelSteering (float value)
     The state of the wheel steering. A value between -1 and 1. A value of 1 steers to the left, and a value of -1
     steers to the right.
int getCurrentStage()
     The current stage of the vessel. Corresponds to the stage number in the in-game UI.
java.util.List<Vessel> ActivateNextStage()
     Activates the next stage. Equivalent to pressing the space bar in-game.
         Returns A list of vessel objects that are jettisoned from the active vessel.
boolean GetActionGroup (int group)
     Returns true if the given action group is enabled.
         Parameters
             • group (int) – A number between 0 and 9 inclusive.
void SetActionGroup (int group, boolean state)
     Sets the state of the given action group (a value between 0 and 9 inclusive).
         Parameters
             • group (int) – A number between 0 and 9 inclusive.
             • state (boolean) -
void ToggleActionGroup (int group)
     Toggles the state of the given action group.
```

176 Chapter 5. Java

• group (int) – A number between 0 and 9 inclusive.

Parameters

Node AddNode (double *UT*, float *prograde*, float *normal*, float *radial*)

Creates a maneuver node at the given universal time, and returns a *Node* object that can be used to modify it. Optionally sets the magnitude of the delta-v for the maneuver node in the prograde, normal and radial directions.

Parameters

- **UT** (double) Universal time of the maneuver node.
- **prograde** (*float*) Delta-v in the prograde direction.
- **normal** (float) Delta-v in the normal direction.
- radial (float) Delta-v in the radial direction.

java.util.List<Node> getNodes()

Returns a list of all existing maneuver nodes, ordered by time from first to last.

void RemoveNodes ()

Remove all maneuver nodes.

public enum SASMode

The behavior of the SAS auto-pilot. See AutoPilot. SASMode.

public SASMode StabilityAssist

Stability assist mode. Dampen out any rotation.

public SASMode Maneuver

Point in the burn direction of the next maneuver node.

public SASMode Prograde

Point in the prograde direction.

public SASMode Retrograde

Point in the retrograde direction.

public SASMode Normal

Point in the orbit normal direction.

public SASMode AntiNormal

Point in the orbit anti-normal direction.

public SASMode Radial

Point in the orbit radial direction.

public SASMode AntiRadial

Point in the orbit anti-radial direction.

public SASMode Target

Point in the direction of the current target.

public SASMode AntiTarget

Point away from the current target.

public enum SpeedMode

See Control.SpeedMode.

public SpeedMode Orbit

Speed is relative to the vessel's orbit.

public SpeedMode Surface

Speed is relative to the surface of the body being orbited.

public SpeedMode Target

Speed is relative to the current target.

5.2.7 Parts

The following classes allow interaction with a vessels individual parts.

- Parts
- Part
- Module
- Specific Types of Part
 - Decoupler
 - Docking Port
 - Engine
 - Landing Gear
 - Landing Leg
 - Launch Clamp
 - Light
 - Parachute
 - Radiator
 - Resource Converter
 - Resource Harvester
 - Reaction Wheel
 - Sensor
 - Solar Panel
- Trees of Parts
 - Traversing the Tree
 - Attachment Modes
- Fuel Lines
- Staging

Parts

public class Parts

Instances of this class are used to interact with the parts of a vessel. An instance can be obtained by calling Vessel.Parts.

```
java.util.List<Part> getAll()
```

A list of all of the vessels parts.

Part getRoot()

The vessels root part.

Note: See the discussion on java-api-parts-trees-of-parts.

```
Part getControlling()
```

```
void setControlling(Part value)
```

The part from which the vessel is controlled.

java.util.List<Part> WithName (String name)

A list of parts whose Part . Name is name.

Parameters

• name (String) -

java.util.List<*Part*> WithTitle (String *title*)

A list of all parts whose Part. Title is title.

Parameters

• title (String) -

java.util.List<Part> WithModule (String moduleName)

A list of all parts that contain a Module whose Module. Name is moduleName.

Parameters

• moduleName (String) -

java.util.List<*Part*> InStage (int *stage*)

A list of all parts that are activated in the given *stage*.

Parameters

• stage (int) -

Note: See the discussion on java-api-parts-staging.

java.util.List<Part> InDecoupleStage (int stage)

A list of all parts that are decoupled in the given stage.

Parameters

• stage (int) -

Note: See the discussion on java-api-parts-staging.

java.util.List<Module> ModulesWithName (String moduleName)

A list of modules (combined across all parts in the vessel) whose Module. Name is moduleName.

Parameters

• moduleName (String) -

java.util.List<Decoupler> getDecouplers()

A list of all decouplers in the vessel.

java.util.List<DockingPort> getDockingPorts()

A list of all docking ports in the vessel.

DockingPort DockingPortWithName (String name)

The first docking port in the vessel with the given port name, as returned by DockingPort.Name. Returns null if there are no such docking ports.

Parameters

• name (String) -

java.util.List<Engine> getEngines ()

A list of all engines in the vessel.

java.util.List<LandingGear> getLandingGear()

A list of all landing gear attached to the vessel.

java.util.List<LandingLeg> getLandingLegs()

A list of all landing legs attached to the vessel.

Part

```
java.util.List<LaunchClamp> getLaunchClamps ()
           A list of all launch clamps attached to the vessel.
      java.util.List<Light> getLights()
           A list of all lights in the vessel.
      java.util.List<Parachute> getParachutes ()
           A list of all parachutes in the vessel.
      java.util.List<Radiator> getRadiators()
           A list of all radiators in the vessel.
      java.util.List<ResourceConverter> getResourceConverters ()
           A list of all resource converters in the vessel.
      java.util.List<ResourceHarvester> getResourceHarvesters()
           A list of all resource harvesters in the vessel.
      java.util.List<ReactionWheel> getReactionWheels ()
           A list of all reaction wheels in the vessel.
      java.util.List<Sensor> getSensors()
           A list of all sensors in the vessel.
      java.util.List<SolarPanel> getSolarPanels ()
           A list of all solar panels in the vessel.
public class Part
     Instances of this class represents a part. A vessel is made of multiple parts. Instances can be obtained by various
     methods in Parts.
      String getName()
           Internal name of the part, as used in part cfg files. For example "Mark1-2Pod".
      String getTitle()
           Title of the part, as shown when the part is right clicked in-game. For example "Mk1-2 Command Pod".
      double getCost()
           The cost of the part, in units of funds.
      Vessel getVessel()
           The vessel that contains this part.
      Part getParent()
           The parts parent. Returns null if the part does not have a parent. This, in combination with
           Part.Children, can be used to traverse the vessels parts tree.
           Note: See the discussion on java-api-parts-trees-of-parts.
      java.util.List<Part> getChildren()
```

180 Chapter 5. Java

Part.Parent, can be used to traverse the vessels parts tree.

Note: See the discussion on java-api-parts-trees-of-parts.

The parts children. Returns an empty list if the part has no children. This, in combination with

boolean getAxiallyAttached()

Whether the part is axially attached to its parent, i.e. on the top or bottom of its parent. If the part has no parent, returns false.

Note: See the discussion on java-api-parts-attachment-modes.

boolean getRadiallyAttached()

Whether the part is radially attached to its parent, i.e. on the side of its parent. If the part has no parent, returns false.

Note: See the discussion on java-api-parts-attachment-modes.

int getStage()

The stage in which this part will be activated. Returns -1 if the part is not activated by staging.

Note: See the discussion on java-api-parts-staging.

int getDecoupleStage()

The stage in which this part will be decoupled. Returns -1 if the part is never decoupled from the vessel.

Note: See the discussion on java-api-parts-staging.

boolean getMassless()

Whether the part is massless.

double getMass()

The current mass of the part, including resources it contains, in kilograms. Returns zero if the part is massless.

double getDryMass()

The mass of the part, not including any resources it contains, in kilograms. Returns zero if the part is massless.

double getImpactTolerance()

The impact tolerance of the part, in meters per second.

double getTemperature()

Temperature of the part, in Kelvin.

double getSkinTemperature()

Temperature of the skin of the part, in Kelvin.

double **getMaxTemperature**()

Maximum temperature that the part can survive, in Kelvin.

double getMaxSkinTemperature()

Maximum temperature that the skin of the part can survive, in Kelvin.

double getExternalTemperature()

Temperature of the atmosphere/vacuum surrounding the part, in Kelvin. This does not include heating from direct sunlight.

double **getThermalMass**()

How much it takes to heat up the part.

double getThermalSkinMass()

How much it takes to heat up the part's skin.

double getThermalResourceMass()

How much it takes to heat up resources in the part.

double getThermalConductionFlux()

The speed that heat is conducting into or out of the part through contact with other parts. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double getThermalConvectionFlux()

The speed that heat is convecting into or out of the part from the surrounding atmosphere. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double getThermalRadiationFlux()

The speed that heat is radiating into or out of the part from the surrounding vacuum. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double getThermalInternalFlux()

The speed that heat is generated by the part. For example, engines generate heat by burning fuel. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

double getThermalSkinToInternalFlux()

The speed that heat is conducting between the part's skin and its internals.

Resources getResources()

A Resources object for the part.

boolean getCrossfeed()

Whether this part is crossfeed capable.

boolean getIsFuelLine()

Whether this part is a fuel line.

java.util.List<Part> getFuelLinesFrom()

The parts that are connected to this part via fuel lines, where the direction of the fuel line is into this part.

Note: See the discussion on java-api-parts-fuel-lines.

java.util.List<Part> getFuelLinesTo()

The parts that are connected to this part via fuel lines, where the direction of the fuel line is out of this part.

Note: See the discussion on java-api-parts-fuel-lines.

java.util.List<*Module*> getModules ()

The modules for this part.

Decoupler getDecoupler()

A Decoupler if the part is a decoupler, otherwise null.

DockingPort getDockingPort()

A DockingPort if the part is a docking port, otherwise null.

Engine getEngine()

An Engine if the part is an engine, otherwise null.

LandingGear getLandingGear()

A LandingGear if the part is a landing gear, otherwise null.

LandingLeg getLandingLeg()

A LandingLeg if the part is a landing leg, otherwise null.

LaunchClamp getLaunchClamp()

A LaunchClamp if the part is a launch clamp, otherwise null.

Light getLight()

A Light if the part is a light, otherwise null.

Parachute getParachute()

A Parachute if the part is a parachute, otherwise null.

Radiator getRadiator()

A *Radiator* if the part is a radiator, otherwise null.

ReactionWheel getReactionWheel()

A ReactionWheel if the part is a reaction wheel, otherwise null.

ResourceConverter getResourceConverter()

A ResourceConverter if the part is a resource converter, otherwise null.

ResourceHarvester getResourceHarvester()

A ResourceHarvester if the part is a resource harvester, otherwise null.

Sensor getSensor()

A Sensor if the part is a sensor, otherwise null.

SolarPanel getSolarPanel()

A SolarPanel if the part is a solar panel, otherwise null.

org.javatuples.Triplet<Double, Double, Double> Position (ReferenceFrame referenceFrame)

The position of the part in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double> Direction (ReferenceFrame referenceFrame)

The direction of the part in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> **Velocity** (*ReferenceFrame referenceFrame*)

The velocity of the part in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Quartet<Double, Double, Double, Double>Rotation (ReferenceFrame

reference-

Frame)

The rotation of the part in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

ReferenceFrame getReferenceFrame()

The reference frame that is fixed relative to this part.

- •The origin is at the position of the part.
- •The axes rotate with the part.
- •The x, y and z axis directions depend on the design of the part.

Note: For docking port parts, this reference frame is not necessarily equivalent to the reference frame for the docking port, returned by <code>DockingPort.ReferenceFrame</code>.

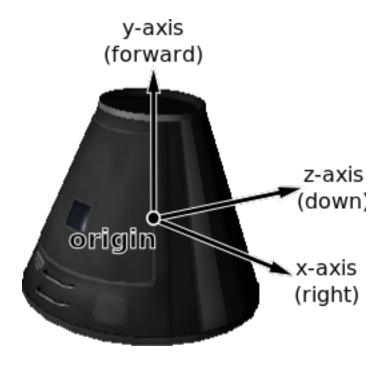


Fig. 5.7: Mk1 Command Pod reference frame origin and axes

Module

public class Module

In KSP, each part has zero or more PartModules associated with it. Each one contains some of the functionality of the part. For example, an engine has a "ModuleEngines" PartModule that contains all the functionality of an engine. This class allows you to interact with KSPs PartModules, and any PartModules that have been added by other mods.

String getName()

Name of the PartModule. For example, "ModuleEngines".

Part getPart()

The part that contains this module.

java.util.Map<String, String> getFields()

The modules field names and their associated values, as a dictionary. These are the values visible in the right-click menu of the part.

boolean **HasField** (String name)

Returns true if the module has a field with the given name.

Parameters

• name (String) - Name of the field.

String GetField (String name)

Returns the value of a field.

Parameters

• name (String) - Name of the field.

java.util.List<String> getEvents()

A list of the names of all of the modules events. Events are the clickable buttons visible in the right-click menu of the part.

boolean **HasEvent** (String name)

true if the module has an event with the given name.

Parameters

• name (String) -

void TriggerEvent (String name)

Trigger the named event. Equivalent to clicking the button in the right-click menu of the part.

Parameters

• name (String) -

java.util.List<String> getActions()

A list of all the names of the modules actions. These are the parts actions that can be assigned to action groups in the in-game editor.

boolean **HasAction** (String name)

true if the part has an action with the given name.

Parameters

• name (String) -

void SetAction (String name, boolean value)

Set the value of an action with the given name.

Parameters

- name (String) -
- value (boolean) -

Specific Types of Part

The following classes provide functionality for specific types of part.

- Decoupler
- Docking Port
- Engine
- Landing Gear
- Landing Leg
- Launch Clamp
- Light
- Parachute
- Radiator
- Resource Converter
- Resource Harvester
- Reaction Wheel
- Sensor
- Solar Panel

Decoupler

```
public class Decoupler
  Obtained by calling Part.Decoupler

Part getPart()
    The part object for this decoupler.

void Decouple()
    Fires the decoupler. Has no effect if the decoupler has already fired.

boolean getDecoupled()
    Whether the decoupler has fired.

float getImpulse()
    The impulse that the decoupler imparts when it is fired, in Newton seconds.
```

Docking Port

public class DockingPort

Obtained by calling Part.DockingPort

Part getPart()

The part object for this docking port.

String getName()

void setName (String value)

The port name of the docking port. This is the name of the port that can be set in the right click menu, when the Docking Port Alignment Indicator mod is installed. If this mod is not installed, returns the title of the part (Part.Title).

DockingPortState getState()

The current state of the docking port.

Part getDockedPart ()

The part that this docking port is docked to. Returns null if this docking port is not docked to anything.

Vessel Undock ()

Undocks the docking port and returns the vessel that was undocked from. After undocking, the active vessel may change (ActiveVessel). This method can be called for either docking port in a docked pair - both calls will have the same effect. Returns null if the docking port is not docked to anything.

float getReengageDistance()

The distance a docking port must move away when it undocks before it becomes ready to dock with another port, in meters.

boolean getHasShield()

Whether the docking port has a shield.

boolean getShielded()

void setShielded (boolean value)

The state of the docking ports shield, if it has one. Returns true if the docking port has a shield, and the shield is closed. Otherwise returns false. When set to true, the shield is closed, and when set to false the shield is opened. If the docking port does not have a shield, setting this attribute has no effect.

org.javatuples.Triplet<Double, Double, Double>Position (ReferenceFrame referenceFrame)

The position of the docking port in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

 $org. javatuples. Triplet < Double, \ Double > \textbf{Direction} \ (\textit{ReferenceFrame referenceFrame})$

The direction that docking port points in, in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Quartet<Double, Double, Double, Double> Rotation (ReferenceFrame

reference-

Frame)

The rotation of the docking port, in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

ReferenceFrame getReferenceFrame()

The reference frame that is fixed relative to this docking port, and oriented with the port.

- •The origin is at the position of the docking port.
- •The axes rotate with the docking port.
- •The x-axis points out to the right side of the docking port.
- •The y-axis points in the direction the docking port is facing.
- •The z-axis points out of the bottom off the docking port.

Note: This reference frame is not necessarily equivalent to the reference frame for the part, returned by Part.ReferenceFrame.

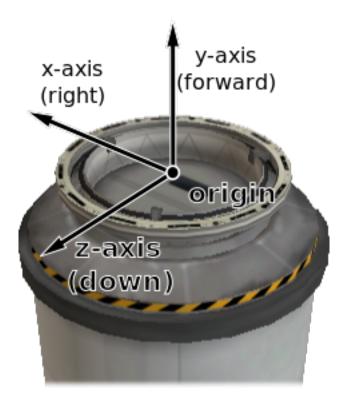


Fig. 5.8: Docking port reference frame origin and axes

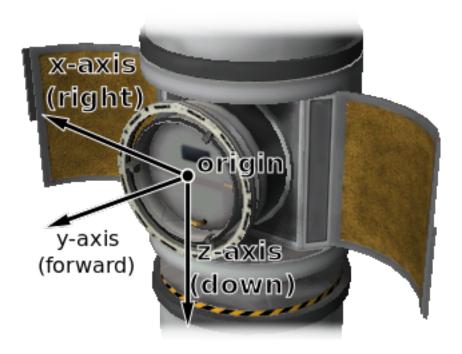


Fig. 5.9: Inline docking port reference frame origin and axes

public enum DockingPortState

See DockingPort.State.

public DockingPortState Ready

The docking port is ready to dock to another docking port.

public DockingPortState Docked

The docking port is docked to another docking port, or docked to another part (from the VAB/SPH).

public DockingPortState Docking

The docking port is very close to another docking port, but has not docked. It is using magnetic force to acquire a solid dock.

public DockingPortState Undocking

The docking port has just been undocked from another docking port, and is disabled until it moves away by a sufficient distance (DockingPort.ReengageDistance).

public DockingPortState Shielded

The docking port has a shield, and the shield is closed.

public DockingPortState Moving

The docking ports shield is currently opening/closing.

Engine

public class Engine

Obtained by calling Part. Engine.

Part getPart()

The part object for this engine.

boolean **getActive**()

void **setActive** (boolean *value*)

Whether the engine is active. Setting this attribute may have no effect, depending on Engine.CanShutdown and Engine.CanRestart.

float getThrust()

The current amount of thrust being produced by the engine, in Newtons. Returns zero if the engine is not active or if it has no fuel.

float getAvailableThrust()

The maximum available amount of thrust that can be produced by the engine, in Newtons. This takes Engine. ThrustLimit into account, and is the amount of thrust produced by the engine when activated and the main throttle is set to 100%. Returns zero if the engine does not have any fuel.

float getMaxThrust()

Gets the maximum amount of thrust that can be produced by the engine, in Newtons. This is the amount of thrust produced by the engine when activated, Engine. ThrustLimit is set to 100% and the main vessel's throttle is set to 100%.

float getMaxVacuumThrust()

The maximum amount of thrust that can be produced by the engine in a vacuum, in Newtons. This is the amount of thrust produced by the engine when activated, Engine. ThrustLimit is set to 100%, the main vessel's throttle is set to 100% and the engine is in a vacuum.

float getThrustLimit()

void setThrustLimit (float value)

The thrust limiter of the engine. A value between 0 and 1. Setting this attribute may have no effect, for example the thrust limit for a solid rocket booster cannot be changed in flight.

float getSpecificImpulse()

The current specific impulse of the engine, in seconds. Returns zero if the engine is not active.

float getVacuumSpecificImpulse()

The vacuum specific impulse of the engine, in seconds.

float getKerbinSeaLevelSpecificImpulse()

The specific impulse of the engine at sea level on Kerbin, in seconds.

java.util.List<String> getPropellants()

The names of resources that the engine consumes.

java.util.Map<String, Single> getPropellantRatios()

The ratios of resources that the engine consumes. A dictionary mapping resource names to the ratios at which they are consumed by the engine.

boolean getHasFuel()

Whether the engine has run out of fuel (or flamed out).

float getThrottle()

The current throttle setting for the engine. A value between 0 and 1. This is not necessarily the same as the vessel's main throttle setting, as some engines take time to adjust their throttle (such as jet engines).

boolean getThrottleLocked()

Whether the Control. Throttle affects the engine. For example, this is true for liquid fueled rockets, and false for solid rocket boosters.

boolean getCanRestart()

Whether the engine can be restarted once shutdown. If the engine cannot be shutdown, returns false. For example, this is true for liquid fueled rockets and false for solid rocket boosters.

```
boolean getCanShutdown()
          Gets whether the engine can be shutdown once activated. For example, this is true for liquid fueled
          rockets and false for solid rocket boosters.
      boolean getGimballed()
          Whether the engine nozzle is gimballed, i.e. can provide a turning force.
      float getGimbalRange()
          The range over which the gimbal can move, in degrees.
      boolean getGimbalLocked()
      void setGimbalLocked (boolean value)
          Whether the engines gimbal is locked in place. Setting this attribute has no effect if the engine is not
          gimballed.
      float getGimbalLimit()
      void setGimbalLimit (float value)
          The gimbal limiter of the engine. A value between 0 and 1. Returns 0 if the gimbal is locked or the engine
          is not gimballed. Setting this attribute has no effect if the engine is not gimballed.
Landing Gear
public class LandingGear
     Obtained by calling Part. LandingGear.
      Part getPart()
          The part object for this landing gear.
      LandingGearState getState()
          Gets the current state of the landing gear.
      boolean getDeployed()
      void setDeployed (boolean value)
          Whether the landing gear is deployed.
public enum LandingGearState
     See LandingGear.State.
     public Landing Gear State Deployed
          Landing gear is fully deployed.
     public Landing Gear State Retracted
          Landing gear is fully retracted.
     public Landing Gear State Deploying
          Landing gear is being deployed.
     public LandingGearState Retracting
          Landing gear is being retracted.
Landing Leg
public class LandingLeg
     Obtained by calling Part.LandingLeg.
      Part getPart()
          The part object for this landing leg.
```

```
LandingLegState getState()
          The current state of the landing leg.
      boolean getDeployed()
      void setDeployed (boolean value)
          Whether the landing leg is deployed.
public enum LandingLegState
     See LandingLeg.State.
     public LandingLegState Deployed
          Landing leg is fully deployed.
     public LandingLegState Retracted
          Landing leg is fully retracted.
     public LandingLegState Deploying
          Landing leg is being deployed.
     public LandingLegState Retracting
          Landing leg is being retracted.
     public LandingLegState Broken
          Landing leg is broken.
     public LandingLegState Repairing
          Landing leg is being repaired.
Launch Clamp
public class LaunchClamp
     Obtained by calling Part.LaunchClamp.
      Part getPart()
          The part object for this launch clamp.
      void Release()
          Releases the docking clamp. Has no effect if the clamp has already been released.
Light
public class Light
     Obtained by calling Part.Light.
      Part getPart()
          The part object for this light.
      boolean getActive()
      void setActive (boolean value)
          Whether the light is switched on.
      float getPowerUsage()
          The current power usage, in units of charge per second.
```

Parachute

```
public class Parachute
     Obtained by calling Part.Parachute.
      Part getPart()
          The part object for this parachute.
      void Deploy()
          Deploys the parachute. This has no effect if the parachute has already been deployed.
      boolean getDeployed()
          Whether the parachute has been deployed.
      ParachuteState getState()
          The current state of the parachute.
      float getDeployAltitude()
      void setDeployAltitude (float value)
          The altitude at which the parachute will full deploy, in meters.
      float getDeployMinPressure()
      void setDeployMinPressure (float value)
          The minimum pressure at which the parachute will semi-deploy, in atmospheres.
public enum ParachuteState
     See Parachute. State.
     public ParachuteState Stowed
          The parachute is safely tucked away inside its housing.
     public ParachuteState Active
          The parachute is still stowed, but ready to semi-deploy.
     public ParachuteState SemiDeployed
          The parachute has been deployed and is providing some drag, but is not fully deployed yet.
     public ParachuteState Deployed
          The parachute is fully deployed.
     public ParachuteState Cut
          The parachute has been cut.
Radiator
public class Radiator
     Obtained by calling Part.Radiator.
      Part getPart()
          The part object for this radiator.
      boolean getDeployed()
      void setDeployed (boolean value)
          Whether the radiator is extended.
```

192 Chapter 5. Java

RadiatorState getState()

The current state of the radiator.

public enum RadiatorState

```
RadiatorState
```

public RadiatorState Extended

Radiator is fully extended.

public RadiatorState Retracted

Radiator is fully retracted.

public RadiatorState Extending

Radiator is being extended.

public RadiatorState Retracting

Radiator is being retracted.

public RadiatorState Broken

Radiator is being broken.

Resource Converter

public class ResourceConverter

Obtained by calling Part . ResourceConverter.

Part getPart()

The part object for this converter.

int getCount()

The number of converters in the part.

String Name (int *index*)

The name of the specified converter.

Parameters

• index (int) – Index of the converter.

boolean **Active** (int *index*)

True if the specified converter is active.

Parameters

• index (int) – Index of the converter.

void Start (int index)

Start the specified converter.

Parameters

• index (int) – Index of the converter.

void Stop (int index)

Stop the specified converter.

Parameters

• index (int) – Index of the converter.

ResourceConverterState State (int index)

The state of the specified converter.

Parameters

• index (int) – Index of the converter.

```
String StatusInfo (int index)
          Status information for the specified converter. This is the full status message shown in the in-game UI.
               Parameters
                   • index (int) – Index of the converter.
      java.util.List<String> Inputs (int index)
          List of the names of resources consumed by the specified converter.
              Parameters
                   • index (int) – Index of the converter.
      java.util.List<String>Outputs (int index)
          List of the names of resources produced by the specified converter.
              Parameters
                   • index (int) – Index of the converter.
public enum ResourceConverterState
     See ResourceConverter.State.
     public ResourceConverterState Running
          Converter is running.
     public ResourceConverterState Idle
          Converter is idle.
     public ResourceConverterState MissingResource
          Converter is missing a required resource.
     public ResourceConverterState StorageFull
          No available storage for output resource.
     public Resource Converter State Capacity
          At preset resource capacity.
     public ResourceConverterState Unknown
                                Possible with modified resource converters.
          Unknown state.
                                                                                    In this case,
                                                                                                     check
          ResourceConverter.StatusInfo for more information.
Resource Harvester
public class ResourceHarvester
     Obtained by calling Part. Resource Harvester.
      Part getPart()
          The part object for this harvester.
      ResourceHarvesterState getState()
          The state of the harvester.
      boolean getDeployed()
      void setDeployed (boolean value)
          Whether the harvester is deployed.
      boolean getActive()
      void setActive (boolean value)
```

194 Chapter 5. Java

Whether the harvester is actively drilling.

```
float getExtractionRate()
```

The rate at which the drill is extracting ore, in units per second.

float getThermalEfficiency()

The thermal efficiency of the drill, as a percentage of its maximum.

float getCoreTemperature()

The core temperature of the drill, in Kelvin.

float getOptimumCoreTemperature()

The core temperature at which the drill will operate with peak efficiency, in Kelvin.

Reaction Wheel

```
public class ReactionWheel
```

 $Obtained \ by \ calling \ {\tt Part.ReactionWheel}.$

```
Part getPart()
```

The part object for this reaction wheel.

```
boolean getActive()
```

void **setActive** (boolean *value*)

Whether the reaction wheel is active.

boolean getBroken()

Whether the reaction wheel is broken.

float getPitchTorque()

The torque in the pitch axis, in Newton meters.

float getYawTorque()

The torque in the yaw axis, in Newton meters.

float getRollTorque()

The torque in the roll axis, in Newton meters.

Sensor

public class Sensor

Obtained by calling Part . Sensor.

Part getPart()

The part object for this sensor.

boolean getActive()

void **setActive** (boolean *value*)

Whether the sensor is active.

String getValue()

The current value of the sensor.

float getPowerUsage()

The current power usage of the sensor, in units of charge per second.

Solar Panel

public class SolarPanel

Obtained by calling Part.SolarPanel.

Part getPart()

The part object for this solar panel.

boolean getDeployed()

void **setDeployed** (boolean *value*)

Whether the solar panel is extended.

SolarPanelState getState()

The current state of the solar panel.

float getEnergyFlow()

The current amount of energy being generated by the solar panel, in units of charge per second.

float getSunExposure()

The current amount of sunlight that is incident on the solar panel, as a percentage. A value between 0 and 1.

public enum SolarPanelState

See SolarPanel.State.

public SolarPanelState Extended

Solar panel is fully extended.

$public {\it Solar Panel State} \ {\bf Retracted}$

Solar panel is fully retracted.

public SolarPanelState Extending

Solar panel is being extended.

public SolarPanelState Retracting

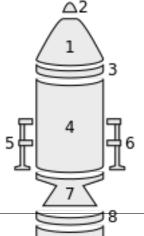
Solar panel is being retracted.

public SolarPanelState Broken

Solar panel is broken.

Trees of Parts

Vessels in KSP are comprised of a number of parts, connected to one another and the corresponding tree of tree structure. example vessel is shown in Figure 1, An craft file parts in Figure 2. The for this example can also be downloaded here.



Traversing the Tree

The tree of parts can be traversed using the attributes SpaceCenter.Parts.Root, SpaceCenter.Part.Parent and SpaceCenter.Part.Children.

The root of the tree is the same as the vessels *root part* (part number 1 in the example above) and can be obtained by calling SpaceCenter.Parts.Root.





A parts children can be obtained by calling SpaceCenter.Part.Children. If the part does not have any children, SpaceCenter.Part.Children returns an empty list. A parts parent can be obtained by calling SpaceCenter.Part.Parent. If the part does not have a parent (as is the case for the root part), SpaceCenter.Part.Parent returns null.

The following Java example uses these attributes to perform a depth-first traversal over all of the parts in a vessel:

```
import java.io.IOException;
import java.util.ArrayDeque;
import java.util.Deque;
import org.javatuples.Pair;
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Part;
import krpc.client.services.SpaceCenter.Vessel;
public class TreeTraversal {
    public static void main(String | args) throw
        Connection connection = Connection.newI
        Vessel vessel = SpaceCenter.newInstance
        Part root = vessel.getParts().getRoot()
        Deque<Pair<Part, Integer>> stack = new A
        stack.push(new Pair<Part, Integer>(root)
        while (stack.size() > 0) {
            Pair<Part, Integer> item = stack.pop
            Part part = item.getValue0();
            int depth = item.getValue1();
            String prefix = "";
            for (int i = 0; i < depth; i++)</pre>
                prefix += " ";
            System.out.println(prefix + part.get
            for (Part child : part getChildren()
                stack.push (new Pair < Part, Intege
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1
TR-18A Stack Decoupler
FL-T400 Fuel Tank
LV-909 Liquid Fuel Engine
TR-18A Stack Decoupler
FL-T800 Fuel Tank
```

```
LV-909 Liquid Fuel Engine
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
       TT18-A Launch Stability Enhancer
       FTX-2 External Fuel Duct
       LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
       TT18-A Launch Stability Enhancer
       FTX-2 External Fuel Duct
       LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
  LT-1 Landing Struts
  LT-1 Landing Struts
Mk16 Parachute
```

Attachment Modes

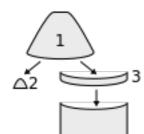
Parts can be attached to other parts either *radially* (on the side of the parent part) or *axially* (on the end of the parent part, to form a stack).

For example, in the vessel pictured above, the parachute (part 2) is *axially* connected to its

parent (the command pod – part 1), and the landing leg (part 5) is *radially* connected to its parent (the fuel tank – part 4).

The root part of a vessel (for example the command pod – part 1) does not have a parent part, so does not have an attachment mode. However, the part is consider to be *axially* attached to nothing.

The following Java example does a depth-first traversal as before, but also prints out the attachment mode used by the part:



```
import java.io.IOException;
import java.util.ArrayDeque;
import java.util.Deque;
import org.javatuples.Pair;
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Part;
import krpc.client.services.SpaceCenter.Vessel;
public class AttachmentModes {
   public static void main(String[] args) throws IOException, RPCException {
        Connection connection = Connection.newInstance();
        Vessel vessel = SpaceCenter.newInstance(connection).getActiveVessel();
        Part root = vessel.getParts().getRoot();
        Deque<Pair<Part, Integer>> stack = new ArrayDeque<Pair<Part, Integer>>();
        stack.push(new Pair<Part, Integer>(root, 0));
```

```
while (stack.size() > 0) {
    Pair<Part, Integer> item = stack.pop();
    Part part = item.getValue0();
    int depth = item.getValue1();
    String prefix = "";
    for (int i = 0; i < depth; i++)
        prefix += " ";
    String attachMode = part.getAxiallyAttached() ? "axial" : "radial";
    System.out.println(prefix + part.getTitle() + " - " + attachMode);
    for (Part child : part.getChildren())
        stack.push(new Pair<Part, Integer>(child, depth + 1));
}
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1 - axial
TR-18A Stack Decoupler - axial
 FL-T400 Fuel Tank - axial
  LV-909 Liquid Fuel Engine - axial
   TR-18A Stack Decoupler - axial
    FL-T800 Fuel Tank - axial
     LV-909 Liquid Fuel Engine - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
  LT-1 Landing Struts - radial
  LT-1 Landing Struts - radial
Mk16 Parachute - axial
```

Fuel Lines

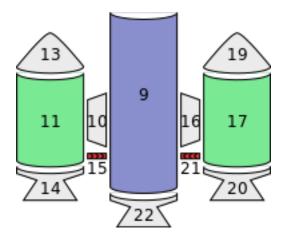


Fig. 5.12: **Figure 5** – Fuel lines from the example in Figure 1. Fuel flows from the parts highlighted in green, into the part highlighted in blue.

Fuel lines are considered parts, and are included in the parts tree (for example, as pictured in Figure 4). However, the parts tree does not contain information about which parts fuel lines connect to. The parent part of a fuel line is the part from which it will take fuel (as shown in Figure 4) however the part that it will send fuel to is not represented in the parts tree.

Figure 5 shows the fuel lines from the example vessel pictured earlier. Fuel line part 15 (in red) takes fuel from a fuel tank (part 11 - in green) and feeds it into another fuel tank (part 9 - in blue). The fuel line is therefore a child of part 11, but its connection to part 9 is not represented in the tree.

The attributes SpaceCenter.Part.FuelLinesFrom and SpaceCenter.Part.FuelLinesTo can be used to discover these connections. In the example in Figure 5, when SpaceCenter.Part.FuelLinesTo is called on fuel tank part 11, it will return a list of parts containing just fuel tank part 9 (the blue part). When SpaceCenter.Part.FuelLinesFrom is called on fuel tank part 9, it will return a list containing fuel tank parts 11 and 17 (the parts colored green).

Staging

Each part has two staging numbers associated with it: the stage in which the part is *activated* and the stage in which the part is *decoupled*. These values can be obtained using SpaceCenter.Part.Stage and SpaceCenter.Part.DecoupleStage respectively. For parts that are not activated by staging, SpaceCenter.Part.Stage returns -1. For parts that are never decoupled, SpaceCenter.Part.DecoupleStage returns a value of -1.

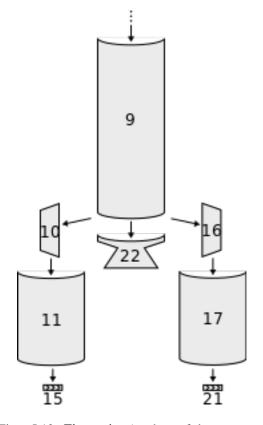


Fig. 5.13: **Figure 4** – A subset of the parts tree from Figure 2 above.

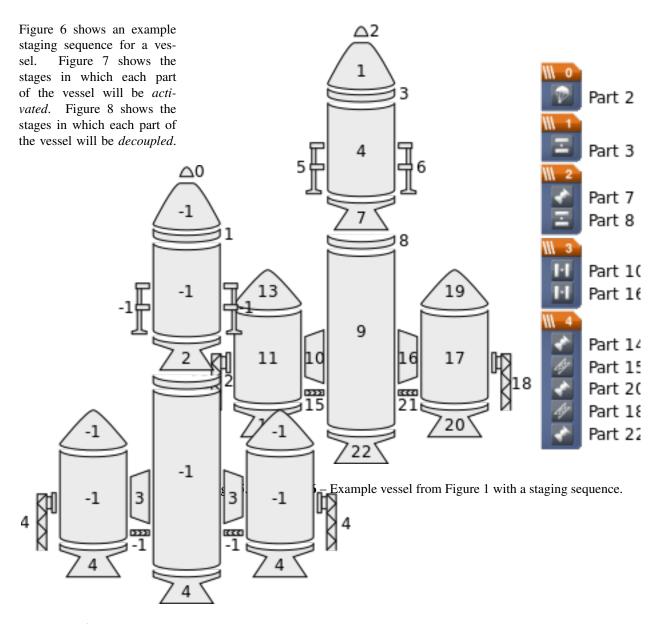


Fig. 5.15: **Figure 7** – The stage in which each part is *activated*.

5.2.8 Resources

public class Resources

Created by calling Vessel.Resources, Vessel.ResourcesInDecoupleStage or Part.Resources.

java.util.List<String> getNames()

A list of resource names that can be stored.

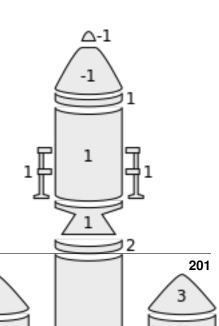
boolean HasResource (String name)

Check whether the named resource can be stored.

Parameters

• name (String) – The name of the resource.

5.2. SpaceCenter API



float **Max** (String name)

Returns the amount of a resource that can be stored.

Parameters

• name (String) – The name of the resource.

float **Amount** (String name)

Returns the amount of a resource that is currently stored.

Parameters

• name (String) - The name of the resource.

float Density (String name)

Returns the density of a resource, in kg/l.

Parameters

• name (String) – The name of the resource.

ResourceFlowMode FlowMode (String name)

Returns the flow mode of a resource.

Parameters

• name (String) – The name of the resource.

public enum ResourceFlowMode

See Resources.FlowMode.

public ResourceFlowMode Vessel

The resource flows to any part in the vessel. For example, electric charge.

public ResourceFlowMode Stage

The resource flows from parts in the first stage, followed by the second, and so on. For example, mono-propellant.

public ResourceFlowMode Adjacent

The resource flows between adjacent parts within the vessel. For example, liquid fuel or oxidizer.

$public \ \textit{ResourceFlowMode} \ \textbf{None}$

The resource does not flow. For example, solid fuel.

5.2.9 Node

public class Node

Represents a maneuver node. Can be created using Control.AddNode.

float getPrograde()

void setPrograde (float value)

The magnitude of the maneuver nodes delta-v in the prograde direction, in meters per second.

float getNormal()

void setNormal (float value)

The magnitude of the maneuver nodes delta-v in the normal direction, in meters per second.

float getRadial()

void setRadial (float value)

The magnitude of the maneuver nodes delta-v in the radial direction, in meters per second.

float getDeltaV()

void setDeltaV (float value)

The delta-v of the maneuver node, in meters per second.

Note: Does not change when executing the maneuver node. See Node. RemainingDeltaV.

float getRemainingDeltaV()

Gets the remaining delta-v of the maneuver node, in meters per second. Changes as the node is executed. This is equivalent to the delta-v reported in-game.

org.javatuples.Triplet<Double, Double, Double>BurnVector (ReferenceFrame referenceFrame)

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s.

Parameters

• referenceFrame (ReferenceFrame) -

Note: Does not change when executing the maneuver node. See Node. RemainingBurnVector.

org.javatuples.Triplet<Double, Double, Double> RemainingBurnVector (ReferenceFrame referenceFrame)

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s. The direction and magnitude change as the burn is executed.

Parameters

• referenceFrame (ReferenceFrame) -

double getUT()

void setUT (double value)

The universal time at which the maneuver will occur, in seconds

double getTimeTo()

The time until the maneuver node will be encountered, in seconds.

Orbit getOrbit()

The orbit that results from executing the maneuver node.

void Remove ()

Removes the maneuver node.

ReferenceFrame getReferenceFrame()

Gets the reference frame that is fixed relative to the maneuver node's burn.

- •The origin is at the position of the maneuver node.
- •The y-axis points in the direction of the burn.
- The x-axis and z-axis point in arbitrary but fixed directions.

ReferenceFrame getOrbitalReferenceFrame()

Gets the reference frame that is fixed relative to the maneuver node, and orientated with the orbital prograde/normal/radial directions of the original orbit at the maneuver node's position.

- •The origin is at the position of the maneuver node.
- •The x-axis points in the orbital anti-radial direction of the original orbit, at the position of the maneuver node.
- •The y-axis points in the orbital prograde direction of the original orbit, at the position of the maneuver node.
- •The z-axis points in the orbital normal direction of the original orbit, at the position of the maneuver node.

org.javatuples.Triplet<Double, Double, Double>Position (ReferenceFrame referenceFrame)

Returns the position vector of the maneuver node in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

org.javatuples.Triplet<Double, Double, Double> **Direction** (ReferenceFrame referenceFrame)

Returns the unit direction vector of the maneuver nodes burn in the given reference frame.

Parameters

• referenceFrame (ReferenceFrame) -

5.2.10 Comms

public class Comms

Used to interact with RemoteTech. Created using a

call to Vessel. Comms.

Note: This class requires RemoteTech to be installed.

boolean getHasLocalControl()

Whether the vessel can be controlled locally.

boolean getHasFlightComputer()

Whether the vessel has a RemoteTech flight computer on board.

boolean getHasConnection()

Whether the vessel can receive commands from the KSC or a command station.

boolean getHasConnectionToGroundStation()

Whether the vessel can transmit science data to a ground station.

double getSignalDelay()

The signal delay when sending commands to the vessel, in seconds.

double getSignalDelayToGroundStation()

The signal delay between the vessel and the closest ground station, in seconds.

double SignalDelayToVessel (Vessel other)

Returns the signal delay between the current vessel and another vessel, in seconds.

Parameters

• other (Vessel) -

5.2.11 ReferenceFrame

public class ReferenceFrame

Represents a reference frame for positions, rotations and velocities. Contains:

- •The position of the origin.
- •The directions of the x, y and z axes.
- •The linear velocity of the frame.
- •The angular velocity of the frame.

Note: This class does not contain any properties or methods. It is only used as a parameter to other functions.

5.2.12 AutoPilot

```
public class AutoPilot
     Provides basic auto-piloting utilities for a vessel.
     Created by calling Vessel. AutoPilot.
 void Engage()
     Engage the auto-pilot.
 void Disengage ()
     Disengage the auto-pilot.
 void Wait()
     Blocks until the vessel is pointing in the target di-
     rection (if set) and has the target roll (if set).
 float getError()
     The error, in degrees, between the direction the
     ship has been asked to point in and the direction
     it is pointing in. Returns zero if the auto-pilot has
     not been engaged, SAS is not enabled, SAS is in
     stability assist mode, or no target direction is set.
 float getRollError()
     The error, in degrees, between the roll the ship has
     been asked to be in and the actual roll. Returns zero
     if the auto-pilot has not been engaged or no target
     roll is set.
 ReferenceFrame getReferenceFrame()
 void setReferenceFrame (ReferenceFrame value)
     The reference frame for the target direction
     (AutoPilot.TargetDirection).
 org.javatuples.Triplet<Double, Double, Double> getTargetDirection()
 void setTargetDirection (org.javatuples.Triplet<Double, Double, Double> value)
     The target direction. null if no target direction is
 void TargetPitchAndHeading (float pitch, float heading)
     Set (AutoPilot.TargetDirection) from a
     pitch and heading angle.
 Parameters
   • pitch (float) - Target pitch angle, in degrees be-
     tween -90^{\circ} and +90^{\circ}.
   • heading (float) - Target heading angle, in de-
     grees between 0° and 360°.
 float getTargetRoll()
 void setTargetRoll (float value)
     The target roll, in degrees. NaN if no target roll is
```

boolean getSAS() void **setSAS** (boolean *value*) The state of SAS. Note: Equivalent to Control.SAS SASMode getSASMode () void setSASMode (SASMode value) The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled. Note: Equivalent to Control. SASMode float getRotationSpeedMultiplier() void setRotationSpeedMultiplier (float value) Target rotation speed multiplier. Defaults to 1. float getMaxRotationSpeed() void setMaxRotationSpeed (float value) Maximum target rotation speed. Defaults to 1. float getRollSpeedMultiplier() void setRollSpeedMultiplier (float value) Target roll speed multiplier. Defaults to 1. float getMaxRollSpeed()

void **setMaxRollSpeed** (float *value*)

Maximum target roll speed. Defaults to 1.

void SetPIDParameters (float Kp, float Ki, float Kd)

Sets the gains for the rotation rate PID controller.

Parameters

- **Kp** (float) Proportional gain.
- **Ki** (float) Integral gain.
- Kd (float) Derivative gain.

5.2.13 Geometry Types

3-dimensional vectors are represented as a 3-tuple. For example:

```
import java.io.IOException;
import org.javatuples.Triplet;
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Vessel;

public class Vector3 {
    public static void main(String[] args) throws IOException, RPCException {
        Connection connection = Connection.newInstance();
        Vessel vessel = SpaceCenter.newInstance(connection).getActiveVessel();
        Triplet<Double, Double, Double> v = vessel.flight(null).getPrograde();
        System.out.println(v.getValue0() + "," + v.getValue1() + "," + v.getValue2());
    }
}
```

Quaternions (rotations in 3-dimensional space) are encoded as a 4-tuple containing the x, y, z and w components. For example:

```
import java.io.IOException;
import org.javatuples.Quartet;
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Vessel;

public class Quaternion {
    public static void main(String[] args) throws IOException, RPCException {
        Connection connection = Connection.newInstance();
        Vessel vessel = SpaceCenter.newInstance(connection).getActiveVessel();
        Quartet<Double, Double, Double> q = vessel.flight(null).getRotation();
        System.out.println(q.getValue0() + "," + q.getValue1() + "," + q.getValue2() + "," + q.getVa
```

5.3 InfernalRobotics API

Provides RPCs to interact with the InfernalRobotics mod. Provides the following classes:

5.3.1 InfernalRobotics

public class InfernalRobotics

This service provides functionality to interact with the InfernalRobotics mod.

```
java.util.List<ControlGroup> getServoGroups ()
```

A list of all the servo groups in the active vessel.

ControlGroup ServoGroupWithName (String name)

Returns the servo group with the given *name* or null if none exists. If multiple servo groups have the same name, only one of them is returned.

Parameters

• name (String) - Name of servo group to find.

Servo ServoWithName (String name)

Returns the servo with the given *name*, from all servo groups, or null if none exists. If multiple servos have the same name, only one of them is returned.

Parameters

• name (String) - Name of the servo to find.

5.3.2 ControlGroup

public class ControlGroup

A group of servos, obtained by calling ServoGroups or ServoGroupWithName. Represents the "Servo Groups" in the Infernal-Robotics UI.

```
String getName()
```

void setName (String value)

The name of the group.

String getForwardKey()

void setForwardKey (String value)

The key assigned to be the "forward" key for the group.

String getReverseKey()

void setReverseKey (String value)

The key assigned to be the "reverse" key for the group.

float getSpeed()

void setSpeed (float value)

The speed multiplier for the group.

boolean getExpanded()

void setExpanded (boolean value)

Whether the group is expanded in the Infernal-Robotics UI.

java.util.List<*Servo*> **getServos**() The servos that are in the group.

Servo ServoWithName (String name)

Returns the servo with the given *name* from this group, or null if none exists.

Parameters

• name (String) - Name of servo to find.

void MoveRight()

Moves all of the servos in the group to the right.

void MoveLeft ()

Moves all of the servos in the group to the left.

void MoveCenter()

Moves all of the servos in the group to the center.

void MoveNextPreset()

Moves all of the servos in the group to the next preset.

void MovePrevPreset ()

Moves all of the servos in the group to the previous preset.

void Stop()

Stops the servos in the group.

5.3.3 Servo

public class Servo

Represents a servo. Obtained using ControlGroup.Servos, ControlGroup.ServoWithName or ServoWithName.

String getName()

void setName (String value)

The name of the servo.

void setHighlight (boolean value)

Whether the servo should be highlighted in-game.

float getPosition()

The position of the servo.

float getMinConfigPosition()

The minimum position of the servo, specified by the part configuration.

float getMaxConfigPosition()

The maximum position of the servo, specified by the part configuration.

float getMinPosition()

void **setMinPosition** (float *value*) The minimum position of the servo, specified by the in-game tweak menu. float getMaxPosition() void **setMaxPosition** (float value) The maximum position of the servo, specified by the in-game tweak menu. float getConfigSpeed() The speed multiplier of the servo, specified by the part configuration. float getSpeed() void setSpeed (float value) The speed multiplier of the servo, specified by the in-game tweak menu. float getCurrentSpeed() void setCurrentSpeed (float value) The current speed at which the servo is moving. float getAcceleration() void setAcceleration (float value) The current speed multiplier set in the UI. boolean getIsMoving() Whether the servo is moving. boolean getIsFreeMoving() Whether the servo is freely moving. boolean getIsLocked() void setIsLocked (boolean value) Whether the servo is locked. boolean getIsAxisInverted() void **setIsAxisInverted** (boolean *value*) Whether the servos axis is inverted. void MoveRight() Moves the servo to the right. void MoveLeft() Moves the servo to the left. void MoveCenter() Moves the servo to the center. void MoveNextPreset () Moves the servo to the next preset.

```
void MovePrevPreset ()
```

Moves the servo to the previous preset.

void MoveTo (float position, float speed)

Moves the servo to *position* and sets the speed multiplier to *speed*.

Parameters

- **position** (*float*) The position to move the serve to.
- **speed** (*float*) Speed multiplier for the movement.

```
void Stop()
```

Stops the servo.

5.3.4 Example

The following example gets the control group named "MyGroup", prints out the names and positions of all of the servos in the group, then moves all of the servos to the right for 1 second.

```
import java.io.IOException;
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.InfernalRobotics;
import krpc.client.services.InfernalRobotics.ControlGroup;
import krpc.client.services.InfernalRobotics.Servo;
public class InfernalRoboticsExample {
   public static void main(String[] args) throws IOException, RPCException, Interrupted Exception {
        Connection connection = Connection.newInstance("InfernalRobotics Example");
        InfernalRobotics ir = InfernalRobotics.newInstance(connection);
        ControlGroup group = ir.servoGroupWithName("MyGroup");
        if (group == null) {
            System.out.println("Group not found");
            return;
        for (Servo servo : group.getServos())
            System.out.println(servo.getName() + " " + servo.getPosition());
        group.moveRight();
        Thread.sleep(1000);
        group.stop();
```

5.4 Kerbal Alarm Clock API

Provides RPCs to interact with the Kerbal Alarm Clock mod. Provides the following classes:

5.4.1 KerbalAlarmClock

public class KerbalAlarmClock

This service provides functionality to interact with the Kerbal Alarm Clock mod.

java.util.List<Alarm> getAlarms ()

A list of all the alarms.

Alarm AlarmWithName (String name)

Get the alarm with the given *name*, or null if no alarms have that name. If more than one alarm has the name, only returns one of them.

Parameters

• name (String) – Name of the alarm to search for.

java.util.List<Alarm> AlarmsWithType (AlarmType type)

Get a list of alarms of the specified type.

Parameters

• type (AlarmType) - Type of alarm to return.

Alarm CreateAlarm (AlarmType type, String name, double ut)
Create a new alarm and return it.

Parameters

- type (AlarmType) Type of the new alarm.
- name (String) Name of the new alarm.
- ut (double) Time at which the new alarm should trigger.

5.4.2 Alarm

public class Alarm

Represents an alarm. Obtained by calling Alarms, AlarmWithName or AlarmsWithType.

```
AlarmAction getAction()
```

void setAction (AlarmAction value)

The action that the alarm triggers.

double getMargin()

void setMargin (double value)

The number of seconds before the event that the alarm will fire.

```
double getTime()
void setTime (double value)
    The time at which the alarm will fire.
AlarmType getType()
    The type of the alarm.
String getID()
    The unique identifier for the alarm.
String getName()
void setName (String value)
    The short name of the alarm.
String getNotes()
void setNotes (String value)
    The long description of the alarm.
double getRemaining()
    The number of seconds until the alarm will fire.
boolean getRepeat ()
void setRepeat (boolean value)
    Whether the alarm will be repeated after it has fired.
double getRepeatPeriod()
void setRepeatPeriod (double value)
    The time delay to automatically create an alarm
    after it has fired.
SpaceCenter.Vessel getVessel ()
void setVessel (SpaceCenter.Vessel value)
    The vessel that the alarm is attached to.
SpaceCenter.CelestialBody getXferOriginBody ()
void setXferOriginBody (SpaceCenter.CelestialBody value)
    The celestial body the vessel is departing from.
SpaceCenter.CelestialBody getXferTargetBody ()
void setXferTargetBody (SpaceCenter.CelestialBody value)
    The celestial body the vessel is arriving at.
void Remove ()
    Removes the alarm.
```

214 Chapter 5. Java

5.4.3 AlarmType

public enum AlarmType

The type of an alarm.

public AlarmType Raw

An alarm for a specific date/time or a specific period in the future.

public AlarmType Maneuver

An alarm based on the next maneuver node on the current ships flight path. This node will be stored and can be restored when you come back to the ship.

public AlarmType ManeuverAuto

See AlarmType.Maneuver.

public AlarmType Apoapsis

An alarm for furthest part of the orbit from the planet.

public AlarmType Periapsis

An alarm for nearest part of the orbit from the planet.

public AlarmType AscendingNode

Ascending node for the targeted object, or equatorial ascending node.

public AlarmType DescendingNode

Descending node for the targeted object, or equatorial descending node.

public AlarmType Closest

An alarm based on the closest approach of this vessel to the targeted vessel, some number of orbits into the future.

public AlarmType Contract

An alarm based on the expiry or deadline of contracts in career modes.

public AlarmType ContractAuto

See AlarmType.Contract.

${\it public}\, {\it AlarmType}\,\, {\it Crew}$

An alarm that is attached to a crew member.

public AlarmType Distance

An alarm that is triggered when a selected target comes within a chosen distance.

public AlarmType EarthTime

An alarm based on the time in the "Earth" alternative Universe (aka the Real World).

public AlarmType LaunchRendevous

An alarm that fires as your landed craft passes under the orbit of your target.

public AlarmType SOIChange

An alarm manually based on when the next SOI

point is on the flight path or set to continually monitor the active flight path and add alarms as it detects SOI changes.

public AlarmType SOIChangeAuto See AlarmType.SOIChange.

public AlarmType Transfer

An alarm based on Interplanetary Transfer Phase Angles, i.e. when should I launch to planet X? Based on Kosmo Not's post and used in Olex's Calculator.

public AlarmType TransferModelled

See AlarmType.Transfer.

5.4.4 AlarmAction

public enum AlarmAction

The action performed by an alarm when it fires.

```
public AlarmAction DoNothing
```

Don't do anything at all...

$public \textit{AlarmAction} \ \textbf{DoNothingDeleteWhenPassed}$

Don't do anything, and delete the alarm.

public AlarmAction KillWarp

Drop out of time warp.

public AlarmAction KillWarpOnly

Drop out of time warp.

public AlarmAction MessageOnly

Display a message.

public AlarmAction PauseGame

Pause the game.

5.4.5 Example

The following example creates a new alarm for the active vessel. The alarm is set to trigger after 10 seconds have passed, and display a message.

```
import java.io.IOException;
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.KerbalAlarmClock;
import krpc.client.services.KerbalAlarmClock.Alarm;
import krpc.client.services.KerbalAlarmClock.AlarmAction;
import krpc.client.services.KerbalAlarmClock.AlarmType;
import krpc.client.services.SpaceCenter;

public class KerbalAlarmClockExample {
    public static void main(String[] args) throws IOException, RPCException {
        Connection connection = Connection.newInstance("Kerbal Alarm Clock Example", "10.0.2.2");
}
```

216 Chapter 5. Java

```
KerbalAlarmClock kac = KerbalAlarmClock.newInstance(connection);
Alarm alarm = kac.createAlarm(AlarmType.RAW, "My New Alarm", SpaceCenter.newInstance(connect.alarm.setNotes("10 seconds have now passed since the alarm was created.");
alarm.setAction(AlarmAction.MESSAGE_ONLY);
}
```

218 Chapter 5. Java

CHAPTER

SIX

LUA

6.1 Lua Client

The krpc module provides functionality to interact with a kRPC server from lua. It can be installed using luarocks.

6.1.1 Connecting to the Server

To connect to a server, use the *krpc.connect* () function. This returns a connection object through which you can interact with the server. For example to connect to a server running on the local machine:

```
local krpc = require 'krpc'
local conn = krpc.connect('Example')
print(conn.krpc:get_status().version)
```

This function also accepts arguments that specify what address and port numbers to connect to. For example:

```
local krpc = require 'krpc'
local conn = krpc.connect('Remote example', 'my.domain.name', 1000, 1001)
print(conn.krpc:get_status().version)
```

6.1.2 Interacting with the Server

Interaction with the server is performed via the client object (of type krpc.Client) returned when connecting to the server using krpc.connect().

Upon connecting, the client interrogates the server to find out what functionality it provides and dynamically adds all of the classes, methods, properties to the client object.

For example, all of the functionality provided by the SpaceCenter service is accessible via conn.space_center and the functionality provided by the InfernalRobotics service is accessible via conn.infernal_robotics.

Calling methods, getting or setting properties, etc. are mapped to remote procedure calls and passed to the server by the lua client.

6.1.3 Streaming Data from the Server

Warning: Streams are not yet supported by the Lua client.

6.1.4 Reference

```
krpc.connect([name=nil][, address='127.0.0.1'][, rpc_port=50000][, stream_port=50001])
```

This function creates a connection to a kRPC server. It returns a krpc.Client object, through which the server can be communicated with.

Parameters

- name (string) A descriptive name for the connection. This is passed to the server and appears, for example, in the client connection dialog on the in-game server window.
- address (string) The address of the server to connect to. Can either be a hostname or an IP address in dotted decimal notation. Defaults to '127.0.0.1'.
- rpc_port (int) The port number of the RPC Server. Defaults to 50000.
- **stream_port** (*int*) The port number of the Stream Server. Defaults to 50001.

class krpc.Client

This class provides the interface for communicating with the server. It is dynamically populated with all the functionality provided by the server. Instances of this class should be obtained by calling krpc.connect().

close()

Closes the connection to the server.

krpc

The built-in KRPC class, providing basic interactions with the server.

```
Return type krpc.KRPC
```

class krpc.KRPC

This class provides access to the basic server functionality provided by the KRPC service. An instance can be obtained by calling *krpc.Client.krpc*. Most of this functionality is used internally by the lua client and therefore does not need to be used directly from application code. The only exception that may be useful is:

```
get_status()
```

Gets a status message from the server containing information including the server's version string and performance statistics.

For example, the following prints out the version string for the server:

```
print('Server version = ' .. conn.krpc:get_status().version)
```

Or to get the rate at which the server is sending and receiving data over the network:

```
local status = conn.krpc:get_status()
print('Data in = ' .. (status.bytes_read_rate/1024) .. ' KB/s')
print('Data out = ' .. (status.bytes_written_rate/1024) .. ' KB/s')
```

6.2 SpaceCenter API

6.2.1 SpaceCenter

class SpaceCenter. SpaceCenter

Provides functionality to interact with Kerbal Space Program. This includes controlling the active vessel, managing its resources, planning maneuver nodes and auto-piloting.

active_vessel

The currently active vessel.

```
Attribute Can be read or written
```

Return type Vessel

vessels

A list of all the vessels in the game.

Attribute Read-only, cannot be set

Return type List of Vessel

bodies

A dictionary of all celestial bodies (planets, moons, etc.) in the game, keyed by the name of the body.

Attribute Read-only, cannot be set

Return type Map from string to CelestialBody

target_body

The currently targeted celestial body.

Attribute Can be read or written

Return type CelestialBody

target_vessel

The currently targeted vessel.

Attribute Can be read or written

Return type Vessel

target_docking_port

The currently targeted docking port.

Attribute Can be read or written

Return type DockingPort

static clear_target()

Clears the current target.

static launch_vessel_from_vab (name)

Launch a new vessel from the VAB onto the launchpad.

Parameters name (string) – Name of the vessel's craft file.

static launch_vessel_from_sph (name)

Launch a new vessel from the SPH onto the runway.

Parameters name (string) – Name of the vessel's craft file.

ut

The current universal time in seconds.

Attribute Read-only, cannot be set

Return type number

g

The value of the gravitational constant G in $N(m/kg)^2$.

Attribute Read-only, cannot be set

Return type number

warp mode

The current time warp mode. Returns <code>WarpMode.none</code> if time warp is not active, <code>WarpMode.rails</code> if regular "on-rails" time warp is active, or <code>WarpMode.physics</code> if physical time warp is active.

Attribute Read-only, cannot be set

Return type WarpMode

warp_rate

The current warp rate. This is the rate at which time is passing for either on-rails or physical time warp. For example, a value of 10 means time is passing 10x faster than normal. Returns 1 if time warp is not active.

Attribute Read-only, cannot be set

Return type number

warp_factor

The current warp factor. This is the index of the rate at which time is passing for either regular "on-rails" or physical time warp. Returns 0 if time warp is not active. When in on-rails time warp, this is equal to rails_warp_factor, and in physics time warp, this is equal to physics_warp_factor.

Attribute Read-only, cannot be set

Return type number

rails_warp_factor

The time warp rate, using regular "on-rails" time warp. A value between 0 and 7 inclusive. 0 means no time warp. Returns 0 if physical time warp is active. If requested time warp factor cannot be set, it will be set to the next lowest possible value. For example, if the vessel is too close to a planet. See the KSP wiki for details.

Attribute Can be read or written

Return type number

physics_warp_factor

The physical time warp rate. A value between 0 and 3 inclusive. 0 means no time warp. Returns 0 if regular "on-rails" time warp is active.

Attribute Can be read or written

Return type number

static can_rails_warp_at ([factor = 1])

Returns True if regular "on-rails" time warp can be used, at the specified warp *factor*. The maximum time warp rate is limited by various things, including how close the active vessel is to a planet. See the KSP wiki for details.

Parameters factor (number) – The warp factor to check.

Return type boolean

maximum_rails_warp_factor

The current maximum regular "on-rails" warp factor that can be set. A value between 0 and 7 inclusive. See the KSP wiki for details.

Attribute Read-only, cannot be set

Return type number

```
static warp_to(ut[, max\_rails\_rate = 100000.0][, max\_physics\_rate = 2.0])
```

Uses time acceleration to warp forward to a time in the future, specified by universal time *ut*. This call blocks until the desired time is reached. Uses regular "on-rails" or physical time warp as appropriate. For

example, physical time warp is used when the active vessel is traveling through an atmosphere. When using regular "on-rails" time warp, the warp rate is limited by max_rails_rate, and when using physical time warp, the warp rate is limited by max_physics_rate.

Parameters

- ut (number) The universal time to warp to, in seconds.
- max_rails_rate (number) The maximum warp rate in regular "on-rails" time warp.
- max_physics_rate (number) The maximum warp rate in physical time warp.

Returns When the time warp is complete.

static transform_position (position, from, to)

Converts a position vector from one reference frame to another.

Parameters

- **position** (*Tuple*) Position vector in reference frame *from*.
- **from** (ReferenceFrame) The reference frame that the position vector is in.
- to (ReferenceFrame) The reference frame to covert the position vector to.

Returns The corresponding position vector in reference frame *to*.

Return type Tuple of (number, number, number)

static transform_direction (direction, from, to)

Converts a direction vector from one reference frame to another.

Parameters

- **direction** (*Tuple*) Direction vector in reference frame *from*.
- **from** (ReferenceFrame) The reference frame that the direction vector is in.
- to (ReferenceFrame) The reference frame to covert the direction vector to.

Returns The corresponding direction vector in reference frame to.

Return type Tuple of (number, number, number)

static transform_rotation (rotation, from, to)

Converts a rotation from one reference frame to another.

Parameters

- rotation (*Tuple*) Rotation in reference frame *from*.
- **from** (ReferenceFrame) The reference frame that the rotation is in.
- to (ReferenceFrame) The corresponding rotation in reference frame to.

Returns The corresponding rotation in reference frame *to*.

Return type Tuple of (number, number, number, number)

static transform_velocity (position, velocity, from, to)

Converts a velocity vector (acting at the specified position vector) from one reference frame to another. The position vector is required to take the relative angular velocity of the reference frames into account.

Parameters

- **position** (*Tuple*) Position vector in reference frame *from*.
- **velocity** (*Tuple*) Velocity vector in reference frame *from*.

- **from** (ReferenceFrame) The reference frame that the position and velocity vectors are in.
- to (ReferenceFrame) The reference frame to covert the velocity vector to.

Returns The corresponding velocity in reference frame *to*.

Return type Tuple of (number, number, number)

far_available

Whether Ferram Aerospace Research is installed.

Attribute Read-only, cannot be set

Return type boolean

remote_tech_available

Whether RemoteTech is installed.

Attribute Read-only, cannot be set

Return type boolean

static draw_direction (direction, reference_frame, color [, length = 10.0])

Draw a direction vector on the active vessel.

Parameters

- **direction** (*Tuple*) Direction to draw the line in.
- reference frame (ReferenceFrame) Reference frame that the direction is in.
- **color** (*Tuple*) The color to use for the line, as an RGB color.
- **length** (*number*) The length of the line. Defaults to 10.

static draw_line (start, end, reference_frame, color)

Draw a line.

Parameters

- **start** (*Tuple*) Position of the start of the line.
- end (Tuple) Position of the end of the line.
- reference_frame (ReferenceFrame) Reference frame that the position are in.
- color (Tuple) The color to use for the line, as an RGB color.

static clear_drawing()

Remove all directions and lines currently being drawn.

class SpaceCenter.WarpMode

Returned by warp_mode

rails

Time warp is active, and in regular "on-rails" mode.

physics

Time warp is active, and in physical time warp mode.

none

Time warp is not active.

6.2.2 Vessel

class SpaceCenter. Vessel

These objects are used to interact with vessels in KSP. This includes getting orbital and flight data, manipulating control inputs and managing resources.

name

The name of the vessel.

Attribute Can be read or written

Return type string

type

The type of the vessel.

Attribute Can be read or written

Return type Vessel Type

situation

The situation the vessel is in.

Attribute Read-only, cannot be set

Return type VesselSituation

met

The mission elapsed time in seconds.

Attribute Read-only, cannot be set

Return type number

flight ([reference_frame = None])

Returns a Flight object that can be used to get flight telemetry for the vessel, in the specified reference frame.

Parameters reference_frame (ReferenceFrame) - Reference frame. Defaults to the vessel's surface reference frame (Vessel.surface_reference_frame).

Return type Flight

target

The target vessel. nil if there is no target. When setting the target, the target cannot be the current vessel.

Attribute Can be read or written

Return type Vessel

orbit

The current orbit of the vessel.

Attribute Read-only, cannot be set

Return type Orbit

control

Returns a *Control* object that can be used to manipulate the vessel's control inputs. For example, its pitch/yaw/roll controls, RCS and thrust.

Attribute Read-only, cannot be set

Return type Control

auto_pilot

An AutoPilot object, that can be used to perform simple auto-piloting of the vessel.

Attribute Read-only, cannot be set

Return type AutoPilot

resources

A Resources object, that can used to get information about resources stored in the vessel.

Attribute Read-only, cannot be set

Return type Resources

resources_in_decouple_stage(stage[, cumulative = True])

Returns a Resources object, that can used to get information about resources stored in a given stage.

Parameters

- **stage** (number) Get resources for parts that are decoupled in this stage.
- **cumulative** (boolean) When False, returns the resources for parts decoupled in just the given stage. When True returns the resources decoupled in the given stage and all subsequent stages combined.

Return type Resources

Note: For details on stage numbering, see the discussion on *Staging*.

parts

A Parts object, that can used to interact with the parts that make up this vessel.

Attribute Read-only, cannot be set

Return type Parts

comms

A Comms object, that can used to interact with RemoteTech for this vessel.

Attribute Read-only, cannot be set

Return type Comms

Note: Requires RemoteTech to be installed.

mass

The total mass of the vessel, including resources, in kg.

Attribute Read-only, cannot be set

Return type number

dry_mass

The total mass of the vessel, excluding resources, in kg.

Attribute Read-only, cannot be set

Return type number

thrust

The total thrust currently being produced by the vessel's engines, in Newtons. This is computed by summing <code>Engine.thrust</code> for every engine in the vessel.

Attribute Read-only, cannot be set

Return type number

available_thrust

Gets the total available thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine.available_thrust</code> for every active engine in the vessel.

Attribute Read-only, cannot be set

Return type number

max_thrust

The total maximum thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine.max_thrust</code> for every active engine.

Attribute Read-only, cannot be set

Return type number

max_vacuum_thrust

The total maximum thrust that can be produced by the vessel's active engines when the vessel is in a vacuum, in Newtons. This is computed by summing <code>Engine.max_vacuum_thrust</code> for every active engine.

Attribute Read-only, cannot be set

Return type number

specific_impulse

The combined specific impulse of all active engines, in seconds. This is computed using the formula described here.

Attribute Read-only, cannot be set

Return type number

vacuum_specific_impulse

The combined vacuum specific impulse of all active engines, in seconds. This is computed using the formula described here.

Attribute Read-only, cannot be set

Return type number

kerbin_sea_level_specific_impulse

The combined specific impulse of all active engines at sea level on Kerbin, in seconds. This is computed using the formula described here.

Attribute Read-only, cannot be set

Return type number

reference_frame

The reference frame that is fixed relative to the vessel, and orientated with the vessel.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel.
- •The x-axis points out to the right of the vessel.
- •The y-axis points in the forward direction of the vessel.
- •The z-axis points out of the bottom off the vessel.

Attribute Read-only, cannot be set Return type ReferenceFrame

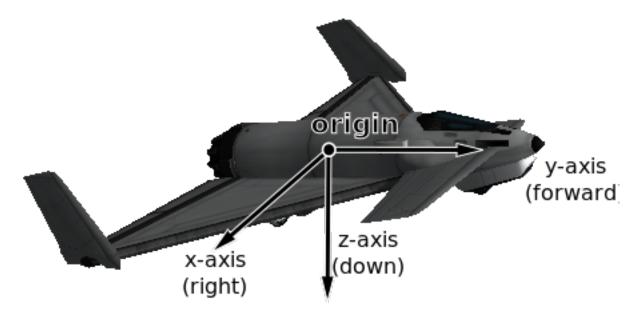


Fig. 6.1: Vessel reference frame origin and axes for the Aeris 3A aircraft

orbital reference frame

The reference frame that is fixed relative to the vessel, and orientated with the vessels orbital prograde/normal/radial directions.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Attribute Read-only, cannot be set Return type ReferenceFrame

Note: Be careful not to confuse this with 'orbit' mode on the navball.

surface_reference_frame

The reference frame that is fixed relative to the vessel, and orientated with the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the north and up directions on the surface of the body.
- •The x-axis points in the zenith direction (upwards, normal to the body being orbited, from the center of the body towards the center of mass of the vessel).
- •The y-axis points northwards towards the astronomical horizon (north, and tangential to the surface of the body the direction in which a compass would point when on the surface).

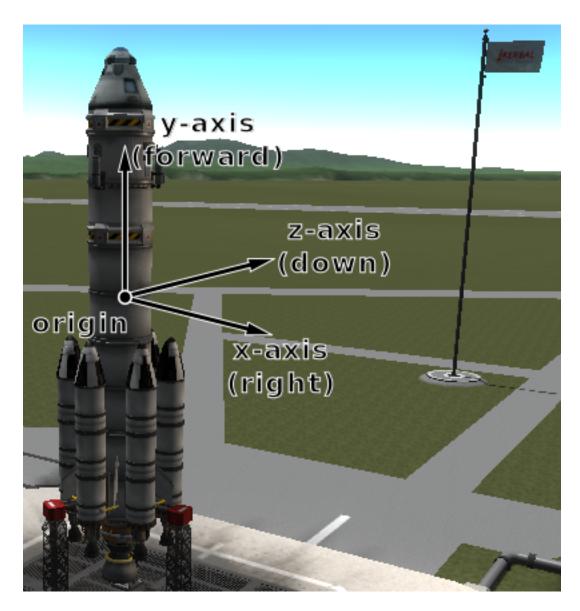


Fig. 6.2: Vessel reference frame origin and axes for the Kerbal-X rocket

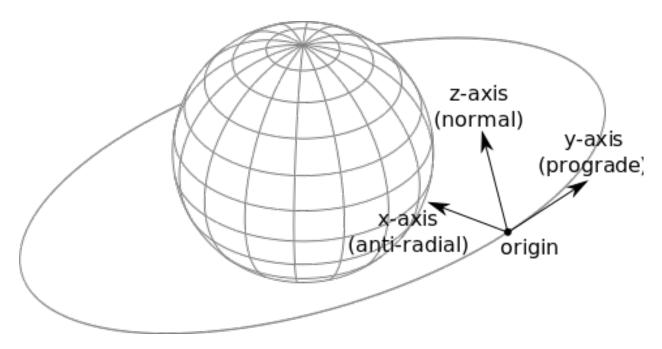


Fig. 6.3: Vessel orbital reference frame origin and axes

•The z-axis points eastwards towards the astronomical horizon (east, and tangential to the surface of the body – east on a compass when on the surface).

Attribute Read-only, cannot be set Return type ReferenceFrame

Note: Be careful not to confuse this with 'surface' mode on the navball.

surface_velocity_reference_frame

The reference frame that is fixed relative to the vessel, and orientated with the velocity vector of the vessel relative to the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel's velocity vector.
- •The y-axis points in the direction of the vessel's velocity vector, relative to the surface of the body being orbited.
- •The z-axis is in the plane of the astronomical horizon.
- •The x-axis is orthogonal to the other two axes.

Attribute Read-only, cannot be set Return type ReferenceFrame

position (reference_frame)

Returns the position vector of the center of mass of the vessel in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

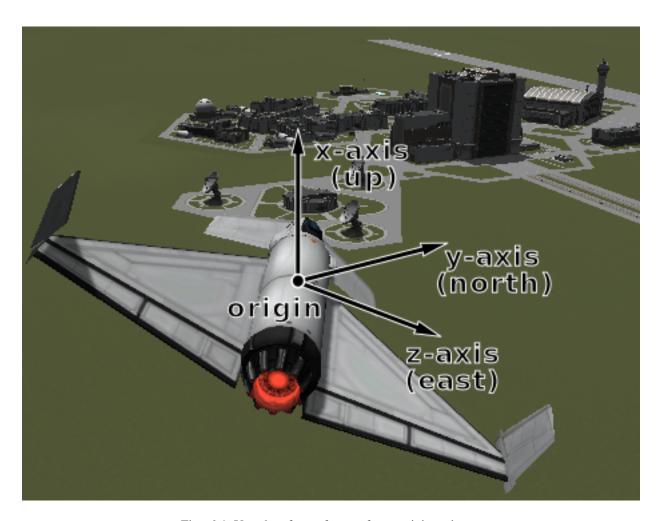


Fig. 6.4: Vessel surface reference frame origin and axes

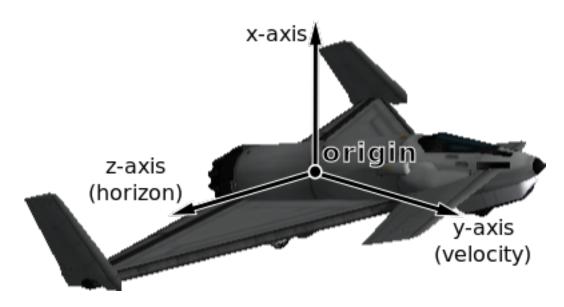


Fig. 6.5: Vessel surface velocity reference frame origin and axes

```
velocity (reference_frame)
          Returns the velocity vector of the center of mass of the vessel in the given reference frame.
              Parameters reference_frame (ReferenceFrame) -
              Return type Tuple of (number, number, number)
     rotation(reference frame)
          Returns the rotation of the center of mass of the vessel in the given reference frame.
              Parameters reference_frame (ReferenceFrame) -
              Return type Tuple of (number, number, number, number)
     direction (reference_frame)
          Returns the direction in which the vessel is pointing, as a unit vector, in the given reference frame.
              Parameters reference_frame (ReferenceFrame) -
              Return type Tuple of (number, number, number)
     angular_velocity (reference_frame)
          Returns the angular velocity of the vessel in the given reference frame. The magnitude of the returned
          vector is the rotational speed in radians per second, and the direction of the vector indicates the axis of
          rotation (using the right hand rule).
              Parameters reference_frame (ReferenceFrame) -
              Return type Tuple of (number, number, number)
class SpaceCenter.VesselType
     See Vessel.type.
     ship
          Ship.
     station
          Station.
     lander
          Lander.
     probe
          Probe.
     rover
          Rover.
     hase
          Base.
     debris
          Debris.
class SpaceCenter.VesselSituation
     See Vessel.situation.
     docked
          Vessel is docked to another.
     escaping
          Escaping.
     flying
          Vessel is flying through an atmosphere.
```

landed

Vessel is landed on the surface of a body.

orbiting

Vessel is orbiting a body.

pre_launch

Vessel is awaiting launch.

splashed

Vessel has splashed down in an ocean.

sub orbital

Vessel is on a sub-orbital trajectory.

6.2.3 CelestialBody

class SpaceCenter.CelestialBody

Represents a celestial body (such as a planet or moon).

name

The name of the body.

Attribute Read-only, cannot be set

Return type string

satellites

A list of celestial bodies that are in orbit around this celestial body.

Attribute Read-only, cannot be set

Return type List of CelestialBody

orbit

The orbit of the body.

Attribute Read-only, cannot be set

Return type Orbit

mass

The mass of the body, in kilograms.

Attribute Read-only, cannot be set

Return type number

gravitational_parameter

The standard gravitational parameter of the body in m^3s^{-2} .

Attribute Read-only, cannot be set

Return type number

surface_gravity

The acceleration due to gravity at sea level (mean altitude) on the body, in m/s^2 .

Attribute Read-only, cannot be set

Return type number

rotational_period

The sidereal rotational period of the body, in seconds.

Attribute Read-only, cannot be set

Return type number

rotational_speed

The rotational speed of the body, in radians per second.

Attribute Read-only, cannot be set

Return type number

equatorial_radius

The equatorial radius of the body, in meters.

Attribute Read-only, cannot be set

Return type number

surface_height (latitude, longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water this is equal to 0.

Parameters

- latitude (number) Latitude in degrees
- longitude (number) Longitude in degrees

Return type number

bedrock_height (latitude, longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water, this is the height of the sea-bed and is therefore a negative value.

Parameters

- latitude (number) Latitude in degrees
- longitude (number) Longitude in degrees

Return type number

msl_position (latitude, longitude, reference_frame)

The position at mean sea level at the given latitude and longitude, in the given reference frame.

Parameters

- latitude (number) Latitude in degrees
- longitude (number) Longitude in degrees
- reference_frame (ReferenceFrame) Reference frame for the returned position vector

Return type Tuple of (number, number, number)

$\verb|surface_position| (\textit{latitude}, \textit{longitude}, \textit{reference_frame})|$

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position of the surface of the water.

Parameters

- latitude (number) Latitude in degrees
- longitude (number) Longitude in degrees
- reference_frame (ReferenceFrame) Reference frame for the returned position vector

Return type Tuple of (number, number, number)

bedrock_position (latitude, longitude, reference_frame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position at the bottom of the sea-bed.

Parameters

- latitude (number) Latitude in degrees
- longitude (number) Longitude in degrees
- reference_frame (ReferenceFrame) Reference frame for the returned position vector

Return type Tuple of (number, number, number)

sphere_of_influence

The radius of the sphere of influence of the body, in meters.

Attribute Read-only, cannot be set

Return type number

has_atmosphere

True if the body has an atmosphere.

Attribute Read-only, cannot be set

Return type boolean

atmosphere depth

The depth of the atmosphere, in meters.

Attribute Read-only, cannot be set

Return type number

has_atmospheric_oxygen

True if there is oxygen in the atmosphere, required for air-breathing engines.

Attribute Read-only, cannot be set

Return type boolean

reference_frame

The reference frame that is fixed relative to the celestial body.

- •The origin is at the center of the body.
- •The axes rotate with the body.
- •The x-axis points from the center of the body towards the intersection of the prime meridian and equator (the position at 0° longitude, 0° latitude).
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points from the center of the body towards the equator at 90°E longitude.

Attribute Read-only, cannot be set

Return type ReferenceFrame

non_rotating_reference_frame

The reference frame that is fixed relative to this celestial body, and orientated in a fixed direction (it does not rotate with the body).

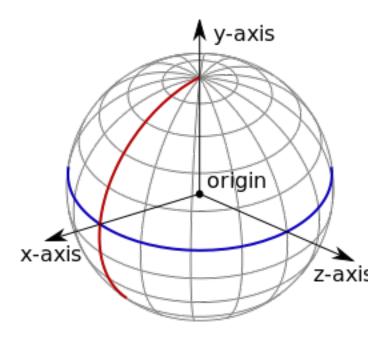


Fig. 6.6: Celestial body reference frame origin and axes. The equator is shown in blue, and the prime meridian in red.

- •The origin is at the center of the body.
- •The axes do not rotate.
- •The x-axis points in an arbitrary direction through the equator.
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points in an arbitrary direction through the equator.

Attribute Read-only, cannot be set Return type ReferenceFrame

orbital_reference_frame

Gets the reference frame that is fixed relative to this celestial body, but orientated with the body's orbital prograde/normal/radial directions.

- •The origin is at the center of the body.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Attribute Read-only, cannot be set Return type ReferenceFrame

position (reference_frame)

Returns the position vector of the center of the body in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

velocity (reference_frame)

Returns the velocity vector of the body in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

rotation (reference_frame)

Returns the rotation of the body in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number, number)

direction (reference_frame)

Returns the direction in which the north pole of the celestial body is pointing, as a unit vector, in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

angular_velocity (reference_frame)

Returns the angular velocity of the body in the specified reference frame. The magnitude of the vector is the rotational speed of the body, in radians per second, and the direction of the vector indicates the axis of rotation, using the right-hand rule.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

6.2.4 Flight

class SpaceCenter.Flight

Used to get flight telemetry for a vessel, by calling Vessel.flight(). All of the information returned by this class is given in the reference frame passed to that method.

Note: To get orbital information, such as the apoapsis or inclination, see Orbit.

q force

The current G force acting on the vessel in m/s^2 .

Attribute Read-only, cannot be set

Return type number

mean altitude

The altitude above sea level, in meters.

Attribute Read-only, cannot be set

Return type number

surface_altitude

The altitude above the surface of the body or sea level, whichever is closer, in meters.

Attribute Read-only, cannot be set

Return type number

bedrock altitude

The altitude above the surface of the body, in meters. When over water, this is the altitude above the sea floor.

Attribute Read-only, cannot be set

Return type number

elevation

The elevation of the terrain under the vessel, in meters. This is the height of the terrain above sea level, and is negative when the vessel is over the sea.

Attribute Read-only, cannot be set

Return type number

latitude

The latitude of the vessel for the body being orbited, in degrees.

Attribute Read-only, cannot be set

Return type number

longitude

The longitude of the vessel for the body being orbited, in degrees.

Attribute Read-only, cannot be set

Return type number

velocity

The velocity vector of the vessel. The magnitude of the vector is the speed of the vessel in meters per second. The direction of the vector is the direction of the vessels motion.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

speed

The speed of the vessel in meters per second.

Attribute Read-only, cannot be set

Return type number

horizontal_speed

The horizontal speed of the vessel in meters per second.

Attribute Read-only, cannot be set

Return type number

vertical_speed

The vertical speed of the vessel in meters per second.

Attribute Read-only, cannot be set

Return type number

center_of_mass

The position of the center of mass of the vessel.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

rotation

The rotation of the vessel.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number, number)

direction

The direction vector that the vessel is pointing in.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

pitch

The pitch angle of the vessel relative to the horizon, in degrees. A value between -90° and +90°.

Attribute Read-only, cannot be set

Return type number

heading

The heading angle of the vessel relative to north, in degrees. A value between 0° and 360°.

Attribute Read-only, cannot be set

Return type number

roll

The roll angle of the vessel relative to the horizon, in degrees. A value between -180 $^{\circ}$ and +180 $^{\circ}$.

Attribute Read-only, cannot be set

Return type number

prograde

The unit direction vector pointing in the prograde direction.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

retrograde

The unit direction vector pointing in the retrograde direction.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

normal

The unit direction vector pointing in the normal direction.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

anti_normal

The unit direction vector pointing in the anti-normal direction.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

radial

The unit direction vector pointing in the radial direction.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

anti radial

The unit direction vector pointing in the anti-radial direction.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

atmosphere_density

The current density of the atmosphere around the vessel, in kg/m^3 .

Attribute Read-only, cannot be set

Return type number

dynamic_pressure

The dynamic pressure acting on the vessel, in Pascals. This is a measure of the strength of the aerodynamic forces. It is equal to $\frac{1}{2}$ air density velocity². It is commonly denoted as Q.

Attribute Read-only, cannot be set

Return type number

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

static_pressure

The static atmospheric pressure acting on the vessel, in Pascals.

Attribute Read-only, cannot be set

Return type number

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

aerodynamic_force

The total aerodynamic forces acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

lift

The aerodynamic lift currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

drag

The aerodynamic drag currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Attribute Read-only, cannot be set

Return type Tuple of (number, number, number)

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

speed_of_sound

The speed of sound, in the atmosphere around the vessel, in m/s.

Attribute Read-only, cannot be set

Return type number

Note: Not available when Ferram Aerospace Research is installed.

mach

The speed of the vessel, in multiples of the speed of sound.

Attribute Read-only, cannot be set

Return type number

Note: Not available when Ferram Aerospace Research is installed.

equivalent_air_speed

The equivalent air speed of the vessel, in m/s.

Attribute Read-only, cannot be set

Return type number

Note: Not available when Ferram Aerospace Research is installed.

terminal_velocity

An estimate of the current terminal velocity of the vessel, in m/s. This is the speed at which the drag forces cancel out the force of gravity.

Attribute Read-only, cannot be set

Return type number

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

angle_of_attack

Gets the pitch angle between the orientation of the vessel and its velocity vector, in degrees.

Attribute Read-only, cannot be set

Return type number

sideslip angle

Gets the yaw angle between the orientation of the vessel and its velocity vector, in degrees.

Attribute Read-only, cannot be set

Return type number

total_air_temperature

The total air temperature of the atmosphere around the vessel, in Kelvin. This temperature includes the Flight.static_air_temperature and the vessel's kinetic energy.

Attribute Read-only, cannot be set

Return type number

static_air_temperature

The static (ambient) temperature of the atmosphere around the vessel, in Kelvin.

Attribute Read-only, cannot be set

Return type number

stall fraction

Gets the current amount of stall, between 0 and 1. A value greater than 0.005 indicates a minor stall and a value greater than 0.5 indicates a large-scale stall.

Attribute Read-only, cannot be set

Return type number

Note: Requires Ferram Aerospace Research.

drag_coefficient

Gets the coefficient of drag. This is the amount of drag produced by the vessel. It depends on air speed, air density and wing area.

Attribute Read-only, cannot be set

Return type number

Note: Requires Ferram Aerospace Research.

lift coefficient

Gets the coefficient of lift. This is the amount of lift produced by the vessel, and depends on air speed, air density and wing area.

Attribute Read-only, cannot be set

Return type number

Note: Requires Ferram Aerospace Research.

ballistic coefficient

Gets the ballistic coefficient.

Attribute Read-only, cannot be set

Return type number

Note: Requires Ferram Aerospace Research.

thrust_specific_fuel_consumption

Gets the thrust specific fuel consumption for the jet engines on the vessel. This is a measure of the efficiency of the engines, with a lower value indicating a more efficient vessel. This value is the number of Newtons of fuel that are burned, per hour, to product one newton of thrust.

Attribute Read-only, cannot be set

Return type number

Note: Requires Ferram Aerospace Research.

6.2.5 Orbit

class SpaceCenter.Orbit

Describes an orbit. For example, the orbit of a vessel, obtained by calling <code>Vessel.orbit</code>, or a celestial body, obtained by calling <code>CelestialBody.orbit</code>.

body

The celestial body (e.g. planet or moon) around which the object is orbiting.

Attribute Read-only, cannot be set

Return type CelestialBody

apoapsis

Gets the apoapsis of the orbit, in meters, from the center of mass of the body being orbited.

Attribute Read-only, cannot be set

Return type number

Note: For the apoapsis altitude reported on the in-game map view, use <code>Orbit.apoapsis_altitude</code>.

periapsis

The periapsis of the orbit, in meters, from the center of mass of the body being orbited.

Attribute Read-only, cannot be set

Return type number

Note: For the periapsis altitude reported on the in-game map view, use <code>Orbit.periapsis_altitude</code>.

apoapsis_altitude

The apoapsis of the orbit, in meters, above the sea level of the body being orbited.

Attribute Read-only, cannot be set

Return type number

Note: This is equal to Orbit.apoapsis minus the equatorial radius of the body.

periapsis_altitude

The periapsis of the orbit, in meters, above the sea level of the body being orbited.

Attribute Read-only, cannot be set

Return type number

Note: This is equal to Orbit.periapsis minus the equatorial radius of the body.

semi_major_axis

The semi-major axis of the orbit, in meters.

Attribute Read-only, cannot be set

Return type number

semi_minor_axis

The semi-minor axis of the orbit, in meters.

Attribute Read-only, cannot be set

Return type number

radius

The current radius of the orbit, in meters. This is the distance between the center of mass of the object in orbit, and the center of mass of the body around which it is orbiting.

Attribute Read-only, cannot be set

Return type number

Note: This value will change over time if the orbit is elliptical.

speed

The current orbital speed of the object in meters per second.

Attribute Read-only, cannot be set

Return type number

Note: This value will change over time if the orbit is elliptical.

period

The orbital period, in seconds.

Attribute Read-only, cannot be set

Return type number

time_to_apoapsis

The time until the object reaches apoapsis, in seconds.

Attribute Read-only, cannot be set

Return type number

time_to_periapsis

The time until the object reaches periapsis, in seconds.

Attribute Read-only, cannot be set

Return type number

eccentricity

The eccentricity of the orbit.

Attribute Read-only, cannot be set

Return type number

inclination

The inclination of the orbit, in radians.

Attribute Read-only, cannot be set

Return type number

longitude_of_ascending_node

The longitude of the ascending node, in radians.

Attribute Read-only, cannot be set

Return type number

argument_of_periapsis

The argument of periapsis, in radians.

Attribute Read-only, cannot be set

Return type number

mean_anomaly_at_epoch

The mean anomaly at epoch.

Attribute Read-only, cannot be set

Return type number

epoch

The time since the epoch (the point at which the mean anomaly at epoch was measured, in seconds.

Attribute Read-only, cannot be set

Return type number

mean_anomaly

The mean anomaly.

Attribute Read-only, cannot be set

Return type number

eccentric_anomaly

The eccentric anomaly.

Attribute Read-only, cannot be set

Return type number

static reference_plane_normal (reference_frame)

The unit direction vector that is normal to the orbits reference plane, in the given reference frame. The reference plane is the plane from which the orbits inclination is measured.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

static reference_plane_direction (reference_frame)

The unit direction vector from which the orbits longitude of ascending node is measured, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

time_to_soi_change

The time until the object changes sphere of influence, in seconds. Returns NaN if the object is not going to change sphere of influence.

Attribute Read-only, cannot be set

Return type number

next_orbit

If the object is going to change sphere of influence in the future, returns the new orbit after the change. Otherwise returns nil.

Attribute Read-only, cannot be set

Return type Orbit

6.2.6 Control

class SpaceCenter.Control

Used to manipulate the controls of a vessel. This includes adjusting the throttle, enabling/disabling systems such as SAS and RCS, or altering the direction in which the vessel is pointing.

Note: Control inputs (such as pitch, yaw and roll) are zeroed when all clients that have set one or more of these inputs are no longer connected.

sas

The state of SAS.

Attribute Can be read or written

Return type boolean

Note: Equivalent to AutoPilot.sas

sas_mode

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Attribute Can be read or written

Return type SASMode

Note: Equivalent to AutoPilot.sas_mode

speed_mode

The current *SpeedMode* of the navball. This is the mode displayed next to the speed at the top of the navball.

Attribute Can be read or written

Return type SpeedMode

rcs

The state of RCS.

Attribute Can be read or written

Return type boolean

gear

The state of the landing gear/legs.

Attribute Can be read or written

Return type boolean

lights

The state of the lights.

Attribute Can be read or written

Return type boolean

brakes

The state of the wheel brakes.

Attribute Can be read or written

Return type boolean

abort

The state of the abort action group.

Attribute Can be read or written

Return type boolean

throttle

The state of the throttle. A value between 0 and 1.

Attribute Can be read or written

Return type number

pitch

The state of the pitch control. A value between -1 and 1. Equivalent to the w and s keys.

Attribute Can be read or written

Return type number

yaw

The state of the yaw control. A value between -1 and 1. Equivalent to the a and d keys.

Attribute Can be read or written

Return type number

roll

The state of the roll control. A value between -1 and 1. Equivalent to the q and e keys.

Attribute Can be read or written

Return type number

forward

The state of the forward translational control. A value between -1 and 1. Equivalent to the h and n keys.

Attribute Can be read or written

Return type number

up

The state of the up translational control. A value between -1 and 1. Equivalent to the i and k keys.

Attribute Can be read or written

Return type number

right

The state of the right translational control. A value between -1 and 1. Equivalent to the j and l keys.

Attribute Can be read or written

Return type number

wheel_throttle

The state of the wheel throttle. A value between -1 and 1. A value of 1 rotates the wheels forwards, a value of -1 rotates the wheels backwards.

Attribute Can be read or written

Return type number

wheel_steering

The state of the wheel steering. A value between -1 and 1. A value of 1 steers to the left, and a value of -1 steers to the right.

Attribute Can be read or written

Return type number

current_stage

The current stage of the vessel. Corresponds to the stage number in the in-game UI.

Attribute Read-only, cannot be set

Return type number

activate_next_stage()

Activates the next stage. Equivalent to pressing the space bar in-game.

Returns A list of vessel objects that are jettisoned from the active vessel.

Return type List of Vessel

${\tt get_action_group}\,(\mathit{group})$

Returns True if the given action group is enabled.

Parameters group (number) – A number between 0 and 9 inclusive.

Return type boolean

set_action_group (group, state)

Sets the state of the given action group (a value between 0 and 9 inclusive).

Parameters

- group (number) A number between 0 and 9 inclusive.
- state (boolean) -

toggle_action_group(group)

Toggles the state of the given action group.

Parameters group (number) – A number between 0 and 9 inclusive.

$add_node(ut[,prograde = 0.0][,normal = 0.0][,radial = 0.0])$

Creates a maneuver node at the given universal time, and returns a *Node* object that can be used to modify it. Optionally sets the magnitude of the delta-v for the maneuver node in the prograde, normal and radial directions.

Parameters

- **ut** (number) Universal time of the maneuver node.
- **prograde** (number) Delta-v in the prograde direction.
- normal (number) Delta-v in the normal direction.
- radial (number) Delta-v in the radial direction.

Return type Node

nodes

Returns a list of all existing maneuver nodes, ordered by time from first to last.

Attribute Read-only, cannot be set

Return type List of Node

remove_nodes()

Remove all maneuver nodes.

class SpaceCenter.SASMode

The behavior of the SAS auto-pilot. See AutoPilot.sas_mode.

stability_assist

Stability assist mode. Dampen out any rotation.

maneuver

Point in the burn direction of the next maneuver node.

prograde

Point in the prograde direction.

retrograde

Point in the retrograde direction.

normal

Point in the orbit normal direction.

anti_normal

Point in the orbit anti-normal direction.

radial

Point in the orbit radial direction.

anti_radial

Point in the orbit anti-radial direction.

target

Point in the direction of the current target.

anti_target

Point away from the current target.

class SpaceCenter.SpeedMode

See Control.speed_mode.

orbit

Speed is relative to the vessel's orbit.

surface

Speed is relative to the surface of the body being orbited.

target

Speed is relative to the current target.

6.2.7 Parts

The following classes allow interaction with a vessels individual parts.

- Parts
- Part
- Module
- Specific Types of Part
 - Decoupler
 - Docking Port
 - Engine
 - Landing Gear
 - Landing Leg
 - Launch Clamp
 - Light
 - Parachute
 - Radiator
 - Resource Converter
 - Resource Harvester
 - Reaction Wheel
 - Sensor
 - Solar Panel
- Trees of Parts
 - Traversing the Tree
 - Attachment Modes
- Fuel Lines
- Staging

Parts

class SpaceCenter.Parts

Instances of this class are used to interact with the parts of a vessel. An instance can be obtained by calling *Vessel.parts*.

all

A list of all of the vessels parts.

Attribute Read-only, cannot be set

Return type List of Part

root

The vessels root part.

Attribute Read-only, cannot be set

Return type Part

```
Note: See the discussion on Trees of Parts.
controlling
     The part from which the vessel is controlled.
         Attribute Can be read or written
         Return type Part
with_name (name)
     A list of parts whose Part.name is name.
         Parameters name (string) -
         Return type List of Part
with_title(title)
     A list of all parts whose Part.title is title.
         Parameters title (string) -
         Return type List of Part
with_module (module_name)
     A list of all parts that contain a Module whose Module.name is module_name.
         Parameters module name (string) -
         Return type List of Part
in_stage(stage)
     A list of all parts that are activated in the given stage.
         Parameters stage (number) -
         Return type List of Part
     Note: See the discussion on Staging.
in_decouple_stage(stage)
     A list of all parts that are decoupled in the given stage.
         Parameters stage (number) -
         Return type List of Part
     Note: See the discussion on Staging.
modules_with_name (module_name)
     A list of modules (combined across all parts in the vessel) whose Module.name is module_name.
         Parameters module_name (string) -
         Return type List of Module
decouplers
     A list of all decouplers in the vessel.
         Attribute Read-only, cannot be set
         Return type List of Decoupler
```

docking_ports

A list of all docking ports in the vessel.

Attribute Read-only, cannot be set

Return type List of DockingPort

docking_port_with_name (name)

The first docking port in the vessel with the given port name, as returned by <code>DockingPort.name</code>. Returns nil if there are no such docking ports.

Parameters name (string) -

Return type DockingPort

engines

A list of all engines in the vessel.

Attribute Read-only, cannot be set

Return type List of Engine

landing_gear

A list of all landing gear attached to the vessel.

Attribute Read-only, cannot be set

Return type List of LandingGear

landing_legs

A list of all landing legs attached to the vessel.

Attribute Read-only, cannot be set

Return type List of LandingLeg

launch_clamps

A list of all launch clamps attached to the vessel.

Attribute Read-only, cannot be set

Return type List of LaunchClamp

lights

A list of all lights in the vessel.

Attribute Read-only, cannot be set

Return type List of Light

parachutes

A list of all parachutes in the vessel.

Attribute Read-only, cannot be set

Return type List of Parachute

radiators

A list of all radiators in the vessel.

Attribute Read-only, cannot be set

Return type List of Radiator

resource_converters

A list of all resource converters in the vessel.

Attribute Read-only, cannot be set

Return type List of ResourceConverter

resource_harvesters

A list of all resource harvesters in the vessel.

Attribute Read-only, cannot be set

Return type List of ResourceHarvester

reaction wheels

A list of all reaction wheels in the vessel.

Attribute Read-only, cannot be set

Return type List of ReactionWheel

sensors

A list of all sensors in the vessel.

Attribute Read-only, cannot be set

Return type List of Sensor

solar_panels

A list of all solar panels in the vessel.

Attribute Read-only, cannot be set

Return type List of SolarPanel

Part

class SpaceCenter.Part

Instances of this class represents a part. A vessel is made of multiple parts. Instances can be obtained by various methods in *Parts*.

name

Internal name of the part, as used in part cfg files. For example "Mark1-2Pod".

Attribute Read-only, cannot be set

Return type string

title

Title of the part, as shown when the part is right clicked in-game. For example "Mk1-2 Command Pod".

Attribute Read-only, cannot be set

Return type string

cost

The cost of the part, in units of funds.

Attribute Read-only, cannot be set

Return type number

vessel

The vessel that contains this part.

Attribute Read-only, cannot be set

Return type Vessel

parent

The parts parent. Returns nil if the part does not have a parent. This, in combination with <code>Part.children</code>, can be used to traverse the vessels parts tree.

Attribute Read-only, cannot be set

Return type *Part*

Note: See the discussion on *Trees of Parts*.

children

The parts children. Returns an empty list if the part has no children. This, in combination with <code>Part.parent</code>, can be used to traverse the vessels parts tree.

Attribute Read-only, cannot be set

Return type List of Part

Note: See the discussion on *Trees of Parts*.

axially_attached

Whether the part is axially attached to its parent, i.e. on the top or bottom of its parent. If the part has no parent, returns False.

Attribute Read-only, cannot be set

Return type boolean

Note: See the discussion on Attachment Modes.

radially_attached

Whether the part is radially attached to its parent, i.e. on the side of its parent. If the part has no parent, returns False.

Attribute Read-only, cannot be set

Return type boolean

Note: See the discussion on *Attachment Modes*.

stage

The stage in which this part will be activated. Returns -1 if the part is not activated by staging.

Attribute Read-only, cannot be set

Return type number

Note: See the discussion on *Staging*.

decouple_stage

The stage in which this part will be decoupled. Returns -1 if the part is never decoupled from the vessel.

Attribute Read-only, cannot be set

Return type number

Note: See the discussion on *Staging*.

massless

Whether the part is massless.

Attribute Read-only, cannot be set

Return type boolean

mass

The current mass of the part, including resources it contains, in kilograms. Returns zero if the part is massless.

Attribute Read-only, cannot be set

Return type number

dry_mass

The mass of the part, not including any resources it contains, in kilograms. Returns zero if the part is massless.

Attribute Read-only, cannot be set

Return type number

impact_tolerance

The impact tolerance of the part, in meters per second.

Attribute Read-only, cannot be set

Return type number

temperature

Temperature of the part, in Kelvin.

Attribute Read-only, cannot be set

Return type number

skin_temperature

Temperature of the skin of the part, in Kelvin.

Attribute Read-only, cannot be set

Return type number

max_temperature

Maximum temperature that the part can survive, in Kelvin.

Attribute Read-only, cannot be set

Return type number

max_skin_temperature

Maximum temperature that the skin of the part can survive, in Kelvin.

Attribute Read-only, cannot be set

Return type number

external_temperature

Temperature of the atmosphere/vacuum surrounding the part, in Kelvin. This does not include heating from direct sunlight.

Attribute Read-only, cannot be set

Return type number

thermal mass

How much it takes to heat up the part.

Attribute Read-only, cannot be set

Return type number

thermal skin mass

How much it takes to heat up the part's skin.

Attribute Read-only, cannot be set

Return type number

thermal_resource_mass

How much it takes to heat up resources in the part.

Attribute Read-only, cannot be set

Return type number

thermal_conduction_flux

The speed that heat is conducting into or out of the part through contact with other parts. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type number

thermal convection flux

The speed that heat is convecting into or out of the part from the surrounding atmosphere. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type number

thermal_radiation_flux

The speed that heat is radiating into or out of the part from the surrounding vacuum. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type number

thermal_internal_flux

The speed that heat is generated by the part. For example, engines generate heat by burning fuel. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type number

thermal_skin_to_internal_flux

The speed that heat is conducting between the part's skin and its internals.

Attribute Read-only, cannot be set

Return type number

resources

A Resources object for the part.

Attribute Read-only, cannot be set

Return type Resources

crossfeed

Whether this part is crossfeed capable.

Attribute Read-only, cannot be set

Return type boolean

is fuel line

Whether this part is a fuel line.

Attribute Read-only, cannot be set

Return type boolean

fuel_lines_from

The parts that are connected to this part via fuel lines, where the direction of the fuel line is into this part.

Attribute Read-only, cannot be set

Return type List of *Part*

Note: See the discussion on Fuel Lines.

fuel_lines_to

The parts that are connected to this part via fuel lines, where the direction of the fuel line is out of this part.

Attribute Read-only, cannot be set

Return type List of Part

Note: See the discussion on *Fuel Lines*.

modules

The modules for this part.

Attribute Read-only, cannot be set

Return type List of Module

decoupler

A Decoupler if the part is a decoupler, otherwise nil.

Attribute Read-only, cannot be set

Return type Decoupler

docking_port

A *DockingPort* if the part is a docking port, otherwise nil.

Attribute Read-only, cannot be set

Return type DockingPort

engine

An *Engine* if the part is an engine, otherwise nil.

Attribute Read-only, cannot be set

Return type Engine

```
landing_gear
    A LandingGear if the part is a landing gear, otherwise nil.
        Attribute Read-only, cannot be set
        Return type LandingGear
landing leg
    A LandingLeg if the part is a landing leg, otherwise nil.
        Attribute Read-only, cannot be set
        Return type LandingLeg
launch_clamp
    A LaunchClamp if the part is a launch clamp, otherwise nil.
        Attribute Read-only, cannot be set
        Return type LaunchClamp
light
    A Light if the part is a light, otherwise nil.
        Attribute Read-only, cannot be set
        Return type Light
parachute
    A Parachute if the part is a parachute, otherwise nil.
        Attribute Read-only, cannot be set
        Return type Parachute
radiator
    A Radiator if the part is a radiator, otherwise nil.
        Attribute Read-only, cannot be set
        Return type Radiator
reaction_wheel
    A ReactionWheel if the part is a reaction wheel, otherwise nil.
        Attribute Read-only, cannot be set
        Return type ReactionWheel
resource_converter
    A ResourceConverter if the part is a resource converter, otherwise nil.
         Attribute Read-only, cannot be set
        Return type ResourceConverter
resource_harvester
    A ResourceHarvester if the part is a resource harvester, otherwise nil.
        Attribute Read-only, cannot be set
        Return type ResourceHarvester
sensor
    A Sensor if the part is a sensor, otherwise nil.
        Attribute Read-only, cannot be set
```

Return type Sensor

solar_panel

A SolarPanel if the part is a solar panel, otherwise nil.

Attribute Read-only, cannot be set

Return type SolarPanel

position (reference_frame)

The position of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

direction (reference_frame)

The direction of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

velocity (reference_frame)

The velocity of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

rotation (reference_frame)

The rotation of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number, number)

reference frame

The reference frame that is fixed relative to this part.

- •The origin is at the position of the part.
- •The axes rotate with the part.
- •The x, y and z axis directions depend on the design of the part.

Attribute Read-only, cannot be set

Return type ReferenceFrame

Note: For docking port parts, this reference frame is not necessarily equivalent to the reference frame for the docking port, returned by <code>DockingPort.reference_frame</code>.

Module

class SpaceCenter.Module

In KSP, each part has zero or more PartModules associated with it. Each one contains some of the functionality of the part. For example, an engine has a "ModuleEngines" PartModule that contains all the functionality of an engine. This class allows you to interact with KSPs PartModules, and any PartModules that have been added by other mods.

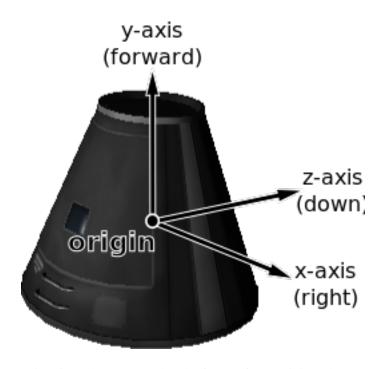


Fig. 6.7: Mk1 Command Pod reference frame origin and axes

name

Name of the PartModule. For example, "ModuleEngines".

Attribute Read-only, cannot be set

Return type string

part

The part that contains this module.

Attribute Read-only, cannot be set

Return type Part

fields

The modules field names and their associated values, as a dictionary. These are the values visible in the right-click menu of the part.

Attribute Read-only, cannot be set

Return type Map from string to string

has_field(name)

Returns True if the module has a field with the given name.

Parameters name (string) - Name of the field.

Return type boolean

get_field(name)

Returns the value of a field.

Parameters name (string) – Name of the field.

Return type string

events

A list of the names of all of the modules events. Events are the clickable buttons visible in the right-click menu of the part.

Attribute Read-only, cannot be set

Return type List of string

has_event (name)

True if the module has an event with the given name.

Parameters name (string) -

Return type boolean

trigger_event(name)

Trigger the named event. Equivalent to clicking the button in the right-click menu of the part.

Parameters name (string) -

actions

A list of all the names of the modules actions. These are the parts actions that can be assigned to action groups in the in-game editor.

Attribute Read-only, cannot be set

Return type List of string

has action(name)

True if the part has an action with the given name.

Parameters name (string) -

Return type boolean

set_action (name[, value = True])

Set the value of an action with the given name.

Parameters

- name (string) -
- value (boolean) -

Specific Types of Part

The following classes provide functionality for specific types of part.

```
• Decoupler
```

- Docking Port
- Engine
- Landing Gear
- Landing Leg
- Launch Clamp
- Light
- Parachute
- Radiator
- Resource Converter
- Resource Harvester
- · Reaction Wheel
- Sensor
- Solar Panel

Decoupler

```
class SpaceCenter.Decoupler
Obtained by calling Part.decoupler

part
The part object for this decoupler.

Attribute Read-only, cannot be set

Return type Part

decouple()
Fires the decoupler. Has no effect if the decoupler has already fired.

decoupled
Whether the decoupler has fired.

Attribute Read-only, cannot be set

Return type boolean

impulse
The impulse that the decoupler imparts when it is fired, in Newton seconds.

Attribute Read-only, cannot be set

Return type number
```

Docking Port

name

```
class SpaceCenter.DockingPort
    Obtained by calling Part.docking_port
    part
        The part object for this docking port.
        Attribute Read-only, cannot be set
        Return type Part
```

The port name of the docking port. This is the name of the port that can be set in the right click menu,

when the Docking Port Alignment Indicator mod is installed. If this mod is not installed, returns the title of the part (Part.title).

Attribute Can be read or written

Return type string

state

The current state of the docking port.

Attribute Read-only, cannot be set

Return type DockingPortState

docked_part

The part that this docking port is docked to. Returns nil if this docking port is not docked to anything.

Attribute Read-only, cannot be set

Return type Part

undock()

Undocks the docking port and returns the vessel that was undocked from. After undocking, the active vessel may change (active_vessel). This method can be called for either docking port in a docked pair - both calls will have the same effect. Returns nil if the docking port is not docked to anything.

Return type Vessel

reengage_distance

The distance a docking port must move away when it undocks before it becomes ready to dock with another port, in meters.

Attribute Read-only, cannot be set

Return type number

has shield

Whether the docking port has a shield.

Attribute Read-only, cannot be set

Return type boolean

shielded

The state of the docking ports shield, if it has one. Returns True if the docking port has a shield, and the shield is closed. Otherwise returns False. When set to True, the shield is closed, and when set to False the shield is opened. If the docking port does not have a shield, setting this attribute has no effect.

Attribute Can be read or written

Return type boolean

position (reference_frame)

The position of the docking port in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

direction (reference_frame)

The direction that docking port points in, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number)

rotation (reference_frame)

The rotation of the docking port, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type Tuple of (number, number, number, number)

reference frame

The reference frame that is fixed relative to this docking port, and oriented with the port.

- •The origin is at the position of the docking port.
- •The axes rotate with the docking port.
- •The x-axis points out to the right side of the docking port.
- •The y-axis points in the direction the docking port is facing.
- •The z-axis points out of the bottom off the docking port.

Attribute Read-only, cannot be set Return type ReferenceFrame

Note: This reference frame is not necessarily equivalent to the reference frame for the part, returned by <code>Part.reference_frame</code>.

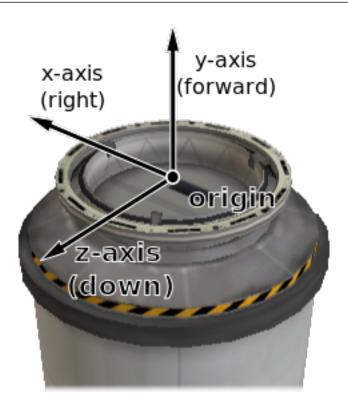


Fig. 6.8: Docking port reference frame origin and axes

class SpaceCenter.DockingPortState

See DockingPort.state.

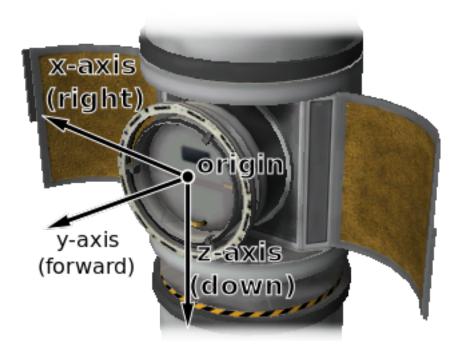


Fig. 6.9: Inline docking port reference frame origin and axes

ready

The docking port is ready to dock to another docking port.

docked

The docking port is docked to another docking port, or docked to another part (from the VAB/SPH).

docking

The docking port is very close to another docking port, but has not docked. It is using magnetic force to acquire a solid dock.

undocking

The docking port has just been undocked from another docking port, and is disabled until it moves away by a sufficient distance (DockingPort.reengage_distance).

shielded

The docking port has a shield, and the shield is closed.

moving

The docking ports shield is currently opening/closing.

Engine

${\bf class}\;{\tt SpaceCenter.Engine}$

Obtained by calling Part.engine.

part

The part object for this engine.

Attribute Read-only, cannot be set

Return type Part

active

Whether the engine is active. Setting this attribute may have no effect, depending on <code>Engine.can_shutdown</code> and <code>Engine.can_restart</code>.

Attribute Can be read or written

Return type boolean

thrust

The current amount of thrust being produced by the engine, in Newtons. Returns zero if the engine is not active or if it has no fuel.

Attribute Read-only, cannot be set

Return type number

available_thrust

The maximum available amount of thrust that can be produced by the engine, in Newtons. This takes *Engine.thrust_limit* into account, and is the amount of thrust produced by the engine when activated and the main throttle is set to 100%. Returns zero if the engine does not have any fuel.

Attribute Read-only, cannot be set

Return type number

max thrust

Gets the maximum amount of thrust that can be produced by the engine, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine.thrust_limit</code> is set to 100% and the main vessel's throttle is set to 100%.

Attribute Read-only, cannot be set

Return type number

max_vacuum_thrust

The maximum amount of thrust that can be produced by the engine in a vacuum, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine.thrust_limit</code> is set to 100%, the main vessel's throttle is set to 100% and the engine is in a vacuum.

Attribute Read-only, cannot be set

Return type number

thrust_limit

The thrust limiter of the engine. A value between 0 and 1. Setting this attribute may have no effect, for example the thrust limit for a solid rocket booster cannot be changed in flight.

Attribute Can be read or written

Return type number

specific_impulse

The current specific impulse of the engine, in seconds. Returns zero if the engine is not active.

Attribute Read-only, cannot be set

Return type number

vacuum specific impulse

The vacuum specific impulse of the engine, in seconds.

Attribute Read-only, cannot be set

Return type number

kerbin_sea_level_specific_impulse

The specific impulse of the engine at sea level on Kerbin, in seconds.

Attribute Read-only, cannot be set

Return type number

propellants

The names of resources that the engine consumes.

Attribute Read-only, cannot be set

Return type List of string

propellant_ratios

The ratios of resources that the engine consumes. A dictionary mapping resource names to the ratios at which they are consumed by the engine.

Attribute Read-only, cannot be set

Return type Map from string to number

has fuel

Whether the engine has run out of fuel (or flamed out).

Attribute Read-only, cannot be set

Return type boolean

throttle

The current throttle setting for the engine. A value between 0 and 1. This is not necessarily the same as the vessel's main throttle setting, as some engines take time to adjust their throttle (such as jet engines).

Attribute Read-only, cannot be set

Return type number

throttle_locked

Whether the *Control.throttle* affects the engine. For example, this is True for liquid fueled rockets, and False for solid rocket boosters.

Attribute Read-only, cannot be set

Return type boolean

can_restart

Whether the engine can be restarted once shutdown. If the engine cannot be shutdown, returns False. For example, this is True for liquid fueled rockets and False for solid rocket boosters.

Attribute Read-only, cannot be set

Return type boolean

can_shutdown

Gets whether the engine can be shutdown once activated. For example, this is True for liquid fueled rockets and False for solid rocket boosters.

Attribute Read-only, cannot be set

Return type boolean

gimballed

Whether the engine nozzle is gimballed, i.e. can provide a turning force.

Attribute Read-only, cannot be set

Return type boolean

gimbal range

The range over which the gimbal can move, in degrees.

Attribute Read-only, cannot be set

Return type number

gimbal locked

Whether the engines gimbal is locked in place. Setting this attribute has no effect if the engine is not gimballed.

Attribute Can be read or written

Return type boolean

gimbal_limit

The gimbal limiter of the engine. A value between 0 and 1. Returns 0 if the gimbal is locked or the engine is not gimballed. Setting this attribute has no effect if the engine is not gimballed.

Attribute Can be read or written

Return type number

Landing Gear

class SpaceCenter.LandingGear

Obtained by calling Part.landing_gear.

part

The part object for this landing gear.

Attribute Read-only, cannot be set

Return type Part

state

Gets the current state of the landing gear.

Attribute Read-only, cannot be set

Return type LandingGearState

deployed

Whether the landing gear is deployed.

Attribute Can be read or written

Return type boolean

class SpaceCenter.LandingGearState

See LandingGear.state.

deployed

Landing gear is fully deployed.

retracted

Landing gear is fully retracted.

deploying

Landing gear is being deployed.

retracting

Landing gear is being retracted.

Landing Leg

part

class SpaceCenter.LandingLeg

Obtained by calling Part.landing_leg.

```
The part object for this landing leg.
              Attribute Read-only, cannot be set
              Return type Part
     state
          The current state of the landing leg.
              Attribute Read-only, cannot be set
              Return type LandingLegState
     deployed
          Whether the landing leg is deployed.
              Attribute Can be read or written
              Return type boolean
class SpaceCenter.LandingLegState
     See LandingLeg. state.
     deployed
          Landing leg is fully deployed.
     retracted
          Landing leg is fully retracted.
     deploying
          Landing leg is being deployed.
     retracting
          Landing leg is being retracted.
     broken
          Landing leg is broken.
     repairing
          Landing leg is being repaired.
Launch Clamp
class SpaceCenter.LaunchClamp
     Obtained by calling Part.launch_clamp.
     part
          The part object for this launch clamp.
              Attribute Read-only, cannot be set
              Return type Part
     release()
          Releases the docking clamp. Has no effect if the clamp has already been released.
```

Light

```
class SpaceCenter.Light
     Obtained by calling Part.light.
     part
          The part object for this light.
               Attribute Read-only, cannot be set
               Return type Part
     active
          Whether the light is switched on.
               Attribute Can be read or written
               Return type boolean
     power_usage
          The current power usage, in units of charge per second.
               Attribute Read-only, cannot be set
               Return type number
Parachute
class SpaceCenter.Parachute
     Obtained by calling Part.parachute.
     part
          The part object for this parachute.
               Attribute Read-only, cannot be set
               Return type Part
     deploy()
          Deploys the parachute. This has no effect if the parachute has already been deployed.
          Whether the parachute has been deployed.
               Attribute Read-only, cannot be set
               Return type boolean
     state
          The current state of the parachute.
               Attribute Read-only, cannot be set
               Return type ParachuteState
     deploy_altitude
          The altitude at which the parachute will full deploy, in meters.
               Attribute Can be read or written
               Return type number
     deploy_min_pressure
          The minimum pressure at which the parachute will semi-deploy, in atmospheres.
```

Attribute Can be read or written

Return type number

class SpaceCenter.ParachuteState

See Parachute.state.

stowed

The parachute is safely tucked away inside its housing.

active

The parachute is still stowed, but ready to semi-deploy.

semi_deployed

The parachute has been deployed and is providing some drag, but is not fully deployed yet.

deployed

The parachute is fully deployed.

cut

The parachute has been cut.

Radiator

class SpaceCenter.Radiator

Obtained by calling Part.radiator.

part

The part object for this radiator.

Attribute Read-only, cannot be set

Return type Part

deployed

Whether the radiator is extended.

Attribute Can be read or written

Return type boolean

state

The current state of the radiator.

Attribute Read-only, cannot be set

Return type RadiatorState

class SpaceCenter.RadiatorState

RadiatorState

extended

Radiator is fully extended.

retracted

Radiator is fully retracted.

extending

Radiator is being extended.

retracting

Radiator is being retracted.

broken

Radiator is being broken.

Resource Converter

```
class SpaceCenter.ResourceConverter
     Obtained by calling Part.resource_converter.
     part
          The part object for this converter.
               Attribute Read-only, cannot be set
               Return type Part
     count
          The number of converters in the part.
               Attribute Read-only, cannot be set
               Return type number
     name (index)
          The name of the specified converter.
               Parameters index (number) – Index of the converter.
               Return type string
     active (index)
          True if the specified converter is active.
               Parameters index (number) – Index of the converter.
               Return type boolean
     start (index)
          Start the specified converter.
               Parameters index (number) – Index of the converter.
     stop (index)
          Stop the specified converter.
               Parameters index (number) – Index of the converter.
     state (index)
          The state of the specified converter.
               Parameters index (number) – Index of the converter.
               Return type ResourceConverterState
     status_info(index)
          Status information for the specified converter. This is the full status message shown in the in-game UI.
               Parameters index (number) – Index of the converter.
               Return type string
     inputs (index)
          List of the names of resources consumed by the specified converter.
               Parameters index (number) – Index of the converter.
```

Return type List of string

outputs (index)

List of the names of resources produced by the specified converter.

Parameters index (number) – Index of the converter.

Return type List of string

${\bf class} \; {\tt SpaceCenter.ResourceConverterState}$

See ResourceConverter.state().

running

Converter is running.

idle

Converter is idle.

missing_resource

Converter is missing a required resource.

storage_full

No available storage for output resource.

capacity

At preset resource capacity.

unknown

Unknown state. Possible with modified resource converters. In this case, check ResourceConverter.status_info() for more information.

Resource Harvester

class SpaceCenter.ResourceHarvester

Obtained by calling Part.resource_harvester.

part

The part object for this harvester.

Attribute Read-only, cannot be set

Return type Part

state

The state of the harvester.

Attribute Read-only, cannot be set

Return type ResourceHarvesterState

deployed

Whether the harvester is deployed.

Attribute Can be read or written

Return type boolean

active

Whether the harvester is actively drilling.

Attribute Can be read or written

Return type boolean

extraction rate

The rate at which the drill is extracting ore, in units per second.

Attribute Read-only, cannot be set

Return type number

thermal_efficiency

The thermal efficiency of the drill, as a percentage of its maximum.

Attribute Read-only, cannot be set

Return type number

core_temperature

The core temperature of the drill, in Kelvin.

Attribute Read-only, cannot be set

Return type number

optimum_core_temperature

The core temperature at which the drill will operate with peak efficiency, in Kelvin.

Attribute Read-only, cannot be set

Return type number

Reaction Wheel

class SpaceCenter.ReactionWheel

Obtained by calling Part.reaction_wheel.

part

The part object for this reaction wheel.

Attribute Read-only, cannot be set

Return type Part

active

Whether the reaction wheel is active.

Attribute Can be read or written

Return type boolean

broken

Whether the reaction wheel is broken.

Attribute Read-only, cannot be set

Return type boolean

pitch_torque

The torque in the pitch axis, in Newton meters.

Attribute Read-only, cannot be set

Return type number

yaw_torque

The torque in the yaw axis, in Newton meters.

Attribute Read-only, cannot be set

Return type number

roll_torque

The torque in the roll axis, in Newton meters.

Attribute Read-only, cannot be set

Return type number

Sensor

```
class SpaceCenter.Sensor
```

Obtained by calling Part.sensor.

part

The part object for this sensor.

Attribute Read-only, cannot be set

Return type Part

active

Whether the sensor is active.

Attribute Can be read or written

Return type boolean

value

The current value of the sensor.

Attribute Read-only, cannot be set

Return type string

power_usage

The current power usage of the sensor, in units of charge per second.

Attribute Read-only, cannot be set

Return type number

Solar Panel

class SpaceCenter.SolarPanel

Obtained by calling Part.solar_panel.

part

The part object for this solar panel.

Attribute Read-only, cannot be set

Return type Part

deployed

Whether the solar panel is extended.

Attribute Can be read or written

Return type boolean

state

The current state of the solar panel.

Attribute Read-only, cannot be set

Return type SolarPanelState

energy_flow

The current amount of energy being generated by the solar panel, in units of charge per second.

Attribute Read-only, cannot be set

Return type number

sun_exposure

The current amount of sunlight that is incident on the solar panel, as a percentage. A value between 0 and 1.

Attribute Read-only, cannot be set

Return type number

class SpaceCenter.SolarPanelState

See SolarPanel.state.

extended

Solar panel is fully extended.

retracted

Solar panel is fully retracted.

extending

Solar panel is being extended.

retracting

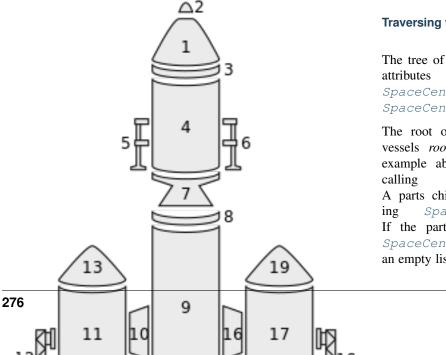
Solar panel is being retracted.

broken

Solar panel is broken.

Trees of Parts

Vessels in KSP are comprised of number of parts, connected to one another in a a tree structure. An example vessel is shown in Figure 1, and the corresponding tree of parts in Figure 2. The craft file for this example can also be downloaded here.



Traversing the Tree

The tree of parts can be traversed using the attributes SpaceCenter.Parts.root, SpaceCenter.Part.parent SpaceCenter.Part.children.

The root of the tree is the same as the vessels root part (part number 1 in the example above) and can be obtained by SpaceCenter.Parts.root. A parts children can be obtained by call-SpaceCenter.Part.children. If the part does not have any children, SpaceCenter.Part.children returns an empty list. A parts parent can be obtained

```
by calling SpaceCenter.Part.parent. If the part does not have a parent (as is the case for the root part), SpaceCenter.Part.parent returns nil.
```

The following Lua example uses these attributes to perform a depth-first traversal over all of the parts in a vessel:

```
local root = vessel.parts.root
local stack = {{root,0}}
while #stack > 0 do
    local part,depth = unpack(table.remove(stack))
    print(string.rep(' ', depth) .. part.title)
    for _,child in ipairs(part.children) do
        table.insert(stack, {child, depth+1})
    end
end
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1
TR-18A Stack Decoupler
 FL-T400 Fuel Tank
  LV-909 Liquid Fuel Engine
   TR-18A Stack Decoupler
    FL-T800 Fuel Tank
     LV-909 Liquid Fuel Engine
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
       TT18-A Launch Stability Enhancer
       FTX-2 External Fuel Duct
       LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
      TT-70 Radial Decoupler
      FL-T400 Fuel Tank
        TT18-A Launch Stability Enhancer
        FTX-2 External Fuel Duct
        LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
  LT-1 Landing Struts
  LT-1 Landing Struts
Mk16 Parachute
```

Attachment Modes

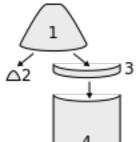
Parts can be attached to other parts either *radially* (on the side of the parent part) or *axially* (on the end of the parent part, to form a stack).

For example, in the vessel pictured above, the parachute (part 2) is *axially* connected to its

parent (the command pod – part 1), and the landing leg (part 5) is *radially* connected to its parent (the fuel tank – part 4).

The root part of a vessel (for example the command pod – part 1) does not have a parent part, so does not have an attachment mode. However, the part is consider to be *axially* attached to nothing.

The following Lua example does a depth-first traversal as before, but also prints out the attachment mode used by the part:



```
local root = vessel.parts.root
local stack = {{root, 0}}
while #stack > 0 do
local part,depth = unpack(table.remove(stack))
local attach_mode
if part.axially_attached then
   attach_mode = 'axial'
else -- radially_attached
   attach_mode = 'radial'
end
print(string.rep(' ', depth) .. part.title .. ' - ' .. attach_mode)
for _,child in ipairs(part.children) do
   table.insert(stack, {child, depth+1})
end
end
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1 - axial
TR-18A Stack Decoupler - axial
 FL-T400 Fuel Tank - axial
  LV-909 Liquid Fuel Engine - axial
   TR-18A Stack Decoupler - axial
    FL-T800 Fuel Tank - axial
     LV-909 Liquid Fuel Engine - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
  LT-1 Landing Struts - radial
  LT-1 Landing Struts - radial
Mk16 Parachute - axial
```

Fuel Lines

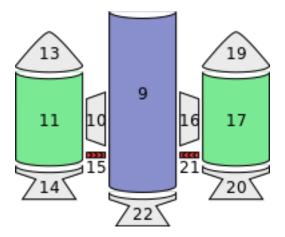


Fig. 6.12: **Figure 5** – Fuel lines from the example in Figure 1. Fuel flows from the parts highlighted in green, into the part highlighted in blue.

Fuel lines are considered parts, and are included in the parts tree (for example, as pictured in Figure 4). However, the parts tree does not contain information about which parts fuel lines connect to. The parent part of a fuel line is the part from which it will take fuel (as shown in Figure 4) however the part that it will send fuel to is not represented in the parts tree.

Figure 5 shows the fuel lines from the example vessel pictured earlier. Fuel line part 15 (in red) takes fuel from a fuel tank (part 11 - in green) and feeds it into another fuel tank (part 9 - in blue). The fuel line is therefore a child of part 11, but its connection to part 9 is not represented in the tree.

The attributes <code>SpaceCenter.Part.fuel_lines_from</code> and <code>SpaceCenter.Part.fuel_lines_to</code> can be used to discover these connections. In the example in Figure 5, when <code>SpaceCenter.Part.fuel_lines_to</code> is called on fuel tank part 11, it will return a list of parts containing just fuel tank part 9 (the blue part). When <code>SpaceCenter.Part.fuel_lines_from</code> is called on fuel tank part 9, it will return a list containing fuel tank parts 11 and 17 (the parts colored green).

Staging

Each part has two staging numbers associated with it: the stage in which the part is activated and the stage in which the part is decoupled. These values can be obtained using SpaceCenter.Part.stage and SpaceCenter.Part.decouple_stage respectively. For parts that are not activated by staging, SpaceCenter.Part.stage returns -1. For parts that are never decoupled, SpaceCenter.Part.decouple_stage returns a value of -1.

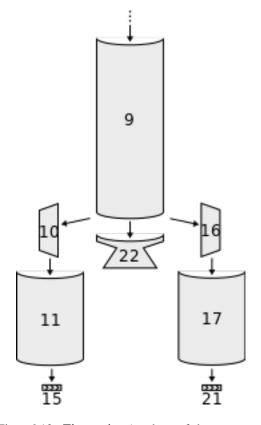


Fig. 6.13: **Figure 4** – A subset of the parts tree from Figure 2 above.

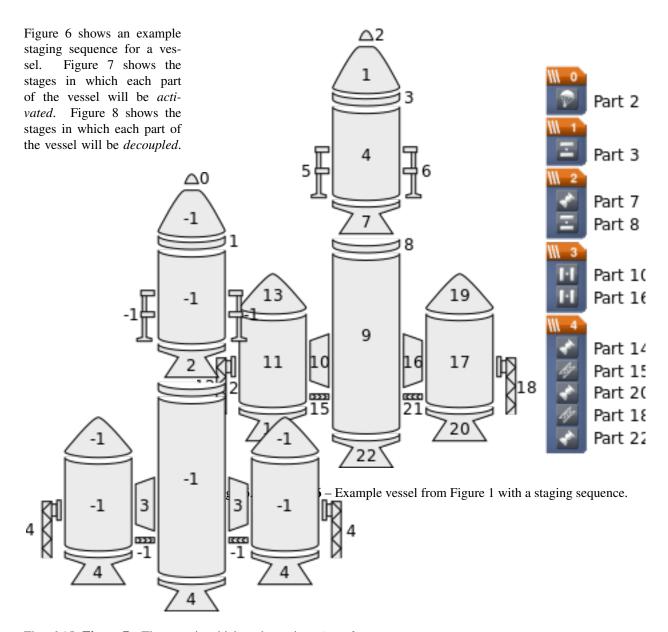
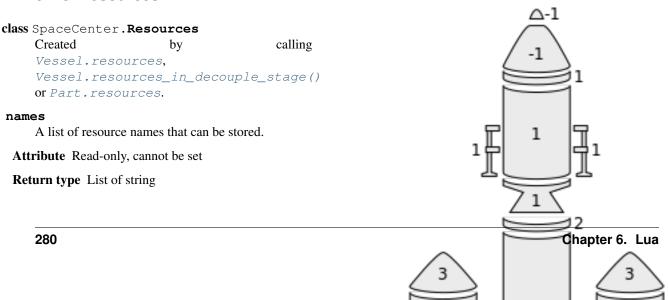


Fig. 6.15: **Figure 7** – The stage in which each part is *activated*.

6.2.8 Resources



has resource(name)

Check whether the named resource can be stored.

Parameters name (string) – The name of the resource.

Return type boolean

max (name)

Returns the amount of a resource that can be stored.

Parameters name (string) - The name of the resource.

Return type number

amount (name)

Returns the amount of a resource that is currently stored.

Parameters name (string) - The name of the resource.

Return type number

static density (name)

Returns the density of a resource, in kg/l.

Parameters name (string) – The name of the resource.

Return type number

static flow_mode (name)

Returns the flow mode of a resource.

Parameters name (string) - The name of the resource.

Return type ResourceFlowMode

class SpaceCenter.ResourceFlowMode

See Resources.flow_mode().

vessel

The resource flows to any part in the vessel. For example, electric charge.

stage

The resource flows from parts in the first stage, followed by the second, and so on. For example, mono-propellant.

adjacent

The resource flows between adjacent parts within the vessel. For example, liquid fuel or oxidizer.

none

The resource does not flow. For example, solid fuel.

6.2.9 Node

class SpaceCenter.Node

Represents a maneuver node. Can be created using Control.add_node().

prograde

The magnitude of the maneuver nodes delta-v in the prograde direction, in meters per second.

Attribute Can be read or written

Return type number

normal

The magnitude of the maneuver nodes delta-v in the normal direction, in meters per second.

Attribute Can be read or written

Return type number

radial

The magnitude of the maneuver nodes delta-v in the radial direction, in meters per second.

Attribute Can be read or written

Return type number

delta_v

The delta-v of the maneuver node, in meters per second.

Attribute Can be read or written

Return type number

Note: Does not change when executing the maneuver node. See *Node.remaining delta v.*

remaining_delta_v

Gets the remaining delta-v of the maneuver node, in meters per second. Changes as the node is executed. This is equivalent to the delta-v reported in-game.

Attribute Read-only, cannot be set

Return type number

burn_vector([reference_frame = None])

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s.

Parameters reference_frame

(ReferenceFrame) -

Return type Tuple of (number, number, number)

Note: Does not change when executing the maneuver node. See Node.remaining_burn_vector().

remaining_burn_vector([reference_frame = None])

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s. The direction and magnitude change as the burn is executed.

Parameters reference_frame

(ReferenceFrame) -

Return type Tuple of (number, number, number)

ut

The universal time at which the maneuver will occur, in seconds.

Attribute Can be read or written

Return type number

time to

The time until the maneuver node will be encountered, in seconds.

Attribute Read-only, cannot be set

Return type number

orbit

The orbit that results from executing the maneuver node.

Attribute Read-only, cannot be set

Return type Orbit

remove()

Removes the maneuver node.

reference_frame

Gets the reference frame that is fixed relative to the maneuver node's burn.

- •The origin is at the position of the maneuver node.
- •The y-axis points in the direction of the burn.
- The x-axis and z-axis point in arbitrary but fixed directions.

Attribute Read-only, cannot be set

Return type ReferenceFrame

orbital_reference_frame

Gets the reference frame that is fixed relative to the maneuver node, and orientated with the orbital prograde/normal/radial directions of the original orbit at the maneuver node's position.

- •The origin is at the position of the maneuver node.
- •The x-axis points in the orbital anti-radial direction of the original orbit, at the position of the maneuver node.
- •The y-axis points in the orbital prograde direction of the original orbit, at the position of the maneuver node.
- •The z-axis points in the orbital normal direction of the original orbit, at the position of the maneuver node.

Attribute Read-only, cannot be set

Return type ReferenceFrame

position (reference_frame)

Returns the position vector of the maneuver node in the given reference frame.

Parameters reference_frame

(ReferenceFrame) -

Return type Tuple of (number, number, number)

direction(reference_frame)

Returns the unit direction vector of the maneuver nodes burn in the given reference frame.

Parameters reference_frame

(ReferenceFrame) -

Return type Tuple of (number, number, number)

6.2.10 Comms

class SpaceCenter.Comms

Used to interact with RemoteTech. Created using a call to Vessel.comms.

Note: This class requires RemoteTech to be installed.

has_local_control

Whether the vessel can be controlled locally.

Attribute Read-only, cannot be set

Return type boolean

has_flight_computer

Whether the vessel has a RemoteTech flight computer on board.

Attribute Read-only, cannot be set

Return type boolean

has_connection

Whether the vessel can receive commands from the KSC or a command station.

Attribute Read-only, cannot be set

Return type boolean

has_connection_to_ground_station

Whether the vessel can transmit science data to a ground station.

Attribute Read-only, cannot be set

Return type boolean

signal_delay

The signal delay when sending commands to the vessel, in seconds.

Attribute Read-only, cannot be set

Return type number

signal_delay_to_ground_station

The signal delay between the vessel and the closest ground station, in seconds.

Attribute Read-only, cannot be set

Return type number

signal_delay_to_vessel (other)

Returns the signal delay between the current vessel and another vessel, in seconds.

Parameters other (Vessel) -

Return type number

6.2.11 ReferenceFrame

class SpaceCenter.ReferenceFrame

Represents a reference frame for positions, rotations and velocities. Contains:

- •The position of the origin.
- •The directions of the x, y and z axes.
- •The linear velocity of the frame.
- •The angular velocity of the frame.

Note: This class does not contain any properties or methods. It is only used as a parameter to other functions.

6.2.12 AutoPilot

class SpaceCenter.AutoPilot

Provides basic auto-piloting utilities for a vessel. Created by calling Vessel.auto_pilot.

engage()

Engage the auto-pilot.

disengage()

Disengage the auto-pilot.

wait()

Blocks until the vessel is pointing in the target direction (if set) and has the target roll (if set).

error

The error, in degrees, between the direction the ship has been asked to point in and the direction it is pointing in. Returns zero if the auto-pilot has not been engaged, SAS is not enabled, SAS is in stability assist mode, or no target direction is set.

Attribute Read-only, cannot be set

Return type number

roll_error

The error, in degrees, between the roll the ship has been asked to be in and the actual roll. Returns zero if the auto-pilot has not been engaged or no target roll is set.

Attribute Read-only, cannot be set

Return type number

reference frame

The reference frame for the target direction (AutoPilot.target direction).

Attribute Can be read or written

Return type ReferenceFrame

target_direction

The target direction. nil if no target direction is set.

Attribute Can be read or written

Return type Tuple of (number, number, number)

target_pitch_and_heading(pitch, heading)

Set (AutoPilot.target_direction) from a pitch and heading angle.

Parameters

• pitch (number) – Target pitch angle, in degrees between -90° and +90°.

286 Chapter 6. Lua

• **heading** (number) – Target heading angle, in degrees between 0° and 360°.

target_roll

The target roll, in degrees. NaN if no target roll is set

Attribute Can be read or written

Return type number

sas

The state of SAS.

Attribute Can be read or written

Return type boolean

Note: Equivalent to Control.sas

sas_mode

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Attribute Can be read or written

Return type SASMode

Note: Equivalent to Control.sas_mode

rotation_speed_multiplier

Target rotation speed multiplier. Defaults to 1.

Attribute Can be read or written

Return type number

max_rotation_speed

Maximum target rotation speed. Defaults to 1.

Attribute Can be read or written

Return type number

roll_speed_multiplier

Target roll speed multiplier. Defaults to 1.

Attribute Can be read or written

Return type number

max_roll_speed

Maximum target roll speed. Defaults to 1.

Attribute Can be read or written

Return type number

$set_pid_parameters([kp = 1.0][, ki = 0.0][, kd = 0.0])$

Sets the gains for the rotation rate PID controller.

Parameters

- **kp** (number) Proportional gain.
- ki (number) Integral gain.
- kd (number) Derivative gain.

6.2.13 Geometry Types

class SpaceCenter.Vector3

3-dimensional vectors are represented as a 3-tuple.

For example:

```
local krpc = require 'krpc.init'
local conn = krpc.connect()
local v = conn.space_center.active_vessel:flight().prograde
print(v[1], v[2], v[3])
```

class SpaceCenter.Quaternion

Quaternions (rotations in 3-dimensional space) are encoded as a 4-tuple containing the x, y, z and w components. For example:

```
local krpc = require 'krpc.init'
local conn = krpc.connect()
local q = conn.space_center.active_vessel:flight().rotation
print(q[1], q[2], q[3], q[4])
```

6.3 InfernalRobotics API

Provides RPCs to interact with the InfernalRobotics mod. Provides the following classes:

6.3.1 InfernalRobotics

class InfernalRobotics.InfernalRobotics

This service provides functionality to interact with the InfernalRobotics mod.

servo_groups

A list of all the servo groups in the active vessel.

Attribute Read-only, cannot be set

Return type List of ControlGroup

${\bf static\ servo_group_with_name}\ (name)$

Returns the servo group with the given *name* or nil if none exists. If multiple servo groups have the same name, only one of them is returned.

Parameters name (string) - Name of servo group to find

Return type ControlGroup

288 Chapter 6. Lua

static servo_with_name (name)

Returns the servo with the given *name*, from all servo groups, or nil if none exists. If multiple servos have the same name, only one of them is returned.

Parameters name (string) - Name of the servo to find.

Return type Servo

6.3.2 ControlGroup

class InfernalRobotics.ControlGroup

A group of servos, obtained by calling servo_groups or servo_group_with_name(). Represents the "Servo Groups" in the InfernalRobotics UI.

name

The name of the group.

Attribute Can be read or written

Return type string

forward key

The key assigned to be the "forward" key for the group.

Attribute Can be read or written

Return type string

reverse_key

The key assigned to be the "reverse" key for the group.

Attribute Can be read or written

Return type string

speed

The speed multiplier for the group.

Attribute Can be read or written

Return type number

expanded

Whether the group is expanded in the Infernal-Robotics UI.

Attribute Can be read or written

Return type boolean

servos

The servos that are in the group.

Attribute Read-only, cannot be set

Return type List of Servo

```
servo_with_name(name)
     Returns the servo with the given name from this
     group, or nil if none exists.
 Parameters name (string) – Name of servo to find.
 Return type Servo
move_right()
     Moves all of the servos in the group to the right.
move_left()
     Moves all of the servos in the group to the left.
move center()
     Moves all of the servos in the group to the center.
move_next_preset()
     Moves all of the servos in the group to the next
     preset.
move_prev_preset()
     Moves all of the servos in the group to the previous
     preset.
stop()
     Stops the servos in the group.
     6.3.3 Servo
class InfernalRobotics.Servo
     Represents a servo.
                                     Obtained
                         ControlGroup.servos,
     ControlGroup.servo_with_name()
     servo_with_name().
name
     The name of the servo.
 Attribute Can be read or written
 Return type string
highlight
     Whether the servo should be highlighted in-game.
 Attribute Write-only, cannot be read
 Return type boolean
position
     The position of the servo.
 Attribute Read-only, cannot be set
 Return type number
min_config_position
     The minimum position of the servo, specified by the
     part configuration.
```

Attribute Read-only, cannot be set

290 Chapter 6. Lua

Return type number

max_config_position

The maximum position of the servo, specified by the part configuration.

Attribute Read-only, cannot be set

Return type number

min_position

The minimum position of the servo, specified by the in-game tweak menu.

Attribute Can be read or written

Return type number

max_position

The maximum position of the servo, specified by the in-game tweak menu.

Attribute Can be read or written

Return type number

config_speed

The speed multiplier of the servo, specified by the part configuration.

Attribute Read-only, cannot be set

Return type number

speed

The speed multiplier of the servo, specified by the in-game tweak menu.

Attribute Can be read or written

Return type number

current_speed

The current speed at which the servo is moving.

Attribute Can be read or written

Return type number

acceleration

The current speed multiplier set in the UI.

Attribute Can be read or written

Return type number

is_moving

Whether the servo is moving.

Attribute Read-only, cannot be set

Return type boolean

is_free_moving

Whether the servo is freely moving.

Attribute Read-only, cannot be set

Return type boolean

is locked

Whether the servo is locked.

Attribute Can be read or written

Return type boolean

is_axis_inverted

Whether the servos axis is inverted.

Attribute Can be read or written

Return type boolean

move_right()

Moves the servo to the right.

move_left()

Moves the servo to the left.

move center()

Moves the servo to the center.

move_next_preset()

Moves the servo to the next preset.

move prev preset()

Moves the servo to the previous preset.

move_to (position, speed)

Moves the servo to *position* and sets the speed multiplier to *speed*.

Parameters

- **position** (*number*) The position to move the servo to.
- **speed** (number) Speed multiplier for the movement.

stop()

Stops the servo.

6.3.4 Example

The following example gets the control group named "MyGroup", prints out the names and positions of all of the servos in the group, then moves all of the servos to the right for 1 second.

```
local krpc = require 'krpc.init'
local platform = require 'krpc.platform'
local Types = require 'krpc.types'

local conn = krpc.connect(nil, nil, nil, 'InfernalRobotics Example')

local group = conn.infernal_robotics.servo_group_with_name('MyGroup')
if group == Types.none then
    print('Group not found')
```

292 Chapter 6. Lua

```
os.exit(1)
end

for _, servo in ipairs(group.servos) do
   print(servo.name, servo.position)
end

group:move_right()
platform.sleep(1)
group:stop()
```

6.4 Kerbal Alarm Clock API

Provides RPCs to interact with the Kerbal Alarm Clock mod. Provides the following classes:

6.4.1 KerbalAlarmClock

class KerbalAlarmClock. KerbalAlarmClock This service provides functionality to interact with the Kerbal Alarm Clock mod.

alarms

A list of all the alarms.

Attribute Read-only, cannot be set

Return type List of Alarm

static alarm_with_name (name)

Get the alarm with the given *name*, or nil if no alarms have that name. If more than one alarm has the name, only returns one of them.

Parameters name (*string*) – Name of the alarm to search for.

Return type Alarm

static alarms_with_type (type)

Get a list of alarms of the specified type.

Parameters type (AlarmType) - Type of alarm to return.

Return type List of Alarm

static create_alarm(type, name, ut)

Create a new alarm and return it.

Parameters

- **type** (AlarmType) Type of the new alarm.
- name (string) Name of the new alarm.
- **ut** (number) Time at which the new alarm should trigger.

Return type Alarm

6.4.2 Alarm

class KerbalAlarmClock.Alarm

Represents an alarm. Obtained by calling alarms, alarm_with_name() or alarms_with_type().

action

The action that the alarm triggers.

Attribute Can be read or written

Return type AlarmAction

margin

The number of seconds before the event that the alarm will fire.

Attribute Can be read or written

Return type number

time

The time at which the alarm will fire.

Attribute Can be read or written

Return type number

type

The type of the alarm.

Attribute Read-only, cannot be set

Return type AlarmType

id

The unique identifier for the alarm.

Attribute Read-only, cannot be set

Return type string

name

The short name of the alarm.

Attribute Can be read or written

Return type string

notes

The long description of the alarm.

Attribute Can be read or written

Return type string

remaining

The number of seconds until the alarm will fire.

Attribute Read-only, cannot be set

Return type number

294 Chapter 6. Lua

repeat

Whether the alarm will be repeated after it has fired.

Attribute Can be read or written

Return type boolean

repeat_period

The time delay to automatically create an alarm after it has fired.

Attribute Can be read or written

Return type number

vessel

The vessel that the alarm is attached to.

Attribute Can be read or written

Return type SpaceCenter. Vessel

xfer_origin_body

The celestial body the vessel is departing from.

Attribute Can be read or written

Return type SpaceCenter.CelestialBody

xfer_target_body

The celestial body the vessel is arriving at.

Attribute Can be read or written

Return type SpaceCenter.CelestialBody

${\tt remove}()$

Removes the alarm.

6.4.3 AlarmType

class KerbalAlarmClock.AlarmType

The type of an alarm.

raw

An alarm for a specific date/time or a specific period in the future.

maneuver

An alarm based on the next maneuver node on the current ships flight path. This node will be stored and can be restored when you come back to the ship.

maneuver_auto

See AlarmType.maneuver.

apoapsis

An alarm for furthest part of the orbit from the planet.

periapsis

An alarm for nearest part of the orbit from the planet.

ascending_node

Ascending node for the targeted object, or equatorial ascending node.

descending_node

Descending node for the targeted object, or equatorial descending node.

closest

An alarm based on the closest approach of this vessel to the targeted vessel, some number of orbits into the future.

contract

An alarm based on the expiry or deadline of contracts in career modes.

contract_auto

See AlarmType.contract.

crew

An alarm that is attached to a crew member.

distance

An alarm that is triggered when a selected target comes within a chosen distance.

earth time

An alarm based on the time in the "Earth" alternative Universe (aka the Real World).

launch_rendevous

An alarm that fires as your landed craft passes under the orbit of your target.

soi_change

An alarm manually based on when the next SOI point is on the flight path or set to continually monitor the active flight path and add alarms as it detects SOI changes.

soi_change_auto

See AlarmType.soi_change.

transfer

An alarm based on Interplanetary Transfer Phase Angles, i.e. when should I launch to planet X? Based on Kosmo Not's post and used in Olex's Calculator.

transfer_modelled

See AlarmType.transfer.

6.4.4 AlarmAction

class KerbalAlarmClock.AlarmAction

The action performed by an alarm when it fires.

do_nothing

Don't do anything at all...

296 Chapter 6. Lua

do_nothing_delete_when_passed

Don't do anything, and delete the alarm.

kill_warp

Drop out of time warp.

kill_warp_only

Drop out of time warp.

message_only

Display a message.

pause_game

Pause the game.

6.4.5 Example

The following example creates a new alarm for the active vessel. The alarm is set to trigger after 10 seconds have passed, and display a message.

```
local krpc = require 'krpc.init'
local conn = krpc.connect(nil, nil, nil, 'Kerbal Alarm Clock Example')

local alarm = conn.kerbal_alarm_clock.create_alarm(
    conn.kerbal_alarm_clock.AlarmType.raw,
    'My New Alarm',
    conn.space_center.ut+10)

alarm.notes = '10 seconds have now passed since the alarm was created.'
alarm.action = conn.kerbal_alarm_clock.AlarmAction.message_only
```

298 Chapter 6. Lua

CHAPTER

SEVEN

PYTHON

7.1 Python Client

The krpc module provides functionality to interact with a kRPC server from python. It can be installed from pypi.

7.1.1 Connecting to the Server

To connect to a server, use the *krpc.connect()* function. This returns a connection object through which you can interact with the server. For example to connect to a server running on the local machine:

```
import krpc
conn = krpc.connect(name='Example')
print(conn.krpc.get_status().version)
```

This function also accepts arguments that specify what address and port numbers to connect to. For example:

```
import krpc
conn = krpc.connect(address='my.domain.name', rpc_port=1000, stream_port=1001, name = 'Remote example
print(conn.krpc.get_status().version)
```

7.1.2 Interacting with the Server

Interaction with the server is performed via the client object (of type krpc.client.Client) returned when connecting to the server using krpc.connect().

Upon connecting, the client interrogates the server to find out what functionality it provides and dynamically adds all of the classes, methods, properties to the client object.

For example, all of the functionality provided by the SpaceCenter service is accessible via conn.space_center and the functionality provided by the InfernalRobotics service is accessible via conn.infernal_robotics. To explore the functionality provided by a service, you can use the help() function from an interactive terminal. For example, running help(conn.space_center) will list all of the classes, enumerations, procedures and properties provides by the SpaceCenter service. Or for a class, such as the vessel class provided by the SpaceCenter service by calling help(conn.space_center.Vessel).

Calling methods, getting or setting properties, etc. are mapped to remote procedure calls and passed to the server by the python client.

7.1.3 Streaming Data from the Server

A stream repeatedly executes a function on the server, with a fixed set of argument values. It provides a more efficient way of repeatedly getting the result of calling function on the server, without having to invoke it directly – which incurs communication overheads.

For example, consider the following loop that continuously prints out the position of the active vessel. This loop incurs significant communication overheads, as the vessel.position function is called repeatedly.

```
vessel = conn.space_center.active_vessel
refframe = vessel.orbit.body.reference_frame
while True:
    print vessel.position(refframe)
```

The following code achieves the same thing, but is far more efficient. It makes a single call to Client.add_stream() to create the stream, which avoids the communication overhead in the previous example.

```
vessel = conn.space_center.active_vessel
refframe = vessel.orbit.body.reference_frame
position = conn.add_stream(vessel.position, refframe)
while True:
    print position()
```

Streams are created by calling Client.add_stream() or using the with statement applied to Client.stream(). Both of these methods return an instance of the krpc.stream.Stream class.

Both methods and attributes can be streamed. The example given above demonstrates how to stream methods. The following example shows how to stream an attribute (in this case <code>vessel.control.abort</code>):

```
abort = conn.add_stream(getattr, vessel.control, 'abort')
while not abort():
...
```

7.1.4 Reference

krpc.connect([address='127.0.0.1'][, rpc_port=50000][, stream_port=50001][, name=None])
This function creates a connection to a kRPC server. It returns a krpc.client.Client object, through which the server can be communicated with.

Parameters

- address (string) The address of the server to connect to. Can either be a hostname or an IP address in dotted decimal notation. Defaults to '127.0.0.1'.
- rpc_port (int) The port number of the RPC Server. Defaults to 50000.
- **stream_port** (*int*) The port number of the Stream Server. Defaults to 50001.
- name (string) A descriptive name for the connection. This is passed to the server and appears, for example, in the client connection dialog on the in-game server window.

```
class krpc.client.Client
```

This class provides the interface for communicating with the server. It is dynamically populated with all the functionality provided by the server. Instances of this class should be obtained by calling krpc.connect().

```
add_stream (func, *args, **kwargs)
```

Create a stream for the function *func* called with arguments *args* and *kwargs*. Returns a krpc.stream.Stream object.

```
stream (func, *args, **kwargs)
```

Allows use of the with statement to create a stream and automatically remove it from the server when it goes out of scope. The function to be streamed should be passed as *func*, and its arguments as *args* and *kwargs*.

For example, to stream the result of method call vessel.position(refframe):

```
vessel = conn.space_center.active_vessel
refframe = vessel.orbit.body.reference_frame
with conn.stream(vessel.position, refframe) as pos:
    print('Position =', pos())
```

Or to stream the property conn.space_center.ut:

```
with conn.stream(getattr(conn.space_center, 'ut')) as ut:
    print('Universal Time =', ut())
```

close()

Closes the connection to the server.

krpc

The built-in KRPC class, providing basic interactions with the server.

```
Return type krpc.client.KRPC
```

```
class krpc.client.KRPC
```

This class provides access to the basic server functionality provided by the KRPC service. An instance can be obtained by calling krpc.client.Client.krpc. Most of this functionality is used internally by the python client (for example to create and remove streams) and therefore does not need to be used directly from application code. The only exception that may be useful is:

```
get_status()
```

Gets a status message from the server containing information including the server's version string and performance statistics.

For example, the following prints out the version string for the server:

```
print('Server version =', conn.krpc.get_status().version)
```

Or to get the rate at which the server is sending and receiving data over the network:

```
status = conn.krpc.get_status()
print('Data in =', (status.bytes_read_rate/1024.0), 'KB/s')
print('Data out =', (status.bytes_written_rate/1024.0), 'KB/s')
```

class krpc.stream.Stream

```
__call__()
```

Gets the most recently received value for the stream.

remove (

Remove the stream from the server.

7.1. Python Client 301

7.2 SpaceCenter API

7.2.1 SpaceCenter

class SpaceCenter.SpaceCenter

Provides functionality to interact with Kerbal Space Program. This includes controlling the active vessel, managing its resources, planning maneuver nodes and auto-piloting.

active_vessel

The currently active vessel.

Attribute Can be read or written

Return type Vessel

vessels

A list of all the vessels in the game.

Attribute Read-only, cannot be set

Return type list of Vessel

bodies

A dictionary of all celestial bodies (planets, moons, etc.) in the game, keyed by the name of the body.

Attribute Read-only, cannot be set

Return type dict from str to CelestialBody

target_body

The currently targeted celestial body.

Attribute Can be read or written

Return type CelestialBody

target_vessel

The currently targeted vessel.

Attribute Can be read or written

Return type Vessel

target_docking_port

The currently targeted docking port.

Attribute Can be read or written

Return type DockingPort

static clear_target()

Clears the current target.

static launch_vessel_from_vab (name)

Launch a new vessel from the VAB onto the launchpad.

Parameters name (str) – Name of the vessel's craft file.

static launch_vessel_from_sph (name)

Launch a new vessel from the SPH onto the runway.

Parameters name (str) – Name of the vessel's craft file.

ut

The current universal time in seconds.

Attribute Read-only, cannot be set

Return type float

g

The value of the gravitational constant G in $N(m/kg)^2$.

Attribute Read-only, cannot be set

Return type float

warp_mode

The current time warp mode. Returns <code>WarpMode.none</code> if time warp is not active, <code>WarpMode.rails</code> if regular "on-rails" time warp is active, or <code>WarpMode.physics</code> if physical time warp is active.

Attribute Read-only, cannot be set

Return type WarpMode

warp_rate

The current warp rate. This is the rate at which time is passing for either on-rails or physical time warp. For example, a value of 10 means time is passing 10x faster than normal. Returns 1 if time warp is not active.

Attribute Read-only, cannot be set

Return type float

warp_factor

The current warp factor. This is the index of the rate at which time is passing for either regular "on-rails" or physical time warp. Returns 0 if time warp is not active. When in on-rails time warp, this is equal to rails_warp_factor, and in physics time warp, this is equal to physics_warp_factor.

Attribute Read-only, cannot be set

Return type float

rails_warp_factor

The time warp rate, using regular "on-rails" time warp. A value between 0 and 7 inclusive. 0 means no time warp. Returns 0 if physical time warp is active. If requested time warp factor cannot be set, it will be set to the next lowest possible value. For example, if the vessel is too close to a planet. See the KSP wiki for details.

Attribute Can be read or written

Return type int

physics_warp_factor

The physical time warp rate. A value between 0 and 3 inclusive. 0 means no time warp. Returns 0 if regular "on-rails" time warp is active.

Attribute Can be read or written

Return type int

static can_rails_warp_at ([factor = 1])

Returns True if regular "on-rails" time warp can be used, at the specified warp *factor*. The maximum time warp rate is limited by various things, including how close the active vessel is to a planet. See the KSP wiki for details.

Parameters factor (int) – The warp factor to check.

Return type bool

maximum rails warp factor

The current maximum regular "on-rails" warp factor that can be set. A value between 0 and 7 inclusive. See the KSP wiki for details.

Attribute Read-only, cannot be set

Return type int

```
static warp_to (ut[, max\_rails\_rate = 100000.0][, max\_physics\_rate = 2.0])
```

Uses time acceleration to warp forward to a time in the future, specified by universal time *ut*. This call blocks until the desired time is reached. Uses regular "on-rails" or physical time warp as appropriate. For example, physical time warp is used when the active vessel is traveling through an atmosphere. When using regular "on-rails" time warp, the warp rate is limited by *max_rails_rate*, and when using physical time warp, the warp rate is limited by *max_physics_rate*.

Parameters

- ut (float) The universal time to warp to, in seconds.
- max_rails_rate (float) The maximum warp rate in regular "on-rails" time warp.
- max_physics_rate (float) The maximum warp rate in physical time warp.

Returns When the time warp is complete.

static transform_position (position, from, to)

Converts a position vector from one reference frame to another.

Parameters

- **position** (tuple) Position vector in reference frame from.
- **from** (ReferenceFrame) The reference frame that the position vector is in.
- to (ReferenceFrame) The reference frame to covert the position vector to.

Returns The corresponding position vector in reference frame *to*.

Return type tuple of (float, float, float)

static transform_direction (direction, from, to)

Converts a direction vector from one reference frame to another.

Parameters

- **direction** (tuple) Direction vector in reference frame from.
- from (ReferenceFrame) The reference frame that the direction vector is in.
- to (ReferenceFrame) The reference frame to covert the direction vector to.

Returns The corresponding direction vector in reference frame to.

Return type tuple of (float, float, float)

static transform_rotation (rotation, from, to)

Converts a rotation from one reference frame to another.

Parameters

- **rotation** (tuple) Rotation in reference frame from.
- from (ReferenceFrame) The reference frame that the rotation is in.
- to (ReferenceFrame) The corresponding rotation in reference frame to.

Returns The corresponding rotation in reference frame to.

Return type tuple of (float, float, float, float)

static transform_velocity (position, velocity, from, to)

Converts a velocity vector (acting at the specified position vector) from one reference frame to another. The position vector is required to take the relative angular velocity of the reference frames into account.

Parameters

- **position** (tuple) Position vector in reference frame from.
- **velocity** (tuple) Velocity vector in reference frame from.
- **from** (ReferenceFrame) The reference frame that the position and velocity vectors are in.
- to (ReferenceFrame) The reference frame to covert the velocity vector to.

Returns The corresponding velocity in reference frame *to*.

Return type tuple of (float, float, float)

far_available

Whether Ferram Aerospace Research is installed.

Attribute Read-only, cannot be set

Return type bool

remote_tech_available

Whether RemoteTech is installed.

Attribute Read-only, cannot be set

Return type bool

static draw_direction (direction, reference_frame, color[, length = 10.0])

Draw a direction vector on the active vessel.

Parameters

- **direction** (tuple) Direction to draw the line in.
- reference_frame (ReferenceFrame) Reference frame that the direction is in.
- **color** (*tuple*) The color to use for the line, as an RGB color.
- **length** (*float*) The length of the line. Defaults to 10.

static draw_line (start, end, reference_frame, color)

Draw a line.

Parameters

- **start** (tuple) Position of the start of the line.
- end (tuple) Position of the end of the line.
- reference_frame (ReferenceFrame) Reference frame that the position are in.
- color (tuple) The color to use for the line, as an RGB color.

static clear_drawing()

Remove all directions and lines currently being drawn.

class SpaceCenter.WarpMode

Returned by warp_mode

rails

Time warp is active, and in regular "on-rails" mode.

physics

Time warp is active, and in physical time warp mode.

none

Time warp is not active.

7.2.2 Vessel

class SpaceCenter. Vessel

These objects are used to interact with vessels in KSP. This includes getting orbital and flight data, manipulating control inputs and managing resources.

name

The name of the vessel.

Attribute Can be read or written

Return type str

type

The type of the vessel.

Attribute Can be read or written

Return type Vessel Type

situation

The situation the vessel is in.

Attribute Read-only, cannot be set

Return type VesselSituation

met

The mission elapsed time in seconds.

Attribute Read-only, cannot be set

Return type float

```
flight ( | reference_frame = None | )
```

Returns a Flight object that can be used to get flight telemetry for the vessel, in the specified reference frame.

Parameters reference_frame (ReferenceFrame) – Reference frame. Defaults to the vessel's surface reference frame (Vessel.surface_reference_frame).

Return type Flight

target

The target vessel. None if there is no target. When setting the target, the target cannot be the current vessel.

Attribute Can be read or written

Return type Vessel

orbit

The current orbit of the vessel.

Attribute Read-only, cannot be set

Return type Orbit

control

Returns a *Control* object that can be used to manipulate the vessel's control inputs. For example, its pitch/yaw/roll controls, RCS and thrust.

Attribute Read-only, cannot be set

Return type Control

auto_pilot

An AutoPilot object, that can be used to perform simple auto-piloting of the vessel.

Attribute Read-only, cannot be set

Return type AutoPilot

resources

A Resources object, that can used to get information about resources stored in the vessel.

Attribute Read-only, cannot be set

Return type Resources

resources_in_decouple_stage(stage[, cumulative = True])

Returns a Resources object, that can used to get information about resources stored in a given stage.

Parameters

- **stage** (*int*) Get resources for parts that are decoupled in this stage.
- **cumulative** (bool) When False, returns the resources for parts decoupled in just the given stage. When True returns the resources decoupled in the given stage and all subsequent stages combined.

Return type Resources

Note: For details on stage numbering, see the discussion on *Staging*.

parts

A *Parts* object, that can used to interact with the parts that make up this vessel.

Attribute Read-only, cannot be set

Return type Parts

comms

A Comms object, that can used to interact with RemoteTech for this vessel.

Attribute Read-only, cannot be set

Return type Comms

Note: Requires RemoteTech to be installed.

mass

The total mass of the vessel, including resources, in kg.

Attribute Read-only, cannot be set

Return type float

dry mass

The total mass of the vessel, excluding resources, in kg.

Attribute Read-only, cannot be set

Return type float

thrust

The total thrust currently being produced by the vessel's engines, in Newtons. This is computed by summing *Engine.thrust* for every engine in the vessel.

Attribute Read-only, cannot be set

Return type float

available_thrust

Gets the total available thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine.available_thrust</code> for every active engine in the vessel.

Attribute Read-only, cannot be set

Return type float

max thrust

The total maximum thrust that can be produced by the vessel's active engines, in Newtons. This is computed by summing <code>Engine.max_thrust</code> for every active engine.

Attribute Read-only, cannot be set

Return type float

max vacuum thrust

The total maximum thrust that can be produced by the vessel's active engines when the vessel is in a vacuum, in Newtons. This is computed by summing <code>Engine.max_vacuum_thrust</code> for every active engine.

Attribute Read-only, cannot be set

Return type float

specific_impulse

The combined specific impulse of all active engines, in seconds. This is computed using the formula described here.

Attribute Read-only, cannot be set

Return type float

vacuum_specific_impulse

The combined vacuum specific impulse of all active engines, in seconds. This is computed using the formula described here.

Attribute Read-only, cannot be set

Return type float

kerbin_sea_level_specific_impulse

The combined specific impulse of all active engines at sea level on Kerbin, in seconds. This is computed using the formula described here.

Attribute Read-only, cannot be set

Return type float

reference frame

The reference frame that is fixed relative to the vessel, and orientated with the vessel.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel.
- •The x-axis points out to the right of the vessel.
- •The y-axis points in the forward direction of the vessel.
- •The z-axis points out of the bottom off the vessel.

Attribute Read-only, cannot be set Return type ReferenceFrame

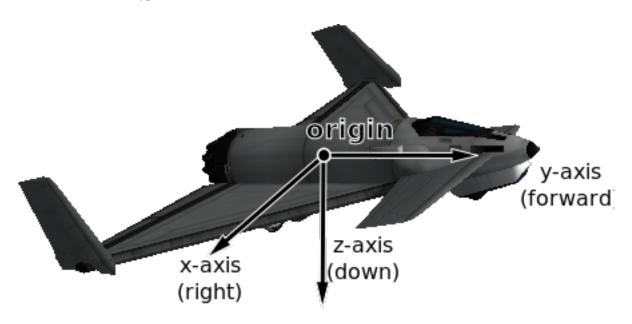


Fig. 7.1: Vessel reference frame origin and axes for the Aeris 3A aircraft

orbital_reference_frame

The reference frame that is fixed relative to the vessel, and orientated with the vessels orbital prograde/normal/radial directions.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Attribute Read-only, cannot be set

Return type ReferenceFrame

Note: Be careful not to confuse this with 'orbit' mode on the navball.

surface_reference_frame

The reference frame that is fixed relative to the vessel, and orientated with the surface of the body being orbited.

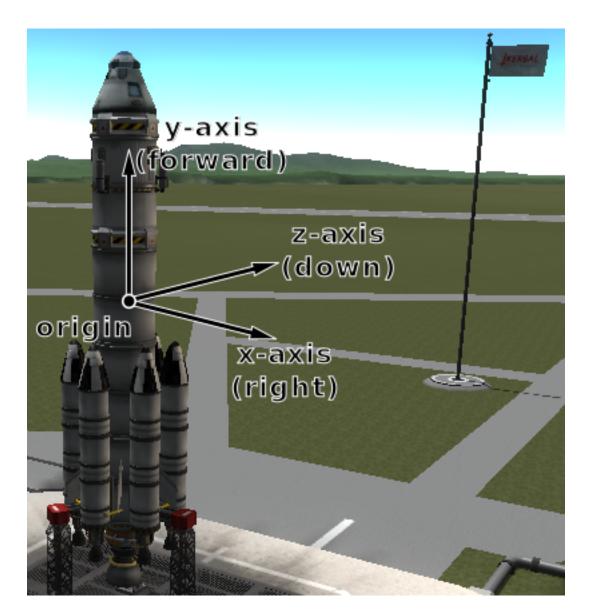


Fig. 7.2: Vessel reference frame origin and axes for the Kerbal-X rocket

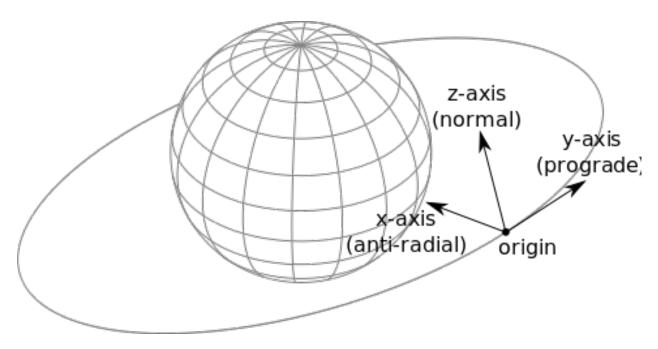


Fig. 7.3: Vessel orbital reference frame origin and axes

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the north and up directions on the surface of the body.
- •The x-axis points in the zenith direction (upwards, normal to the body being orbited, from the center of the body towards the center of mass of the vessel).
- •The y-axis points northwards towards the astronomical horizon (north, and tangential to the surface of the body the direction in which a compass would point when on the surface).
- •The z-axis points eastwards towards the astronomical horizon (east, and tangential to the surface of the body east on a compass when on the surface).

Attribute Read-only, cannot be set

Return type ReferenceFrame

Note: Be careful not to confuse this with 'surface' mode on the navball.

surface_velocity_reference_frame

The reference frame that is fixed relative to the vessel, and orientated with the velocity vector of the vessel relative to the surface of the body being orbited.

- •The origin is at the center of mass of the vessel.
- •The axes rotate with the vessel's velocity vector.
- •The y-axis points in the direction of the vessel's velocity vector, relative to the surface of the body being orbited.
- •The z-axis is in the plane of the astronomical horizon.
- •The x-axis is orthogonal to the other two axes.

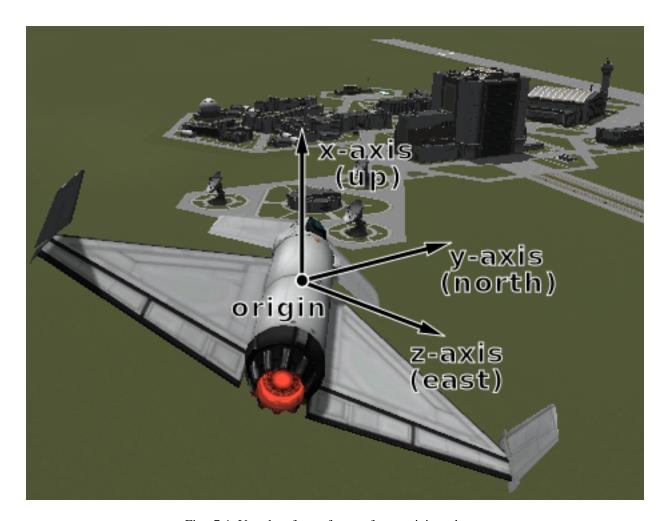


Fig. 7.4: Vessel surface reference frame origin and axes

Attribute Read-only, cannot be set Return type ReferenceFrame

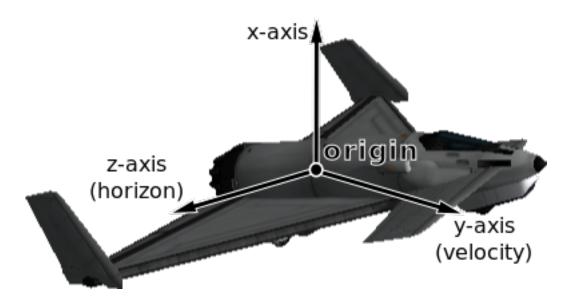


Fig. 7.5: Vessel surface velocity reference frame origin and axes

position (reference frame)

Returns the position vector of the center of mass of the vessel in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

velocity (reference_frame)

Returns the velocity vector of the center of mass of the vessel in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

rotation (reference_frame)

Returns the rotation of the center of mass of the vessel in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float, float)

direction (reference_frame)

Returns the direction in which the vessel is pointing, as a unit vector, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

angular_velocity (reference_frame)

Returns the angular velocity of the vessel in the given reference frame. The magnitude of the returned vector is the rotational speed in radians per second, and the direction of the vector indicates the axis of rotation (using the right hand rule).

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

```
class SpaceCenter.VesselType
     See Vessel.type.
     ship
          Ship.
     station
          Station.
     lander
          Lander.
     probe
          Probe.
     rover
          Rover.
     base
          Base.
     debris
          Debris.
{\bf class} \; {\tt SpaceCenter.VesselSituation}
     See Vessel.situation.
     docked
          Vessel is docked to another.
     escaping
          Escaping.
     flying
          Vessel is flying through an atmosphere.
          Vessel is landed on the surface of a body.
     orbiting
          Vessel is orbiting a body.
     pre_launch
          Vessel is awaiting launch.
     splashed
          Vessel has splashed down in an ocean.
     sub orbital
          Vessel is on a sub-orbital trajectory.
7.2.3 CelestialBody
class SpaceCenter.CelestialBody
     Represents a celestial body (such as a planet or moon).
     name
          The name of the body.
              Attribute Read-only, cannot be set
              Return type str
```

satellites

A list of celestial bodies that are in orbit around this celestial body.

Attribute Read-only, cannot be set

Return type list of CelestialBody

orbit

The orbit of the body.

Attribute Read-only, cannot be set

Return type Orbit

mass

The mass of the body, in kilograms.

Attribute Read-only, cannot be set

Return type float

gravitational_parameter

The standard gravitational parameter of the body in m^3s^{-2} .

Attribute Read-only, cannot be set

Return type float

surface_gravity

The acceleration due to gravity at sea level (mean altitude) on the body, in m/s^2 .

Attribute Read-only, cannot be set

Return type float

rotational_period

The sidereal rotational period of the body, in seconds.

Attribute Read-only, cannot be set

Return type float

rotational_speed

The rotational speed of the body, in radians per second.

Attribute Read-only, cannot be set

Return type float

equatorial_radius

The equatorial radius of the body, in meters.

Attribute Read-only, cannot be set

Return type float

surface_height (latitude, longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water this is equal to 0.

Parameters

- latitude (float) Latitude in degrees
- longitude (float) Longitude in degrees

Return type float

bedrock height (latitude, longitude)

The height of the surface relative to mean sea level at the given position, in meters. When over water, this is the height of the sea-bed and is therefore a negative value.

Parameters

- latitude (float) Latitude in degrees
- longitude (float) Longitude in degrees

Return type float

msl_position (latitude, longitude, reference_frame)

The position at mean sea level at the given latitude and longitude, in the given reference frame.

Parameters

- latitude (float) Latitude in degrees
- longitude (float) Longitude in degrees
- reference_frame (ReferenceFrame) Reference frame for the returned position vector

Return type tuple of (float, float, float)

surface_position (latitude, longitude, reference_frame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position of the surface of the water.

Parameters

- latitude (float) Latitude in degrees
- longitude (float) Longitude in degrees
- reference_frame (ReferenceFrame) Reference frame for the returned position vector

Return type tuple of (float, float, float)

bedrock_position (latitude, longitude, reference_frame)

The position of the surface at the given latitude and longitude, in the given reference frame. When over water, this is the position at the bottom of the sea-bed.

Parameters

- latitude (float) Latitude in degrees
- longitude (float) Longitude in degrees
- reference_frame (ReferenceFrame) Reference frame for the returned position vector

Return type tuple of (float, float, float)

sphere_of_influence

The radius of the sphere of influence of the body, in meters.

Attribute Read-only, cannot be set

Return type float

has_atmosphere

True if the body has an atmosphere.

Attribute Read-only, cannot be set

Return type bool

atmosphere_depth

The depth of the atmosphere, in meters.

Attribute Read-only, cannot be set

Return type float

has_atmospheric_oxygen

True if there is oxygen in the atmosphere, required for air-breathing engines.

Attribute Read-only, cannot be set

Return type bool

reference_frame

The reference frame that is fixed relative to the celestial body.

- •The origin is at the center of the body.
- •The axes rotate with the body.
- •The x-axis points from the center of the body towards the intersection of the prime meridian and equator (the position at 0° longitude, 0° latitude).
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points from the center of the body towards the equator at 90°E longitude.

Attribute Read-only, cannot be set Return type ReferenceFrame

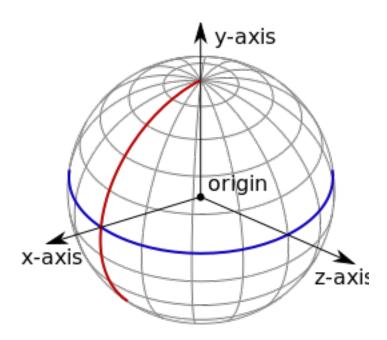


Fig. 7.6: Celestial body reference frame origin and axes. The equator is shown in blue, and the prime meridian in red.

non_rotating_reference_frame

The reference frame that is fixed relative to this celestial body, and orientated in a fixed direction (it does not rotate with the body).

•The origin is at the center of the body.

- •The axes do not rotate.
- •The x-axis points in an arbitrary direction through the equator.
- •The y-axis points from the center of the body towards the north pole.
- •The z-axis points in an arbitrary direction through the equator.

Attribute Read-only, cannot be set

Return type ReferenceFrame

orbital_reference_frame

Gets the reference frame that is fixed relative to this celestial body, but orientated with the body's orbital prograde/normal/radial directions.

- •The origin is at the center of the body.
- •The axes rotate with the orbital prograde/normal/radial directions.
- •The x-axis points in the orbital anti-radial direction.
- •The y-axis points in the orbital prograde direction.
- •The z-axis points in the orbital normal direction.

Attribute Read-only, cannot be set

Return type ReferenceFrame

position (reference_frame)

Returns the position vector of the center of the body in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

velocity(reference_frame)

Returns the velocity vector of the body in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

rotation (reference_frame)

Returns the rotation of the body in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float, float)

direction (reference_frame)

Returns the direction in which the north pole of the celestial body is pointing, as a unit vector, in the specified reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

angular_velocity(reference_frame)

Returns the angular velocity of the body in the specified reference frame. The magnitude of the vector is the rotational speed of the body, in radians per second, and the direction of the vector indicates the axis of rotation, using the right-hand rule.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

7.2.4 Flight

class SpaceCenter.Flight

Used to get flight telemetry for a vessel, by calling <code>Vessel.flight()</code>. All of the information returned by this class is given in the reference frame passed to that method.

Note: To get orbital information, such as the apoapsis or inclination, see *Orbit*.

g_force

The current G force acting on the vessel in m/s^2 .

Attribute Read-only, cannot be set

Return type float

mean_altitude

The altitude above sea level, in meters.

Attribute Read-only, cannot be set

Return type float

surface_altitude

The altitude above the surface of the body or sea level, whichever is closer, in meters.

Attribute Read-only, cannot be set

Return type float

bedrock_altitude

The altitude above the surface of the body, in meters. When over water, this is the altitude above the sea floor.

Attribute Read-only, cannot be set

Return type float

elevation

The elevation of the terrain under the vessel, in meters. This is the height of the terrain above sea level, and is negative when the vessel is over the sea.

Attribute Read-only, cannot be set

Return type float

latitude

The latitude of the vessel for the body being orbited, in degrees.

Attribute Read-only, cannot be set

Return type float

longitude

The longitude of the vessel for the body being orbited, in degrees.

Attribute Read-only, cannot be set

Return type float

velocity

The velocity vector of the vessel. The magnitude of the vector is the speed of the vessel in meters per second. The direction of the vector is the direction of the vessels motion.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

speed

The speed of the vessel in meters per second.

Attribute Read-only, cannot be set

Return type float

horizontal_speed

The horizontal speed of the vessel in meters per second.

Attribute Read-only, cannot be set

Return type float

vertical speed

The vertical speed of the vessel in meters per second.

Attribute Read-only, cannot be set

Return type float

center of mass

The position of the center of mass of the vessel.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

rotation

The rotation of the vessel.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float, float)

direction

The direction vector that the vessel is pointing in.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

pitch

The pitch angle of the vessel relative to the horizon, in degrees. A value between -90° and +90°.

Attribute Read-only, cannot be set

Return type float

heading

The heading angle of the vessel relative to north, in degrees. A value between 0° and 360°.

Attribute Read-only, cannot be set

Return type float

roll

The roll angle of the vessel relative to the horizon, in degrees. A value between -180° and $+180^{\circ}$.

Attribute Read-only, cannot be set

Return type float

prograde

The unit direction vector pointing in the prograde direction.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

retrograde

The unit direction vector pointing in the retrograde direction.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

normal

The unit direction vector pointing in the normal direction.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

anti_normal

The unit direction vector pointing in the anti-normal direction.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

radial

The unit direction vector pointing in the radial direction.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

anti_radial

The unit direction vector pointing in the anti-radial direction.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

atmosphere_density

The current density of the atmosphere around the vessel, in kg/m^3 .

Attribute Read-only, cannot be set

Return type float

dynamic pressure

The dynamic pressure acting on the vessel, in Pascals. This is a measure of the strength of the aerodynamic forces. It is equal to $\frac{1}{2}$ air density velocity². It is commonly denoted as Q.

Attribute Read-only, cannot be set

Return type float

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

static_pressure

The static atmospheric pressure acting on the vessel, in Pascals.

Attribute Read-only, cannot be set

Return type float

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

aerodynamic force

The total aerodynamic forces acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

lift

The aerodynamic lift currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

drag

The aerodynamic drag currently acting on the vessel, as a vector pointing in the direction of the force, with its magnitude equal to the strength of the force in Newtons.

Attribute Read-only, cannot be set

Return type tuple of (float, float, float)

Note: Calculated using KSPs stock aerodynamic model. Not available when Ferram Aerospace Research is installed.

speed_of_sound

The speed of sound, in the atmosphere around the vessel, in m/s.

Attribute Read-only, cannot be set

Return type float

Note: Not available when Ferram Aerospace Research is installed.

mach

The speed of the vessel, in multiples of the speed of sound.

Attribute Read-only, cannot be set

Return type float

Note: Not available when Ferram Aerospace Research is installed.

equivalent_air_speed

The equivalent air speed of the vessel, in m/s.

Attribute Read-only, cannot be set

Return type float

Note: Not available when Ferram Aerospace Research is installed.

terminal_velocity

An estimate of the current terminal velocity of the vessel, in m/s. This is the speed at which the drag forces cancel out the force of gravity.

Attribute Read-only, cannot be set

Return type float

Note: Calculated using KSPs stock aerodynamic model, or Ferram Aerospace Research if it is installed.

angle_of_attack

Gets the pitch angle between the orientation of the vessel and its velocity vector, in degrees.

Attribute Read-only, cannot be set

Return type float

sideslip_angle

Gets the yaw angle between the orientation of the vessel and its velocity vector, in degrees.

Attribute Read-only, cannot be set

Return type float

total_air_temperature

The total air temperature of the atmosphere around the vessel, in Kelvin. This temperature includes the Flight.static_air_temperature and the vessel's kinetic energy.

Attribute Read-only, cannot be set

Return type float

static air temperature

The static (ambient) temperature of the atmosphere around the vessel, in Kelvin.

Attribute Read-only, cannot be set

Return type float

stall_fraction

Gets the current amount of stall, between 0 and 1. A value greater than 0.005 indicates a minor stall and a value greater than 0.5 indicates a large-scale stall.

Attribute Read-only, cannot be set

Return type float

Note: Requires Ferram Aerospace Research.

drag_coefficient

Gets the coefficient of drag. This is the amount of drag produced by the vessel. It depends on air speed, air density and wing area.

Attribute Read-only, cannot be set

Return type float

Note: Requires Ferram Aerospace Research.

lift coefficient

Gets the coefficient of lift. This is the amount of lift produced by the vessel, and depends on air speed, air density and wing area.

Attribute Read-only, cannot be set

Return type float

Note: Requires Ferram Aerospace Research.

ballistic coefficient

Gets the ballistic coefficient.

Attribute Read-only, cannot be set

Return type float

Note: Requires Ferram Aerospace Research.

thrust_specific_fuel_consumption

Gets the thrust specific fuel consumption for the jet engines on the vessel. This is a measure of the efficiency of the engines, with a lower value indicating a more efficient vessel. This value is the number of Newtons of fuel that are burned, per hour, to product one newton of thrust.

Attribute Read-only, cannot be set

Return type float

Note: Requires Ferram Aerospace Research.

7.2.5 Orbit

class SpaceCenter.Orbit

Describes an orbit. For example, the orbit of a vessel, obtained by calling <code>Vessel.orbit</code>, or a celestial body, obtained by calling <code>CelestialBody.orbit</code>.

body

The celestial body (e.g. planet or moon) around which the object is orbiting.

Attribute Read-only, cannot be set

Return type CelestialBody

apoapsis

Gets the apoapsis of the orbit, in meters, from the center of mass of the body being orbited.

Attribute Read-only, cannot be set

Return type float

Note: For the apoapsis altitude reported on the in-game map view, use Orbit.apoapsis_altitude.

periapsis

The periapsis of the orbit, in meters, from the center of mass of the body being orbited.

Attribute Read-only, cannot be set

Return type float

Note: For the periapsis altitude reported on the in-game map view, use Orbit.periapsis_altitude.

apoapsis_altitude

The apoapsis of the orbit, in meters, above the sea level of the body being orbited.

Attribute Read-only, cannot be set

Return type float

Note: This is equal to *Orbit* . *apoapsis* minus the equatorial radius of the body.

periapsis_altitude

The periapsis of the orbit, in meters, above the sea level of the body being orbited.

Attribute Read-only, cannot be set

Return type float

Note: This is equal to *Orbit.periapsis* minus the equatorial radius of the body.

semi_major_axis

The semi-major axis of the orbit, in meters.

Attribute Read-only, cannot be set

Return type float

semi_minor_axis

The semi-minor axis of the orbit, in meters.

Attribute Read-only, cannot be set

Return type float

radius

The current radius of the orbit, in meters. This is the distance between the center of mass of the object in orbit, and the center of mass of the body around which it is orbiting.

Attribute Read-only, cannot be set

Return type float

Note: This value will change over time if the orbit is elliptical.

speed

The current orbital speed of the object in meters per second.

Attribute Read-only, cannot be set

Return type float

Note: This value will change over time if the orbit is elliptical.

period

The orbital period, in seconds.

Attribute Read-only, cannot be set

Return type float

time_to_apoapsis

The time until the object reaches apoapsis, in seconds.

Attribute Read-only, cannot be set

Return type float

time_to_periapsis

The time until the object reaches periapsis, in seconds.

Attribute Read-only, cannot be set

Return type float

eccentricity

The eccentricity of the orbit.

Attribute Read-only, cannot be set

Return type float

inclination

The inclination of the orbit, in radians.

Attribute Read-only, cannot be set

Return type float

longitude_of_ascending_node

The longitude of the ascending node, in radians.

Attribute Read-only, cannot be set

Return type float

argument_of_periapsis

The argument of periapsis, in radians.

Attribute Read-only, cannot be set

Return type float

mean_anomaly_at_epoch

The mean anomaly at epoch.

Attribute Read-only, cannot be set

Return type float

epoch

The time since the epoch (the point at which the mean anomaly at epoch was measured, in seconds.

Attribute Read-only, cannot be set

Return type float

mean_anomaly

The mean anomaly.

Attribute Read-only, cannot be set

Return type float

eccentric_anomaly

The eccentric anomaly.

Attribute Read-only, cannot be set

Return type float

static reference_plane_normal (reference_frame)

The unit direction vector that is normal to the orbits reference plane, in the given reference frame. The reference plane is the plane from which the orbits inclination is measured.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

static reference_plane_direction (reference_frame)

The unit direction vector from which the orbits longitude of ascending node is measured, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

time_to_soi_change

The time until the object changes sphere of influence, in seconds. Returns NaN if the object is not going to change sphere of influence.

Attribute Read-only, cannot be set

Return type float

next_orbit

If the object is going to change sphere of influence in the future, returns the new orbit after the change. Otherwise returns None.

Attribute Read-only, cannot be set

Return type Orbit

7.2.6 Control

class SpaceCenter.Control

Used to manipulate the controls of a vessel. This includes adjusting the throttle, enabling/disabling systems such as SAS and RCS, or altering the direction in which the vessel is pointing.

Note: Control inputs (such as pitch, yaw and roll) are zeroed when all clients that have set one or more of these inputs are no longer connected.

sas

The state of SAS.

Attribute Can be read or written

Return type bool

Note: Equivalent to AutoPilot.sas

sas mode

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Attribute Can be read or written

Return type SASMode

Note: Equivalent to AutoPilot.sas_mode

speed_mode

The current SpeedMode of the navball. This is the mode displayed next to the speed at the top of the navball.

Attribute Can be read or written

Return type SpeedMode

rcs

The state of RCS.

Attribute Can be read or written

Return type bool

gear

The state of the landing gear/legs.

Attribute Can be read or written

Return type bool

lights

The state of the lights.

Attribute Can be read or written

Return type bool

brakes

The state of the wheel brakes.

Attribute Can be read or written

Return type bool

abort

The state of the abort action group.

Attribute Can be read or written

Return type bool

throttle

The state of the throttle. A value between 0 and 1.

Attribute Can be read or written

Return type float

pitch

The state of the pitch control. A value between -1 and 1. Equivalent to the w and s keys.

Attribute Can be read or written

Return type float

yaw

The state of the yaw control. A value between -1 and 1. Equivalent to the a and d keys.

Attribute Can be read or written

Return type float

roll

The state of the roll control. A value between -1 and 1. Equivalent to the q and e keys.

Attribute Can be read or written

Return type float

forward

The state of the forward translational control. A value between -1 and 1. Equivalent to the h and n keys.

Attribute Can be read or written

Return type float

up

The state of the up translational control. A value between -1 and 1. Equivalent to the i and k keys.

Attribute Can be read or written

Return type float

right

The state of the right translational control. A value between -1 and 1. Equivalent to the j and l keys.

Attribute Can be read or written

Return type float

wheel_throttle

The state of the wheel throttle. A value between -1 and 1. A value of 1 rotates the wheels forwards, a value of -1 rotates the wheels backwards.

Attribute Can be read or written

Return type float

wheel_steering

The state of the wheel steering. A value between -1 and 1. A value of 1 steers to the left, and a value of -1 steers to the right.

Attribute Can be read or written

Return type float

current stage

The current stage of the vessel. Corresponds to the stage number in the in-game UI.

Attribute Read-only, cannot be set

Return type int

activate_next_stage()

Activates the next stage. Equivalent to pressing the space bar in-game.

Returns A list of vessel objects that are jettisoned from the active vessel.

Return type list of Vessel

get_action_group(group)

Returns True if the given action group is enabled.

Parameters group (int) – A number between 0 and 9 inclusive.

Return type bool

set_action_group (group, state)

Sets the state of the given action group (a value between 0 and 9 inclusive).

Parameters

- group (int) A number between 0 and 9 inclusive.
- state (bool) -

toggle_action_group(group)

Toggles the state of the given action group.

Parameters group (int) – A number between 0 and 9 inclusive.

```
add_node (ut[, prograde = 0.0][, normal = 0.0][, radial = 0.0])
```

Creates a maneuver node at the given universal time, and returns a *Node* object that can be used to modify it. Optionally sets the magnitude of the delta-v for the maneuver node in the prograde, normal and radial directions.

Parameters

- **ut** (*float*) Universal time of the maneuver node.
- **prograde** (*float*) Delta-v in the prograde direction.
- **normal** (float) Delta-v in the normal direction.
- radial (float) Delta-v in the radial direction.

Return type Node

nodes

Returns a list of all existing maneuver nodes, ordered by time from first to last.

Attribute Read-only, cannot be set

Return type list of Node

remove_nodes()

Remove all maneuver nodes.

class SpaceCenter.SASMode

The behavior of the SAS auto-pilot. See AutoPilot.sas_mode.

stability assist

Stability assist mode. Dampen out any rotation.

maneuver

Point in the burn direction of the next maneuver node.

prograde

Point in the prograde direction.

retrograde

Point in the retrograde direction.

normal

Point in the orbit normal direction.

anti normal

Point in the orbit anti-normal direction.

radial

Point in the orbit radial direction.

anti_radial

Point in the orbit anti-radial direction.

target

Point in the direction of the current target.

anti_target

Point away from the current target.

class SpaceCenter.SpeedMode

See Control.speed_mode.

orbit

Speed is relative to the vessel's orbit.

surface

Speed is relative to the surface of the body being orbited.

target

Speed is relative to the current target.

7.2.7 Parts

The following classes allow interaction with a vessels individual parts.

- Parts
- Part
- Module
- Specific Types of Part
 - Decoupler
 - Docking Port
 - Engine
 - Landing Gear
 - Landing Leg
 - Launch Clamp
 - Light
 - Parachute
 - Radiator
 - Resource Converter
 - Resource Harvester
 - Reaction Wheel
 - Sensor
 - Solar Panel
- Trees of Parts
 - Traversing the Tree
 - Attachment Modes
- Fuel Lines
- Staging

Parts

class SpaceCenter.Parts

Instances of this class are used to interact with the parts of a vessel. An instance can be obtained by calling *Vessel.parts*.

all

A list of all of the vessels parts.

Attribute Read-only, cannot be set

Return type list of Part

root

The vessels root part.

Attribute Read-only, cannot be set

Return type Part

Note: See the discussion on *Trees of Parts*.

controlling

The part from which the vessel is controlled.

Attribute Can be read or written

Return type Part

with_name (name)

A list of parts whose Part. name is name.

```
Parameters name (str) -
```

Return type list of Part

with_title(title)

A list of all parts whose Part.title is title.

Parameters title (str) -

Return type list of *Part*

with_module (module_name)

A list of all parts that contain a *Module* whose *Module.name* is *module_name*.

Parameters module_name (str)-

Return type list of *Part*

in_stage(stage)

A list of all parts that are activated in the given *stage*.

Parameters stage (int) -

Return type list of *Part*

Note: See the discussion on *Staging*.

in_decouple_stage(stage)

A list of all parts that are decoupled in the given stage.

Parameters stage (int)-

Return type list of Part

Note: See the discussion on *Staging*.

modules_with_name (module_name)

A list of modules (combined across all parts in the vessel) whose Module.name is module_name.

Parameters module_name (str)-

Return type list of Module

decouplers

A list of all decouplers in the vessel.

Attribute Read-only, cannot be set

Return type list of *Decoupler*

docking_ports

A list of all docking ports in the vessel.

Attribute Read-only, cannot be set

Return type list of *DockingPort*

docking_port_with_name (name)

The first docking port in the vessel with the given port name, as returned by <code>DockingPort.name</code>. Returns <code>None</code> if there are no such docking ports.

Parameters name (str) -

Return type DockingPort

engines

A list of all engines in the vessel.

Attribute Read-only, cannot be set

Return type list of *Engine*

landing_gear

A list of all landing gear attached to the vessel.

Attribute Read-only, cannot be set

Return type list of LandingGear

landing_legs

A list of all landing legs attached to the vessel.

Attribute Read-only, cannot be set

Return type list of LandingLeg

launch_clamps

A list of all launch clamps attached to the vessel.

Attribute Read-only, cannot be set

Return type list of LaunchClamp

lights

A list of all lights in the vessel.

Attribute Read-only, cannot be set

Return type list of *Light*

parachutes

A list of all parachutes in the vessel.

Attribute Read-only, cannot be set

Return type list of Parachute

radiators

A list of all radiators in the vessel.

Attribute Read-only, cannot be set

Return type list of Radiator

resource converters

A list of all resource converters in the vessel.

Attribute Read-only, cannot be set

Return type list of ResourceConverter

resource_harvesters

A list of all resource harvesters in the vessel.

Attribute Read-only, cannot be set

Return type list of ResourceHarvester

reaction_wheels

A list of all reaction wheels in the vessel.

```
Attribute Read-only, cannot be set
```

Return type list of ReactionWheel

sensors

A list of all sensors in the vessel.

Attribute Read-only, cannot be set

Return type list of Sensor

solar_panels

A list of all solar panels in the vessel.

Attribute Read-only, cannot be set

Return type list of SolarPanel

Part

class SpaceCenter.Part

Instances of this class represents a part. A vessel is made of multiple parts. Instances can be obtained by various methods in *Parts*.

name

Internal name of the part, as used in part cfg files. For example "Mark1-2Pod".

Attribute Read-only, cannot be set

Return type str

title

Title of the part, as shown when the part is right clicked in-game. For example "Mk1-2 Command Pod".

Attribute Read-only, cannot be set

Return type str

cost

The cost of the part, in units of funds.

Attribute Read-only, cannot be set

Return type float

vessel

The vessel that contains this part.

Attribute Read-only, cannot be set

Return type Vessel

parent

The parts parent. Returns None if the part does not have a parent. This, in combination with <code>Part.children</code>, can be used to traverse the vessels parts tree.

Attribute Read-only, cannot be set

Return type Part

Note: See the discussion on *Trees of Parts*.

children

The parts children. Returns an empty list if the part has no children. This, in combination with <code>Part.parent</code>, can be used to traverse the vessels parts tree.

Attribute Read-only, cannot be set

Return type list of Part

Note: See the discussion on *Trees of Parts*.

axially_attached

Whether the part is axially attached to its parent, i.e. on the top or bottom of its parent. If the part has no parent, returns False.

Attribute Read-only, cannot be set

Return type bool

Note: See the discussion on *Attachment Modes*.

radially_attached

Whether the part is radially attached to its parent, i.e. on the side of its parent. If the part has no parent, returns False.

Attribute Read-only, cannot be set

Return type bool

Note: See the discussion on Attachment Modes.

stage

The stage in which this part will be activated. Returns -1 if the part is not activated by staging.

Attribute Read-only, cannot be set

Return type int

Note: See the discussion on *Staging*.

decouple_stage

The stage in which this part will be decoupled. Returns -1 if the part is never decoupled from the vessel.

Attribute Read-only, cannot be set

Return type int

Note: See the discussion on *Staging*.

massless

Whether the part is massless.

Attribute Read-only, cannot be set

Return type bool

mass

The current mass of the part, including resources it contains, in kilograms. Returns zero if the part is massless.

Attribute Read-only, cannot be set

Return type float

dry_mass

The mass of the part, not including any resources it contains, in kilograms. Returns zero if the part is massless.

Attribute Read-only, cannot be set

Return type float

impact_tolerance

The impact tolerance of the part, in meters per second.

Attribute Read-only, cannot be set

Return type float

temperature

Temperature of the part, in Kelvin.

Attribute Read-only, cannot be set

Return type float

skin_temperature

Temperature of the skin of the part, in Kelvin.

Attribute Read-only, cannot be set

Return type float

max_temperature

Maximum temperature that the part can survive, in Kelvin.

Attribute Read-only, cannot be set

Return type float

max_skin_temperature

Maximum temperature that the skin of the part can survive, in Kelvin.

Attribute Read-only, cannot be set

Return type float

external_temperature

Temperature of the atmosphere/vacuum surrounding the part, in Kelvin. This does not include heating from direct sunlight.

Attribute Read-only, cannot be set

Return type float

thermal_mass

How much it takes to heat up the part.

Attribute Read-only, cannot be set

Return type float

thermal skin mass

How much it takes to heat up the part's skin.

Attribute Read-only, cannot be set

Return type float

thermal_resource_mass

How much it takes to heat up resources in the part.

Attribute Read-only, cannot be set

Return type float

thermal_conduction_flux

The speed that heat is conducting into or out of the part through contact with other parts. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type float

thermal convection flux

The speed that heat is convecting into or out of the part from the surrounding atmosphere. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type float

thermal radiation flux

The speed that heat is radiating into or out of the part from the surrounding vacuum. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type float

thermal internal flux

The speed that heat is generated by the part. For example, engines generate heat by burning fuel. A positive number means the part is gaining heat energy, negative means it is losing heat energy.

Attribute Read-only, cannot be set

Return type float

thermal_skin_to_internal_flux

The speed that heat is conducting between the part's skin and its internals.

Attribute Read-only, cannot be set

Return type float

resources

A Resources object for the part.

Attribute Read-only, cannot be set

Return type Resources

crossfeed

Whether this part is crossfeed capable.

Attribute Read-only, cannot be set

Return type bool

is fuel line

Whether this part is a fuel line.

Attribute Read-only, cannot be set

Return type bool

fuel_lines_from

The parts that are connected to this part via fuel lines, where the direction of the fuel line is into this part.

Attribute Read-only, cannot be set

Return type list of *Part*

Note: See the discussion on *Fuel Lines*.

fuel_lines_to

The parts that are connected to this part via fuel lines, where the direction of the fuel line is out of this part.

Attribute Read-only, cannot be set

Return type list of *Part*

Note: See the discussion on *Fuel Lines*.

modules

The modules for this part.

Attribute Read-only, cannot be set

Return type list of Module

decoupler

A Decoupler if the part is a decoupler, otherwise None.

Attribute Read-only, cannot be set

Return type Decoupler

docking_port

A DockingPort if the part is a docking port, otherwise None.

Attribute Read-only, cannot be set

Return type DockingPort

engine

An Engine if the part is an engine, otherwise None.

Attribute Read-only, cannot be set

Return type Engine

landing_gear

A ${\it LandingGear}$ if the part is a landing gear , otherwise None.

Attribute Read-only, cannot be set

Return type LandingGear

landing_leg

A LandingLeg if the part is a landing leg, otherwise None.

```
Attribute Read-only, cannot be set
```

Return type LandingLeg

launch_clamp

A LaunchClamp if the part is a launch clamp, otherwise None.

Attribute Read-only, cannot be set

Return type LaunchClamp

light

A Light if the part is a light, otherwise None.

Attribute Read-only, cannot be set

Return type Light

parachute

A Parachute if the part is a parachute, otherwise None.

Attribute Read-only, cannot be set

Return type Parachute

radiator

A Radiator if the part is a radiator, otherwise None.

Attribute Read-only, cannot be set

Return type Radiator

reaction wheel

A ReactionWheel if the part is a reaction wheel, otherwise None.

Attribute Read-only, cannot be set

Return type ReactionWheel

resource converter

A ResourceConverter if the part is a resource converter, otherwise None.

Attribute Read-only, cannot be set

Return type ResourceConverter

resource harvester

A ResourceHarvester if the part is a resource harvester, otherwise None.

Attribute Read-only, cannot be set

Return type ResourceHarvester

sensor

A Sensor if the part is a sensor, otherwise None.

Attribute Read-only, cannot be set

Return type Sensor

solar_panel

A SolarPanel if the part is a solar panel, otherwise None.

Attribute Read-only, cannot be set

Return type SolarPanel

```
position (reference_frame)
```

The position of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

direction (reference frame)

The direction of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

velocity (reference_frame)

The velocity of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

rotation (reference_frame)

The rotation of the part in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float, float)

reference_frame

The reference frame that is fixed relative to this part.

- •The origin is at the position of the part.
- •The axes rotate with the part.
- •The x, y and z axis directions depend on the design of the part.

Attribute Read-only, cannot be set

Return type ReferenceFrame

Note: For docking port parts, this reference frame is not necessarily equivalent to the reference frame for the docking port, returned by <code>DockingPort.reference_frame</code>.

Module

class SpaceCenter.Module

In KSP, each part has zero or more PartModules associated with it. Each one contains some of the functionality of the part. For example, an engine has a "ModuleEngines" PartModule that contains all the functionality of an engine. This class allows you to interact with KSPs PartModules, and any PartModules that have been added by other mods.

name

Name of the PartModule. For example, "ModuleEngines".

Attribute Read-only, cannot be set

Return type str

part

The part that contains this module.

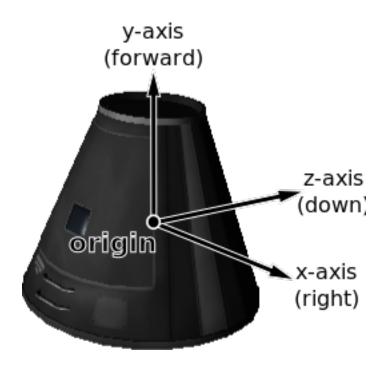


Fig. 7.7: Mk1 Command Pod reference frame origin and axes

Attribute Read-only, cannot be set

Return type Part

fields

The modules field names and their associated values, as a dictionary. These are the values visible in the right-click menu of the part.

Attribute Read-only, cannot be set

Return type dict from str to str

has_field(name)

Returns True if the module has a field with the given name.

Parameters name (str) – Name of the field.

Return type bool

get_field(name)

Returns the value of a field.

Parameters name (str) – Name of the field.

Return type str

events

A list of the names of all of the modules events. Events are the clickable buttons visible in the right-click menu of the part.

Attribute Read-only, cannot be set

Return type list of str

has_event (name)

True if the module has an event with the given name.

```
Parameters name (str) -
```

Return type bool

trigger_event(name)

Trigger the named event. Equivalent to clicking the button in the right-click menu of the part.

```
Parameters name (str) -
```

actions

A list of all the names of the modules actions. These are the parts actions that can be assigned to action groups in the in-game editor.

Attribute Read-only, cannot be set

Return type list of str

has_action (name)

True if the part has an action with the given name.

Parameters name (str) -

Return type bool

```
set_action (name[, value = True])
```

Set the value of an action with the given name.

Parameters

- name (str) -
- value (bool) -

Specific Types of Part

The following classes provide functionality for specific types of part.

- Decoupler
- Docking Port
- Engine
- Landing Gear
- Landing Leg
- Launch Clamp
- Light
- Parachute
- Radiator
- Resource Converter
- Resource Harvester
- Reaction Wheel
- Sensor
- Solar Panel

Decoupler

```
class SpaceCenter.Decoupler
```

Obtained by calling Part.decoupler

part

The part object for this decoupler.

Attribute Read-only, cannot be set

Return type Part

decouple()

Fires the decoupler. Has no effect if the decoupler has already fired.

decoupled

Whether the decoupler has fired.

Attribute Read-only, cannot be set

Return type bool

impulse

The impulse that the decoupler imparts when it is fired, in Newton seconds.

Attribute Read-only, cannot be set

Return type float

Docking Port

class SpaceCenter.DockingPort

Obtained by calling Part.docking_port

part

The part object for this docking port.

Attribute Read-only, cannot be set

Return type Part

name

The port name of the docking port. This is the name of the port that can be set in the right click menu, when the Docking Port Alignment Indicator mod is installed. If this mod is not installed, returns the title of the part (Part.title).

Attribute Can be read or written

Return type str

state

The current state of the docking port.

Attribute Read-only, cannot be set

Return type DockingPortState

docked_part

The part that this docking port is docked to. Returns None if this docking port is not docked to anything.

Attribute Read-only, cannot be set

Return type *Part*

undock()

Undocks the docking port and returns the vessel that was undocked from. After undocking, the active vessel may change (active_vessel). This method can be called for either docking port in a docked pair - both calls will have the same effect. Returns None if the docking port is not docked to anything.

Return type Vessel

reengage_distance

The distance a docking port must move away when it undocks before it becomes ready to dock with another port, in meters.

Attribute Read-only, cannot be set

Return type float

has shield

Whether the docking port has a shield.

Attribute Read-only, cannot be set

Return type bool

shielded

The state of the docking ports shield, if it has one. Returns True if the docking port has a shield, and the shield is closed. Otherwise returns False. When set to True, the shield is closed, and when set to False the shield is opened. If the docking port does not have a shield, setting this attribute has no effect.

Attribute Can be read or written

Return type bool

position (reference_frame)

The position of the docking port in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

direction (reference_frame)

The direction that docking port points in, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float)

rotation (reference_frame)

The rotation of the docking port, in the given reference frame.

Parameters reference_frame (ReferenceFrame) -

Return type tuple of (float, float, float, float)

reference frame

The reference frame that is fixed relative to this docking port, and oriented with the port.

- •The origin is at the position of the docking port.
- •The axes rotate with the docking port.
- •The x-axis points out to the right side of the docking port.
- •The y-axis points in the direction the docking port is facing.
- •The z-axis points out of the bottom off the docking port.

Attribute Read-only, cannot be set

Return type ReferenceFrame

Note: This reference frame is not necessarily equivalent to the reference frame for the part, returned by <code>Part.reference frame</code>.

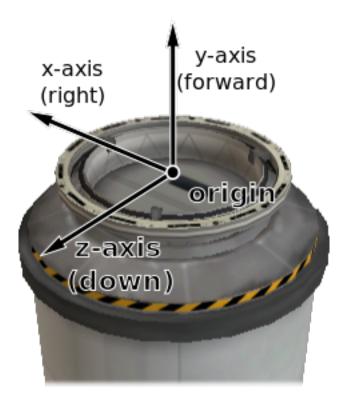


Fig. 7.8: Docking port reference frame origin and axes

class SpaceCenter.DockingPortState

See DockingPort.state.

ready

The docking port is ready to dock to another docking port.

docked

The docking port is docked to another docking port, or docked to another part (from the VAB/SPH).

docking

The docking port is very close to another docking port, but has not docked. It is using magnetic force to acquire a solid dock.

undocking

The docking port has just been undocked from another docking port, and is disabled until it moves away by a sufficient distance (DockingPort.reengage_distance).

shielded

The docking port has a shield, and the shield is closed.

moving

The docking ports shield is currently opening/closing.

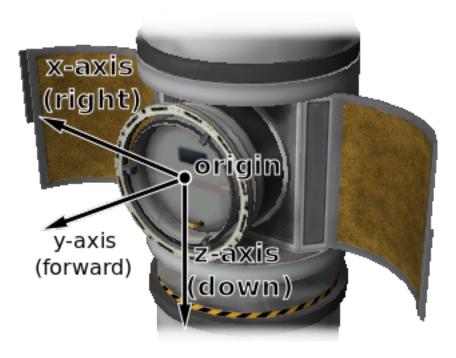


Fig. 7.9: Inline docking port reference frame origin and axes

Engine

class SpaceCenter.Engine

Obtained by calling Part.engine.

part

The part object for this engine.

Attribute Read-only, cannot be set

Return type Part

active

Whether the engine is active. Setting this attribute may have no effect, depending on <code>Engine.can_shutdown</code> and <code>Engine.can_restart</code>.

Attribute Can be read or written

Return type bool

thrust

The current amount of thrust being produced by the engine, in Newtons. Returns zero if the engine is not active or if it has no fuel.

Attribute Read-only, cannot be set

Return type float

available_thrust

The maximum available amount of thrust that can be produced by the engine, in Newtons. This takes *Engine.thrust_limit* into account, and is the amount of thrust produced by the engine when activated and the main throttle is set to 100%. Returns zero if the engine does not have any fuel.

Attribute Read-only, cannot be set

Return type float

max thrust

Gets the maximum amount of thrust that can be produced by the engine, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine.thrust_limit</code> is set to 100% and the main vessel's throttle is set to 100%.

Attribute Read-only, cannot be set

Return type float

max_vacuum_thrust

The maximum amount of thrust that can be produced by the engine in a vacuum, in Newtons. This is the amount of thrust produced by the engine when activated, <code>Engine.thrust_limit</code> is set to 100%, the main vessel's throttle is set to 100% and the engine is in a vacuum.

Attribute Read-only, cannot be set

Return type float

thrust_limit

The thrust limiter of the engine. A value between 0 and 1. Setting this attribute may have no effect, for example the thrust limit for a solid rocket booster cannot be changed in flight.

Attribute Can be read or written

Return type float

specific_impulse

The current specific impulse of the engine, in seconds. Returns zero if the engine is not active.

Attribute Read-only, cannot be set

Return type float

vacuum_specific_impulse

The vacuum specific impulse of the engine, in seconds.

Attribute Read-only, cannot be set

Return type float

kerbin_sea_level_specific_impulse

The specific impulse of the engine at sea level on Kerbin, in seconds.

Attribute Read-only, cannot be set

Return type float

propellants

The names of resources that the engine consumes.

Attribute Read-only, cannot be set

Return type list of str

propellant_ratios

The ratios of resources that the engine consumes. A dictionary mapping resource names to the ratios at which they are consumed by the engine.

Attribute Read-only, cannot be set

Return type dict from str to float

has fuel

Whether the engine has run out of fuel (or flamed out).

Attribute Read-only, cannot be set

Return type bool

throttle

The current throttle setting for the engine. A value between 0 and 1. This is not necessarily the same as the vessel's main throttle setting, as some engines take time to adjust their throttle (such as jet engines).

Attribute Read-only, cannot be set

Return type float

throttle_locked

Whether the <code>Control.throttle</code> affects the engine. For example, this is <code>True</code> for liquid fueled rockets, and <code>False</code> for solid rocket boosters.

Attribute Read-only, cannot be set

Return type bool

can_restart

Whether the engine can be restarted once shutdown. If the engine cannot be shutdown, returns False. For example, this is True for liquid fueled rockets and False for solid rocket boosters.

Attribute Read-only, cannot be set

Return type bool

can shutdown

Gets whether the engine can be shutdown once activated. For example, this is True for liquid fueled rockets and False for solid rocket boosters.

Attribute Read-only, cannot be set

Return type bool

gimballed

Whether the engine nozzle is gimballed, i.e. can provide a turning force.

Attribute Read-only, cannot be set

Return type bool

gimbal_range

The range over which the gimbal can move, in degrees.

Attribute Read-only, cannot be set

Return type float

gimbal locked

Whether the engines gimbal is locked in place. Setting this attribute has no effect if the engine is not gimballed.

Attribute Can be read or written

Return type bool

gimbal_limit

The gimbal limiter of the engine. A value between 0 and 1. Returns 0 if the gimbal is locked or the engine is not gimballed. Setting this attribute has no effect if the engine is not gimballed.

Attribute Can be read or written

Return type float

Landing Gear

class SpaceCenter.LandingGear

```
Obtained by calling Part.landing_gear.
     part
          The part object for this landing gear.
              Attribute Read-only, cannot be set
              Return type Part
     state
          Gets the current state of the landing gear.
              Attribute Read-only, cannot be set
              Return type LandingGearState
     deployed
          Whether the landing gear is deployed.
              Attribute Can be read or written
              Return type bool
class SpaceCenter.LandingGearState
     See LandingGear.state.
     deployed
          Landing gear is fully deployed.
     retracted
          Landing gear is fully retracted.
     deploying
          Landing gear is being deployed.
     retracting
          Landing gear is being retracted.
Landing Leg
class SpaceCenter.LandingLeg
     Obtained by calling Part.landing_leg.
     part
          The part object for this landing leg.
              Attribute Read-only, cannot be set
              Return type Part
     state
          The current state of the landing leg.
              Attribute Read-only, cannot be set
              Return type LandingLegState
     deployed
          Whether the landing leg is deployed.
              Attribute Can be read or written
```

Return type bool

```
class SpaceCenter.LandingLegState
     See LandingLeg.state.
     deployed
          Landing leg is fully deployed.
     retracted
          Landing leg is fully retracted.
     deploying
          Landing leg is being deployed.
     retracting
          Landing leg is being retracted.
     broken
          Landing leg is broken.
     repairing
          Landing leg is being repaired.
Launch Clamp
class SpaceCenter.LaunchClamp
     Obtained by calling Part.launch_clamp.
     part
          The part object for this launch clamp.
              Attribute Read-only, cannot be set
              Return type Part
     release()
          Releases the docking clamp. Has no effect if the clamp has already been released.
Light
class SpaceCenter.Light
     Obtained by calling Part.light.
          The part object for this light.
              Attribute Read-only, cannot be set
              Return type Part
     active
          Whether the light is switched on.
              Attribute Can be read or written
              Return type bool
     power_usage
          The current power usage, in units of charge per second.
```

Attribute Read-only, cannot be set

Return type float

Parachute

class SpaceCenter.Parachute

Obtained by calling Part.parachute.

part

The part object for this parachute.

Attribute Read-only, cannot be set

Return type Part

deploy()

Deploys the parachute. This has no effect if the parachute has already been deployed.

deployed

Whether the parachute has been deployed.

Attribute Read-only, cannot be set

Return type bool

state

The current state of the parachute.

Attribute Read-only, cannot be set

Return type ParachuteState

deploy_altitude

The altitude at which the parachute will full deploy, in meters.

Attribute Can be read or written

Return type float

deploy_min_pressure

The minimum pressure at which the parachute will semi-deploy, in atmospheres.

Attribute Can be read or written

Return type float

class SpaceCenter.ParachuteState

See Parachute.state.

stowed

The parachute is safely tucked away inside its housing.

active

The parachute is still stowed, but ready to semi-deploy.

semi_deployed

The parachute has been deployed and is providing some drag, but is not fully deployed yet.

deployed

The parachute is fully deployed.

cut

The parachute has been cut.

Radiator

part

class SpaceCenter.Radiator

Obtained by calling Part.radiator.

```
The part object for this radiator.
              Attribute Read-only, cannot be set
              Return type Part
     deployed
          Whether the radiator is extended.
               Attribute Can be read or written
              Return type bool
     state
          The current state of the radiator.
              Attribute Read-only, cannot be set
              Return type RadiatorState
class SpaceCenter.RadiatorState
     RadiatorState
     extended
          Radiator is fully extended.
     retracted
          Radiator is fully retracted.
     extending
          Radiator is being extended.
     retracting
          Radiator is being retracted.
     broken
          Radiator is being broken.
Resource Converter
class SpaceCenter.ResourceConverter
     Obtained by calling {\it Part.resource\_converter.}
     part
          The part object for this converter.
              Attribute Read-only, cannot be set
              Return type Part
     count
          The number of converters in the part.
              Attribute Read-only, cannot be set
              Return type int
```

name (index)

```
The name of the specified converter.
               Parameters index (int) – Index of the converter.
               Return type str
     active (index)
          True if the specified converter is active.
               Parameters index (int) – Index of the converter.
               Return type bool
     start (index)
          Start the specified converter.
               Parameters index (int) – Index of the converter.
     stop (index)
          Stop the specified converter.
               Parameters index (int) – Index of the converter.
     state (index)
          The state of the specified converter.
               Parameters index (int) – Index of the converter.
               Return type ResourceConverterState
     status info(index)
          Status information for the specified converter. This is the full status message shown in the in-game UI.
               Parameters index (int) – Index of the converter.
               Return type str
     inputs (index)
          List of the names of resources consumed by the specified converter.
               Parameters index (int) – Index of the converter.
               Return type list of str
     outputs (index)
          List of the names of resources produced by the specified converter.
               Parameters index (int) – Index of the converter.
               Return type list of str
class SpaceCenter.ResourceConverterState
     See ResourceConverter.state().
     running
          Converter is running.
     idle
          Converter is idle.
     missing_resource
          Converter is missing a required resource.
     storage_full
          No available storage for output resource.
```

capacity

At preset resource capacity.

unknown

Unknown state. Possible with modified resource converters. In this case, check ResourceConverter.status_info() for more information.

Resource Harvester

${\bf class} \; {\tt SpaceCenter.ResourceHarvester}$

Obtained by calling Part.resource_harvester.

part

The part object for this harvester.

Attribute Read-only, cannot be set

Return type Part

state

The state of the harvester.

Attribute Read-only, cannot be set

Return type ResourceHarvesterState

deployed

Whether the harvester is deployed.

Attribute Can be read or written

Return type bool

active

Whether the harvester is actively drilling.

Attribute Can be read or written

Return type bool

extraction_rate

The rate at which the drill is extracting ore, in units per second.

Attribute Read-only, cannot be set

Return type float

thermal_efficiency

The thermal efficiency of the drill, as a percentage of its maximum.

Attribute Read-only, cannot be set

Return type float

core_temperature

The core temperature of the drill, in Kelvin.

Attribute Read-only, cannot be set

Return type float

optimum_core_temperature

The core temperature at which the drill will operate with peak efficiency, in Kelvin.

Attribute Read-only, cannot be set

Return type float

Reaction Wheel

```
class SpaceCenter.ReactionWheel
     Obtained by calling Part.reaction_wheel.
     part
          The part object for this reaction wheel.
              Attribute Read-only, cannot be set
              Return type Part
     active
          Whether the reaction wheel is active.
               Attribute Can be read or written
              Return type bool
     broken
          Whether the reaction wheel is broken.
              Attribute Read-only, cannot be set
              Return type bool
     pitch_torque
          The torque in the pitch axis, in Newton meters.
              Attribute Read-only, cannot be set
              Return type float
     yaw_torque
          The torque in the yaw axis, in Newton meters.
              Attribute Read-only, cannot be set
              Return type float
     roll_torque
          The torque in the roll axis, in Newton meters.
              Attribute Read-only, cannot be set
              Return type float
Sensor
class SpaceCenter.Sensor
     Obtained by calling Part.sensor.
     part
          The part object for this sensor.
              Attribute Read-only, cannot be set
              Return type Part
     active
          Whether the sensor is active.
```

```
Attribute Can be read or written
```

Return type bool

value

The current value of the sensor.

Attribute Read-only, cannot be set

Return type str

power_usage

The current power usage of the sensor, in units of charge per second.

Attribute Read-only, cannot be set

Return type float

Solar Panel

class SpaceCenter.SolarPanel

Obtained by calling Part.solar_panel.

part

The part object for this solar panel.

Attribute Read-only, cannot be set

Return type Part

deployed

Whether the solar panel is extended.

Attribute Can be read or written

Return type bool

state

The current state of the solar panel.

Attribute Read-only, cannot be set

Return type SolarPanelState

energy flow

The current amount of energy being generated by the solar panel, in units of charge per second.

Attribute Read-only, cannot be set

Return type float

sun_exposure

The current amount of sunlight that is incident on the solar panel, as a percentage. A value between 0 and 1.

Attribute Read-only, cannot be set

Return type float

class SpaceCenter.SolarPanelState

See SolarPanel.state.

extended

Solar panel is fully extended.

retracted

Solar panel is fully retracted.

extending

Solar panel is being extended.

retracting

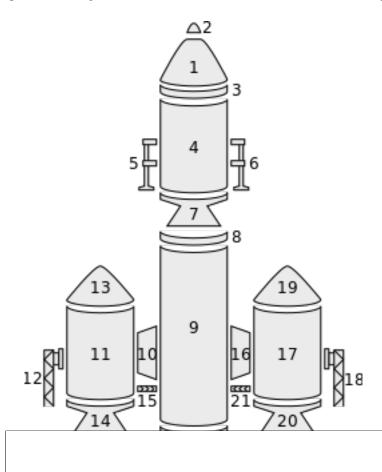
Solar panel is being retracted.

broken

Solar panel is broken.

Trees of Parts

Vessels in KSP comprised are of number of parts, connected to one another in a 1, and the corresponding tree of in Figure tree structure. An example vessel is shown craft file can also be downloaded here. parts in Figure 2. The this example for



Traversing the Tree

The tree of parts can be traversed using the attributes SpaceCenter.Parts.root, SpaceCenter.Part.parent and SpaceCenter.Part.children.

The root of the tree is the same as the vessels root part (part number 1 in the example above) and can be obtained by calling SpaceCenter.Parts.root. A parts children can be obtained by calling SpaceCenter.Part.children. If the part does not have any children, SpaceCenter.Part.children returns an empty list. A parts parent can be obtained by calling SpaceCenter.Part.parent. If the part does not have a parent (as is the case for the root part), SpaceCenter.Part.parent returns None.

The following Python example uses these attributes to perform a depth-first traversal over all of the parts in a vessel:

```
root = vessel.parts.root
stack = [(root, 0)]
while len(stack) > 0:
   part,depth = stack.pop()
   print(' '*depth, part.title)
   for child in part.children:
        stack.append((child, depth+1))
```

When this code is execute using the craft file for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1
TR-18A Stack Decoupler
 FL-T400 Fuel Tank
  LV-909 Liquid Fuel Engine
   TR-18A Stack Decoupler
    FL-T800 Fuel Tank
     LV-909 Liquid Fuel Engine
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
        TT18-A Launch Stability Enhancer
        FTX-2 External Fuel Duct
       LV-909 Liquid Fuel Engine
       Aerodynamic Nose Cone
     TT-70 Radial Decoupler
      FL-T400 Fuel Tank
        TT18-A Launch Stability Enhancer
        FTX-2 External Fuel Duct
        LV-909 Liquid Fuel Engine
        Aerodynamic Nose Cone
  LT-1 Landing Struts
  LT-1 Landing Struts
Mk16 Parachute
```

Attachment Modes

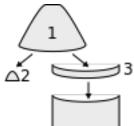
Parts can be attached to other parts either *radially* (on the side of the parent part) or *axially* (on the end of the parent part, to form a stack).

For example, in the vessel pictured above, the parachute (part 2) is *axially* connected to its

parent (the command pod – part 1), and the landing leg (part 5) is *radially* connected to its parent (the fuel tank – part 4).

The root part of a vessel (for example the command pod – part 1) does not have a parent part, so does not have an attachment mode. However, the part is consider to be *axially* attached to nothing.

The following Python example does a depthfirst traversal as before, but also prints out the attachment mode used by the part:



```
root = vessel.parts.root
stack = [(root, 0)]
while len(stack) > 0:
    part,depth = stack.pop()
    if part.axially_attached:
        attach_mode = 'axial'
    else: # radially_attached
        attach_mode = 'radial'
    print(' '*depth, part.title, '-', attach_mode)
    for child in part.children:
        stack.append((child, depth+1))
```

When this code is execute using the craft file

for the example vessel pictured above, the following is printed out:

```
Command Pod Mk1 - axial
TR-18A Stack Decoupler - axial
 FL-T400 Fuel Tank - axial
  LV-909 Liquid Fuel Engine - axial
   TR-18A Stack Decoupler - axial
    FL-T800 Fuel Tank - axial
     LV-909 Liquid Fuel Engine - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
     TT-70 Radial Decoupler - radial
      FL-T400 Fuel Tank - radial
       TT18-A Launch Stability Enhancer - radial
       FTX-2 External Fuel Duct - radial
       LV-909 Liquid Fuel Engine - axial
       Aerodynamic Nose Cone - axial
  LT-1 Landing Struts - radial
  LT-1 Landing Struts - radial
Mk16 Parachute - axial
```

Fuel Lines

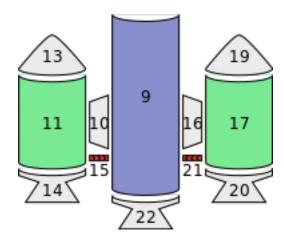
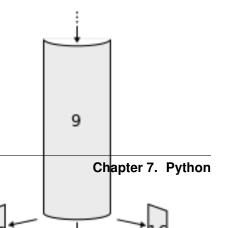


Fig. 7.12: **Figure 5** – Fuel lines from the example in Figure 1. Fuel flows from the parts highlighted in green, into the part highlighted in blue.

Fuel lines are considered parts, and are included in the parts tree (for example, as pictured in Figure 4). However, the parts tree does not contain information about which parts fuel lines connect to. The parent part of a fuel line is the part from which it will take fuel (as shown in Figure 4) however the part that it will send fuel to is not represented in the parts tree.

Figure 5 shows the fuel lines from the example vessel pictured earlier. Fuel line part 15 (in red) takes fuel from a fuel tank (part



11 – in green) and feeds it into another fuel tank (part 9 – in blue). The fuel line is therefore a child of part 11, but its connection to part 9 is not represented in the tree.

The attributes <code>SpaceCenter.Part.fuel_lines_from</code> and <code>SpaceCenter.Part.fuel_lines_to</code> can be used to discover these connections. In the example in Figure 5, when <code>SpaceCenter.Part.fuel_lines_to</code> is called on fuel tank part 11, it will return a list of parts containing just fuel tank part 9 (the blue part). When <code>SpaceCenter.Part.fuel_lines_from</code> is called on fuel tank part 9, it will return a list containing fuel tank parts 11 and 17 (the parts colored green).

Staging

Each part has two staging numbers associated with it: the stage in which the part is activated and the stage in which the part is decoupled. These values can be obtained using SpaceCenter.Part.stage and SpaceCenter.Part.decouple_stage respectively. For parts that are not activated by staging, SpaceCenter.Part.stage returns -1. For parts that are never decoupled, SpaceCenter.Part.decouple_stage returns a value of -1.

Figure 6 shows an example staging sequence for a vessel. Figure 7 shows the stages in which each part of the vessel will be *activated*. Figure 8 shows the stages in which each part of the vessel will be *decoupled*.

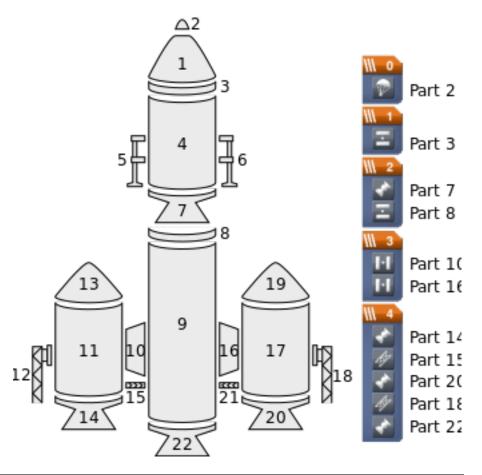


Fig. 7.14: Figure 6 – Example vessel from Figure 1 with a staging sequence 361

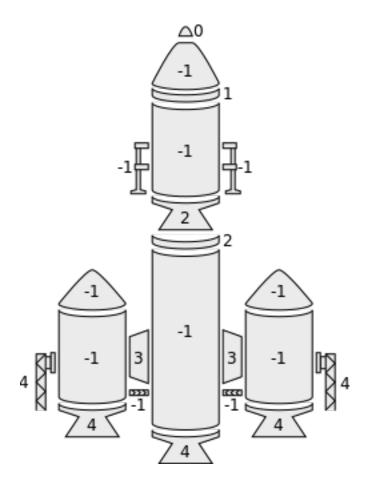
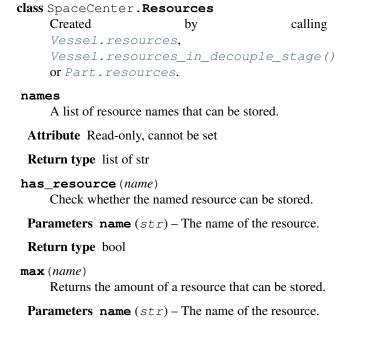
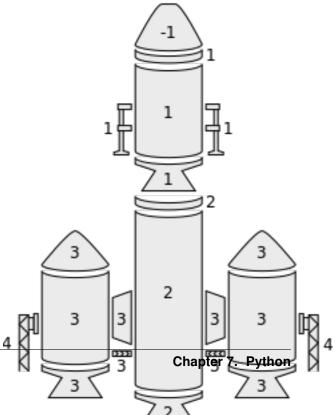


Fig. 7.15: **Figure 7** – The stage in which each part is *activated*.

7.2.8 Resources

362





Δ-1

Return type float

amount (name)

Returns the amount of a resource that is currently stored.

Parameters name (str) – The name of the resource.

Return type float

static density (name)

Returns the density of a resource, in kg/l.

Parameters name (str) – The name of the resource.

Return type float

static flow_mode (name)

Returns the flow mode of a resource.

Parameters name (str) – The name of the resource.

Return type ResourceFlowMode

class SpaceCenter.ResourceFlowMode

See Resources.flow_mode().

vessel

The resource flows to any part in the vessel. For example, electric charge.

stage

The resource flows from parts in the first stage, followed by the second, and so on. For example, mono-propellant.

adjacent

The resource flows between adjacent parts within the vessel. For example, liquid fuel or oxidizer.

none

The resource does not flow. For example, solid fuel.

7.2.9 Node

class SpaceCenter.Node

Represents a maneuver node. Can be created using Control.add_node().

prograde

The magnitude of the maneuver nodes delta-v in the prograde direction, in meters per second.

Attribute Can be read or written

Return type float

normal

The magnitude of the maneuver nodes delta-v in the normal direction, in meters per second.

Attribute Can be read or written

Return type float

radial

The magnitude of the maneuver nodes delta-v in the radial direction, in meters per second.

Attribute Can be read or written

Return type float

delta v

The delta-v of the maneuver node, in meters per second.

Attribute Can be read or written

Return type float

Note: Does not change when executing the maneuver node. See *Node.remaining_delta_v*.

remaining_delta_v

Gets the remaining delta-v of the maneuver node, in meters per second. Changes as the node is executed. This is equivalent to the delta-v reported in-game.

Attribute Read-only, cannot be set

Return type float

```
burn_vector([reference_frame = None])
```

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s.

Parameters reference_frame

(ReferenceFrame) -

Return type tuple of (float, float, float)

Note: Does not change when executing the maneuver node. See Node.remaining_burn_vector().

remaining_burn_vector([reference_frame = None])

Returns a vector whose direction the direction of the maneuver node burn, and whose magnitude is the delta-v of the burn in m/s. The direction and magnitude change as the burn is executed.

Parameters reference_frame

(ReferenceFrame) -

Return type tuple of (float, float, float)

ut

The universal time at which the maneuver will occur, in seconds.

Attribute Can be read or written

Return type float

time_to

The time until the maneuver node will be encountered, in seconds.

Attribute Read-only, cannot be set

Return type float

orbit

The orbit that results from executing the maneuver node.

Attribute Read-only, cannot be set

Return type Orbit

remove()

Removes the maneuver node.

reference_frame

Gets the reference frame that is fixed relative to the maneuver node's burn.

- •The origin is at the position of the maneuver node.
- •The y-axis points in the direction of the burn.
- •The x-axis and z-axis point in arbitrary but fixed directions.

Attribute Read-only, cannot be set

Return type ReferenceFrame

orbital_reference_frame

Gets the reference frame that is fixed relative to the maneuver node, and orientated with the orbital prograde/normal/radial directions of the original orbit at the maneuver node's position.

- •The origin is at the position of the maneuver node.
- •The x-axis points in the orbital anti-radial direction of the original orbit, at the position of the maneuver node.
- •The y-axis points in the orbital prograde direction of the original orbit, at the position of the maneuver node
- •The z-axis points in the orbital normal direction of the original orbit, at the position of the maneuver node.

Attribute Read-only, cannot be set

Return type ReferenceFrame

position (reference_frame)

Returns the position vector of the maneuver node in the given reference frame.

Parameters reference_frame

(ReferenceFrame) -

Return type tuple of (float, float, float)

direction (reference_frame)

Returns the unit direction vector of the maneuver nodes burn in the given reference frame.

Parameters reference_frame

(ReferenceFrame) -

Return type tuple of (float, float, float)

7.2.10 Comms

class SpaceCenter.Comms

Used to interact with RemoteTech. Created using a call to Vessel.comms.

Note: This class requires RemoteTech to be installed.

has_local_control

Whether the vessel can be controlled locally.

Attribute Read-only, cannot be set

Return type bool

has_flight_computer

Whether the vessel has a RemoteTech flight computer on board.

Attribute Read-only, cannot be set

Return type bool

has_connection

Whether the vessel can receive commands from the KSC or a command station.

Attribute Read-only, cannot be set

Return type bool

has_connection_to_ground_station

Whether the vessel can transmit science data to a ground station.

Attribute Read-only, cannot be set

Return type bool

signal_delay

The signal delay when sending commands to the vessel, in seconds.

Attribute Read-only, cannot be set

Return type float

signal_delay_to_ground_station

The signal delay between the vessel and the closest ground station, in seconds.

Attribute Read-only, cannot be set

Return type float

signal_delay_to_vessel (other)

Returns the signal delay between the current vessel and another vessel, in seconds.

Parameters other (Vessel) -

Return type float

7.2.11 ReferenceFrame

class SpaceCenter.ReferenceFrame

Represents a reference frame for positions, rotations and velocities. Contains:

- •The position of the origin.
- •The directions of the x, y and z axes.
- •The linear velocity of the frame.
- •The angular velocity of the frame.

Note: This class does not contain any properties or methods. It is only used as a parameter to other functions.

7.2.12 AutoPilot

class SpaceCenter.AutoPilot

Provides basic auto-piloting utilities for a vessel. Created by calling Vessel.auto_pilot.

engage()

Engage the auto-pilot.

disengage()

Disengage the auto-pilot.

wait()

Blocks until the vessel is pointing in the target direction (if set) and has the target roll (if set).

error

The error, in degrees, between the direction the ship has been asked to point in and the direction it is pointing in. Returns zero if the auto-pilot has not been engaged, SAS is not enabled, SAS is in stability assist mode, or no target direction is set.

Attribute Read-only, cannot be set

Return type float

roll_error

The error, in degrees, between the roll the ship has been asked to be in and the actual roll. Returns zero if the auto-pilot has not been engaged or no target roll is set.

Attribute Read-only, cannot be set

Return type float

reference_frame

The reference frame for the target direction (AutoPilot.target_direction).

Attribute Can be read or written

Return type ReferenceFrame

target_direction

The target direction. None if no target direction is set

Attribute Can be read or written

Return type tuple of (float, float, float)

target_pitch_and_heading(pitch, heading)

Set (AutoPilot.target_direction) from a pitch and heading angle.

Parameters

- pitch (float) Target pitch angle, in degrees between -90° and +90°.
- heading (float) Target heading angle, in degrees between 0° and 360°.

target_roll

The target roll, in degrees. NaN if no target roll is set.

Attribute Can be read or written

Return type float

sas

The state of SAS.

Attribute Can be read or written

Return type bool

Note: Equivalent to Control.sas

sas mode

The current SASMode. These modes are equivalent to the mode buttons to the left of the navball that appear when SAS is enabled.

Attribute Can be read or written

Return type SASMode

Note: Equivalent to Control.sas_mode

rotation_speed_multiplier

Target rotation speed multiplier. Defaults to 1.

Attribute Can be read or written

Return type float

max_rotation_speed

Maximum target rotation speed. Defaults to 1.

Attribute Can be read or written

Return type float

roll_speed_multiplier

Target roll speed multiplier. Defaults to 1.

Attribute Can be read or written

Return type float

max_roll_speed

Maximum target roll speed. Defaults to 1.

Attribute Can be read or written

Return type float

$\verb|set_pid_parameters| ([kp = 1.0][, ki = 0.0][, kd = 0.0])$

Sets the gains for the rotation rate PID controller.

Parameters

- **kp** (float) Proportional gain.
- **ki** (float) Integral gain.
- **kd** (*float*) Derivative gain.

7.2.13 Geometry Types

class SpaceCenter.Vector3

3-dimensional vectors are represented as a 3-tuple.

For example:

```
import krpc
conn = krpc.connect()
v = conn.space_center.active_vessel.flight().prograde
print(v[0], v[1], v[2])
```

class SpaceCenter.Quaternion

Quaternions (rotations in 3-dimensional space) are encoded as a 4-tuple containing the x, y, z and w components. For example:

```
import krpc
conn = krpc.connect()
q = conn.space_center.active_vessel.flight().rotation
print(q[0], q[1], q[2], q[3])
```

7.3 InfernalRobotics API

Provides RPCs to interact with the InfernalRobotics mod. Provides the following classes:

7.3.1 InfernalRobotics

class InfernalRobotics.InfernalRobotics

This service provides functionality to interact with the InfernalRobotics mod.

servo_groups

A list of all the servo groups in the active vessel.

Attribute Read-only, cannot be set

Return type list of ControlGroup

static servo_group_with_name (name)

Returns the servo group with the given *name* or None if none exists. If multiple servo groups have the same name, only one of them is returned.

Parameters name (str) - Name of servo group to find.

Return type ControlGroup

static servo with name(name)

Returns the servo with the given *name*, from all servo groups, or None if none exists. If multiple servos have the same name, only one of them is returned.

Parameters name (str) – Name of the servo to find.

Return type Servo

7.3.2 ControlGroup

class InfernalRobotics.ControlGroup

A group of servos, obtained by calling servo_groups or servo_group_with_name(). Represents the "Servo Groups" in the InfernalRobotics UI.

name

The name of the group.

Attribute Can be read or written

Return type str

forward_key

The key assigned to be the "forward" key for the group.

Attribute Can be read or written

Return type str

reverse_key

The key assigned to be the "reverse" key for the group.

Attribute Can be read or written

Return type str

speed

The speed multiplier for the group.

Attribute Can be read or written

Return type float

expanded

Whether the group is expanded in the Infernal-Robotics UI.

Attribute Can be read or written

Return type bool

servos

The servos that are in the group.

Attribute Read-only, cannot be set

Return type list of Servo

servo_with_name(name)

Returns the servo with the given *name* from this group, or None if none exists.

Parameters name (str) – Name of servo to find.

Return type Servo

move_right()

Moves all of the servos in the group to the right.

move_left()

Moves all of the servos in the group to the left.

move_center()

Moves all of the servos in the group to the center.

move_next_preset()

Moves all of the servos in the group to the next preset.

move_prev_preset()

Moves all of the servos in the group to the previous preset.

stop()

Stops the servos in the group.

7.3.3 Servo

class InfernalRobotics.Servo

Represents a servo. Obtained using ${\it ControlGroup.servos}, {\it ControlGroup.servo_with_name()}$ or ${\it servo_with_name()}.$

name

The name of the servo.

Attribute Can be read or written

Return type str

highlight

Whether the servo should be highlighted in-game.

Attribute Write-only, cannot be read

Return type bool

position

The position of the servo.

Attribute Read-only, cannot be set

Return type float

min_config_position

The minimum position of the servo, specified by the part configuration.

Attribute Read-only, cannot be set

Return type float

max_config_position

The maximum position of the servo, specified by the part configuration.

Attribute Read-only, cannot be set

Return type float

min_position

The minimum position of the servo, specified by the in-game tweak menu.

Attribute Can be read or written

Return type float

max_position

The maximum position of the servo, specified by the in-game tweak menu.

Attribute Can be read or written

Return type float

config_speed

The speed multiplier of the servo, specified by the part configuration.

Attribute Read-only, cannot be set

Return type float

speed

The speed multiplier of the servo, specified by the in-game tweak menu.

Attribute Can be read or written

Return type float

current_speed

The current speed at which the servo is moving.

Attribute Can be read or written

Return type float

acceleration

The current speed multiplier set in the UI.

Attribute Can be read or written

Return type float

is_moving

Whether the servo is moving.

Attribute Read-only, cannot be set

Return type bool

is_free_moving

Whether the servo is freely moving.

Attribute Read-only, cannot be set

Return type bool

is locked

Whether the servo is locked.

Attribute Can be read or written

Return type bool

is_axis_inverted

Whether the servos axis is inverted.

Attribute Can be read or written

Return type bool

move_right()

Moves the servo to the right.

move_left()

Moves the servo to the left.

Moves the servo to the next preset.

```
move_prev_preset()
```

Moves the servo to the previous preset.

```
move_to (position, speed)
```

Moves the servo to *position* and sets the speed multiplier to *speed*.

Parameters

- **position** (*float*) The position to move the serve to.
- **speed** (*float*) Speed multiplier for the movement.

```
stop()
```

Stops the servo.

7.3.4 Example

The following example gets the control group named "MyGroup", prints out the names and positions of all of the servos in the group, then moves all of the servos to the right for 1 second.

```
import krpc, time
conn = krpc.connect(name='InfernalRobotics Example')

group = conn.infernal_robotics.servo_group_with_name('MyGroup')

if group is None:
    print('Group not found')
    exit(1)

for servo in group.servos:
    print servo.name, servo.position

group.move_right()
time.sleep(1)
group.stop()
```

7.4 Kerbal Alarm Clock API

Provides RPCs to interact with the Kerbal Alarm Clock mod. Provides the following classes:

7.4.1 KerbalAlarmClock

class KerbalAlarmClock.KerbalAlarmClock

This service provides functionality to interact with

the Kerbal Alarm Clock mod.

alarms

A list of all the alarms.

Attribute Read-only, cannot be set

Return type list of Alarm

static alarm_with_name (name)

Get the alarm with the given *name*, or None if no alarms have that name. If more than one alarm has the name, only returns one of them.

Parameters name (str) – Name of the alarm to search for.

Return type Alarm

static alarms_with_type (type)

Get a list of alarms of the specified *type*.

Parameters type (AlarmType) - Type of alarm to return.

Return type list of Alarm

static create_alarm(type, name, ut)

Create a new alarm and return it.

Parameters

- type (AlarmType) Type of the new alarm.
- name (str) Name of the new alarm.
- ut (float) Time at which the new alarm should trigger.

Return type Alarm

7.4.2 Alarm

class KerbalAlarmClock.Alarm

Represents an alarm. Obtained by calling alarms, alarm_with_name() or alarms_with_type().

action

The action that the alarm triggers.

Attribute Can be read or written

Return type AlarmAction

margin

The number of seconds before the event that the alarm will fire.

Attribute Can be read or written

Return type float

time

The time at which the alarm will fire.

Attribute Can be read or written

Return type float

type

The type of the alarm.

Attribute Read-only, cannot be set

Return type AlarmType

id

The unique identifier for the alarm.

Attribute Read-only, cannot be set

Return type str

name

The short name of the alarm.

Attribute Can be read or written

Return type str

notes

The long description of the alarm.

Attribute Can be read or written

Return type str

remaining

The number of seconds until the alarm will fire.

Attribute Read-only, cannot be set

Return type float

repeat

Whether the alarm will be repeated after it has fired.

Attribute Can be read or written

Return type bool

repeat_period

The time delay to automatically create an alarm after it has fired.

Attribute Can be read or written

Return type float

vessel

The vessel that the alarm is attached to.

Attribute Can be read or written

Return type SpaceCenter. Vessel

xfer_origin_body

The celestial body the vessel is departing from.

Attribute Can be read or written

Return type SpaceCenter.CelestialBody

xfer_target_body

The celestial body the vessel is arriving at.

Attribute Can be read or written

Return type SpaceCenter.CelestialBody

remove()

Removes the alarm.

7.4.3 AlarmType

class KerbalAlarmClock.AlarmType

The type of an alarm.

raw

An alarm for a specific date/time or a specific period in the future.

maneuver

An alarm based on the next maneuver node on the current ships flight path. This node will be stored and can be restored when you come back to the ship.

maneuver_auto

See AlarmType.maneuver.

apoapsis

An alarm for furthest part of the orbit from the planet.

periapsis

An alarm for nearest part of the orbit from the planet.

ascending_node

Ascending node for the targeted object, or equatorial ascending node.

descending node

Descending node for the targeted object, or equatorial descending node.

closest

An alarm based on the closest approach of this vessel to the targeted vessel, some number of orbits into the future.

contract

An alarm based on the expiry or deadline of contracts in career modes.

contract_auto

See AlarmType.contract.

crew

An alarm that is attached to a crew member.

distance

An alarm that is triggered when a selected target comes within a chosen distance.

earth time

An alarm based on the time in the "Earth" alternative Universe (aka the Real World).

launch rendevous

An alarm that fires as your landed craft passes under the orbit of your target.

soi_change

An alarm manually based on when the next SOI point is on the flight path or set to continually monitor the active flight path and add alarms as it detects SOI changes.

soi_change_auto

See AlarmType.soi_change.

transfer

An alarm based on Interplanetary Transfer Phase Angles, i.e. when should I launch to planet X? Based on Kosmo Not's post and used in Olex's Calculator.

transfer_modelled

See AlarmType.transfer.

7.4.4 AlarmAction

class KerbalAlarmClock.AlarmAction

The action performed by an alarm when it fires.

do_nothing

Don't do anything at all...

do_nothing_delete_when_passed

Don't do anything, and delete the alarm.

kill_warp

Drop out of time warp.

kill_warp_only

Drop out of time warp.

message_only

Display a message.

pause_game

Pause the game.

7.4.5 Example

The following example creates a new alarm for the active vessel. The alarm is set to trigger after 10 seconds have passed, and display a message.

CHAPTER

EIGHT

COMPILING KRPC

kRPC uses the Bazel build system.

Bazel automatically downloads most of the required dependencies to build kRPC. However the following will need to be installed on your system:

- Mono C# compiler and runtime
- Python, including virtualenv and pip
- pdflatex
- RSVG for converting SVGs to PNGs
- libxml, libxslt and python development (for building Java documentation)

To install these dependencies on Ubuntu, follow the instructions on the Mono website and install the required packages using: sudo apt-get install mono-complete python-virtualenv python-pip texlive-latex-base texlive-latex-recommended texlive-fonts-recommended texlive-latex-extra librsvg2-bin libxml2-dev libxslt1-dev python-dev

Before building kRPC you need to make lib/ksp point to a directory containing Kerbal Space Program. For example on Linux, if your KSP directory is at /path/to/ksp and your kRPC source tree at /path/to/krpc you can create a symlink using: ln -s /path/to/ksp /path/to/krpc/lib/ksp

You may also need to modify the symlink at lib/mono-4.5 to point to the correct location of your Mono installation

To build kRPC, run bazel build //:krpc. The resulting archive containing the GameData directory, client libraries etc. will be placed in the bazel-out directory.

8.1 Using an IDE

A C# solution file is provided in the root of the project for use with MonoDevelop or a similar C# IDE.

Some of the C# source files are generated by the Bazel build scripts. You need to run bazel build //:csproj to generate these files before the C# solution file and C# project files can be used.

8.2 Running the Tests

The tests can be run using bazel test //:test. C# NUnit tests can be also run from an IDE using the C# solution file.

Bazel automatically downloads most of the required dependencies to run the tests, however you will also need to install Lua and LuaRocks on your system. These can be install on Ubuntu using: sudo apt-get install lua5.1 luarocks

CHAPTER

NINE

EXTENDING KRPC

9.1 The kRPC Architecture

kRPC consists of two components: a server and a client. The server plugin (provided by KRPC.dll) runs inside KSP. It provides a collection of *procedures* that clients can run. These procedures are arranged in groups called *services* to keep things organized. It also provides an in-game user interface that can be used to start/stop the server, change settings and monitor active clients.

Clients run outside of KSP. This gives you the freedom to run scripts in whatever environment you want. A client communicates with the server to run procedures. kRPC comes with several client libraries that implement the *communication protocol*, making it easy to write programs for these languages that can talk to the server.

kRPC comes with a collection of standard functionality for interacting with vessels, contained in a service called SpaceCenter. This service provides procedures for things like getting flight/orbital data and controlling the active vessel. This service is provided by KRPC. SpaceCenter.dll.

9.2 Service API

Third party mods can add functionality to kRPC using the *Service API*. This is done by adding *attributes* to your own classes, methods and properties to make them visible through the server. When the kRPC server starts, it scans all the assemblies loaded by the game, looking for classes, methods and properties with these attributes.

The following example implements a service that can control the throttle and staging of the active vessel. To add this to the server, compile the code and place the DLL in your GameData directory.

```
Staging.ActivateNextStage ();
}
}
```

The following example shows how this service can then be used from a python client:

```
import krpc
conn = krpc.connect()
conn.launch_control.throttle = 1
conn.launch_control.activate_stage()
```

Some of the client libraries automatically pick up changes to the functionality provided by the server, including the Python and Lua clients. However, some clients require stub code to be generated from the service assembly so that they can interact with new or changed functionality. See *clientgen* for details on how to generate these stubs.

9.2.1 Attributes

The following C# attributes can be used to add functionality to the kRPC server.

```
class KRPCService ([Name][, GameScene])
```

Parameters

- Name (string) Optional name for the service. If omitted, the service name is set to the name of the class this attribute is applied to.
- GameScene (GameScene) The game scenes in which the services procedures are available.

This attribute is applied to a static class, to indicate that all methods, properties and classes declared within it are part of the same service. The name of the service is set to the name of the class, or – if present – the Name parameter.

Multiple services with the same name can be declared, as long the classes, procedures and methods they contain have unique names. The classes will be merged to appear as a single service on the server.

The type to which this attribute is applied must satisfy the following criteria:

- •The type must be a class.
- •The class must be public static.
- •The name of the class, or the Name parameter if specified, must be a valid *kRPC identifier*.
- •The class must not be declared within another class that has the *KRPCService* attribute. Nesting of services is not permitted.

Services are configured to be available in specific *game scenes* via the GameScene parameter. If the GameScene parameter is not specified, the service is available in any scene. If a procedure is called when the service is not available, it will throw an exception.

Examples

•Declare a service called EVA:

```
[KRPCService]
public static class EVA {
     ...
}
```

•Declare a service called MyEVAService (different to the name of the class):

```
[KRPCService (Name = "MyEVAService")]
public static class EVA {
    ...
}
```

•Declare a service called FlightTools that is only available during the Flight game scene:

```
[KRPCService (GameScene = GameScene.Flight)]
public static class FlightTools {
    ...
}
```

class KRPCProcedure

This attribute is applied to static methods, to add them to the server as procedures.

The method to which this attribute is applied must satisfy the following criteria:

- •The method must be public static.
- •The name of the method must be a valid *kRPC identifier*.
- •The method must be declared inside a class that is a KRPCService.
- •The parameter types and return type must be types that kRPC knows how to serialize.
- •Parameters can have default arguments.

Example

The following defines a service called EVA with a PlantFlag procedure that takes a name and an optional description, and returns a Flag object.

```
[KRPCService]
public static class EVA {
    [KRPCProcedure]
    public static Flag PlantFlag (string name, string description = "")
    {
        ...
    }
}
```

This can be called from a python client as follows:

```
import krpc
conn = krpc.connect()
flag = conn.eva.plant_flag('Landing Site', 'One small step for Kerbal-kind')
```

class KRPCClass ([Service])

Parameters Service (*string*) – Optional name of the service to add this class to. If omitted, the class is added to the service that contains its definition.

This attribute is applied to non-static classes. It adds the class to the server, so that references to instances of the class can be passed between client and server.

A KRPCClass must be part of a service, just like a KRPCProcedure. However, it would be restrictive if the class had to be declared as a nested class inside a class with the KRPCService attribute. Therefore, a KRPCClass can be declared outside of any service if it has the Service parameter set to the name of the service that it is part of. Also, the service that the Service parameter refers to does not have to exist. If it does not exist, a service with the given name is created.

The class to which this attribute is applied must satisfy the following criteria:

9.2. Service API 385

- •The class must be public and *not* static.
- •The name of the class must be a valid *kRPC identifier*.
- •The class must either be declared inside a class that is a *KRPCService*, or have its Service parameter set to the name of the service it is part of.

Examples

•Declare a class called Flag in the EVA service:

```
[KRPCService]
public static class EVA {
    [KRPCClass]
    public class Flag {
         ...
    }
}
```

•Declare a class called Flag, without nesting the class definition in a service class:

```
[KRPCClass (Service = "EVA")]
public class Flag {
    ...
}
```

class KRPCMethod

This attribute is applied to methods inside a KRPCClass. This allows a client to call methods on an instance, or static methods in the class.

The method to which this attribute is applied must satisfy the following criteria:

- •The method must be public.
- •The name of the method must be a valid *kRPC identifier*.
- •The method must be declared in a KRPCClass.
- •The parameter types and return type must be types that kRPC can serialize.
- •Parameters can have default arguments.

Example

Declare a Remove method in the Flag class:

```
[KRPCClass (Service = "EVA")]
public class Flag {
    [KRPCMethod]
    void Remove()
    {
        ...
    }
}
```

class KRPCProperty

This attribute is applied to class properties, and comes in two flavors:

- 1.Applied to static properties in a *KRPCService*. In this case, the property must satisfy the following criteria:
 - •Must be public static and have at least one publicly accessible getter or setter.
 - •The name of the property must be a valid *kRPC identifier*.

- •Must be declared inside a KRPCService.
- 2.Applied to non-static properties in a *KRPCClass*. In this case, the property must satisfy the following criteria:
 - •Must be public and not static, and have at least one publicly accessible getter or setter.
 - •The name of the property must be a valid *kRPC identifier*.
 - •Must be declared inside a KRPCClass.

Examples

•Applied to a static property in a service:

```
[KRPCService]
public static class EVA {
    [KRPCProperty]
    public Flag LastFlag
    {
       get { ... }
    }
}
```

This property can be accessed from a python client as follows:

```
import krpc
conn = krpc.connect()
flag = conn.eva.last_flag
```

•Applied to a non-static property in a class:

```
[KRPCClass (Service = "EVA")]
public class Flag {
    [KRPCProperty]
    public void Name { get; set; }

    [KRPCProperty]
    public void Description { get; set; }
}
```

class KRPCEnum ([Service])

Parameters Service (string) – Optional name of the service to add this enum to. If omitted, the enum is added to the service that contains its definition.

This attribute is applied to enumeration types. It adds the enumeration and its permissible values to the server. This attribute works similarly to *KRPCClass*, but is applied to enumeration types.

A KRPCEnum must be part of a service, just like a KRPCClass. Similarly, a KRPCEnum can be declared outside of a service if it has its Service parameter set to the name of the service that it is part of.

The enumeration type to which this attribute is applied must satisfy the following criteria:

- •The enumeration must be public.
- •The name of the enumeration must be a valid *kRPC identifier*.
- •The enumeration must either be declared inside a *KRPCService*, or have it's Service parameter set to the name of the service it is part of.
- •The underlying C# type must be an int.

Examples

9.2. Service API 387

•Declare an enumeration type with two values:

```
[KRPCEnum (Service = "EVA")]
public enum FlagState {
    Raised,
    Lowered
}
```

This can be used from a python client as follows:

```
import krpc
conn = krpc.connect()
state = conn.eva.FlagState.lowered
```

9.2.2 Identifiers

An identifier must only contain alphanumeric characters and underscores. An identifier must not start with an underscore. Identifiers should follow CamelCase capitalization conventions.

Note: Although underscores are permitted, they should be avoided as they are used for internal name mangling.

9.2.3 Serializable Types

A type can only be used as a parameter or return type if kRPC knows how to serialize it. The following types are serializable:

- The C# types double, float, int, long, uint, ulong, bool, string and byte[]
- Any type annotated with KRPCClass
- Any type annotated with KRPCEnum
- Collections of serializable types:
 - System.Collections.Generic.IList<T> where T is a serializable type
 - System.Collections.Generic.IDictionary<K, V> where K is one of int, long, uint, ulong, bool or string and V is a serializable type
 - System.Collections.HashSet<V> where V is a serializable type
- Return types can be void
- Protocol buffer message types from namespace KRPC. Schema. KRPC

9.2.4 Game Scenes

Each service is configured to be available from a particular game scene, or scenes.

class GameScene

SpaceCenter

The game scene showing the Kerbal Space Center buildings.

Flight

The game scene showing a vessel in flight (or on the launchpad/runway).

TrackingStation

The tracking station.

EditorVAB

The Vehicle Assembly Building.

EditorSPH

The Space Plane Hangar.

Editor

Either the VAB or the SPH.

A11

All game scenes.

Examples

• Declare a service that is available in the GameScene.Flight game scene:

```
[KRPCService (GameScene = GameScene.Flight)]
public static class MyService {
    ...
}
```

• Declare a service that is available in the <code>GameScene.Flight</code> and <code>GameScene.Editor</code> game scenes:

```
[KRPCService (GameScene = (GameScene.Flight | GameScene.Editor))]
public static class MyService {
    ...
}
```

9.3 Documentation

Documentation can be added using C# XML documentation. The documentation will be automatically exported to clients when they connect.

9.4 Further Examples

See the SpaceCenter service implementation for more extensive examples.

9.5 Generating Service Code for Static Clients

Some of the client libraries dynamically construct the code necessary to interact with the server when they connect. This means that these libraries will automatically pick up changes to service code. Such client libraries include those for Python and Lua.

Other client libraries required code to be generated and compiled into them statically. They do not automatically pick up changes to service code. Such client libraries include those for C++ and C#.

Code for these 'static' libraries is generated using the *krpc-clientgen* tool. This is provided as a Python script on PyPi. It can be installed using pip:

```
pip install krpc.clientgen
```

You can then run the script from the command line:

9.3. Documentation 389

```
$ krpc-clientgen --help
usage: krpc-clientgen [-h] [-v] [-o OUTPUT] [--ksp KSP]
                       [--output-defs OUTPUT_DEFS]
                      {cpp,csharp} service input [input ...]
Generate client source code for kRPC services.
positional arguments:
 {cpp,csharp} Language to generate
  service
                      Name of service to generate
 input
                       Path to service definition JSON file or assembly
optional arguments:
  -h, --help show this help message and exit -v, --version show program's version number and exit
  -o OUTPUT, --output OUTPUT
                        Path to write source code to. If not specified, writes
                        source code to standard output.
  --ksp KSP
                        Path to Kerbal Space Program directory. Required when
                        reading from an assembly DLL(s)
  --output-defs OUTPUT_DEFS
                        When generting client code from a DLL, output the
                        service definitions to the given JSON file
```

Client code can be generated either directly from an assembly DLL containing the service, or from a JSON file that has previously been generated from an assembly DLL (using the --output-defs flag).

Generating client code from an assembly DLL requires a copy of Kerbal Space Program and a C# runtime to be available on the machine. In contrast, generating client code from a JSON file does not have these requirements and so is more portable.

9.5.1 Example

The following demonstrates how to generate code for the C++ and C# clients to interact with the LaunchControl service, given in an example previously.

krpc-clientgen expects to be passed the location of your copy of Kerbal Space Program, the name of the language to generate, the name of the service (from the *KRPCService* attribute), a path to the assembly containing the service and the path to write the generated code to.

For C++, run the following:

```
krpc-clientgen --ksp=/path/to/ksp cpp LaunchControl LaunchControl.dll
launch_control.hpp
```

To then use the LaunchControl service from C++, you need to link your code against the C++ client library, and include *launch control.hpp*.

For C#, run the following:

```
krpc-clientgen --ksp=/path/to/ksp csharp LaunchControl LaunchControl.dll
LaunchControl.cs
```

To then use the LaunchControl service from a C# client, you need to reference the KRPC.Client.dll and include LaunchControl.cs in your project.

CHAPTER

TEN

COMMUNICATION PROTOCOL

Clients invoke Remote Procedure Calls (RPCs) by communicating with the server using Protocol Buffer v3 messages sent over a TCP/IP connection.

The kRPC download comes with a protocol buffer message definitions file (KRPC.proto) that defines the structure of these messages. It also includes versions of this file compiled for Python, Java and C++ using Google's protocol buffers compiler.

10.1 Establishing a Connection

kRPC consists of two servers: an *RPC Server* (over which clients send and receive RPCs) and a *Stream Server* (over which clients receive *Streams*). A client first connects to the *RPC Server*, then (optionally) to the *Stream Server*.

10.1.1 Connecting to the RPC Server

To establish a connection to the RPC Server, a client must do the following:

- 1. Open a TCP socket to the server on its RPC port (which defaults to 50000).
- 2. Send this 12 byte hello message: 0x48 0x45 0x4C 0x4C 0x4F 0x2D 0x52 0x50 0x43 0x00 0x00 0x00
- 3. Send a 32 byte message containing a name for the connection, that will be displayed on the in-game server window. This should be a UTF-8 encoded string, up to a maximum of 32 bytes in length. If the string is shorter than 32 bytes, it should be padded with zeros.
- 4. Receive a 16 byte unique client identifier. This is sent to the client when the connection is granted, for example after the user has clicked accept on the in-game UI.

For example, this python code will connect to the *RPC Server* at address 127.0.0.1:50000 using the identifier Jeb:

```
import socket
rpc_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
rpc_conn.connect(('127.0.0.1', 50000))
# Send the 12 byte hello message
rpc_conn.sendall(b'\x48\x45\x4C\x4C\x4F\x2D\x52\x50\x43\x00\x00')
# Send the 32 byte client name 'Jeb' padded with zeroes
name = 'Jeb'.encode('utf-8')
name += (b'\x00' * (32-len(name)))
rpc_conn.sendall(name)
# Receive the 16 byte client identifier
identifier = b''
while len(identifier) < 16:</pre>
```

```
identifier += rpc_conn.recv(16 - len(identifier))
# Connection successful. Print out a message along with the client identifier.
printable_identifier = ''.join('%02s' % x for x in identifier)
print('Connected to RPC server, client idenfitier = %s' % printable_identifier)
```

10.1.2 Connecting to the Stream Server

To establish a connection to the Stream Server, a client must first connect to the RPC Server then do the following:

- 1. Open a TCP socket to the server on its Stream port (which defaults to 50001).
- 2. Send this 12 byte hello message: 0x48 0x45 0x4C 0x4C 0x4F 0x2D 0x53 0x54 0x52 0x45 0x41 0x4D
- 3. Send a 16 byte message containing the client's unique identifier. This identifier is given to the client after it successfully connects to the *RPC Server*.
- 4. Receive a 2 byte OK message: 0x4F 0x4B This indicates a successful connection.

Note: Connecting to the Stream Server is optional. If the client doesn't require stream functionality, there is no need to connect.

For example, this python code will connect to the *Stream Server* at address 127.0.0.1:50001. Note that identifier is the unique client identifier received when *connecting to the RPC server*.

```
import socket
stream_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
stream_conn.connect(('127.0.0.1', 50001))
# Send the 12 byte hello message
stream_conn.sendall(b'\x48\x45\x4C\x4C\x4F\x2D\x53\x54\x52\x45\x41\x4D')
# Send the 16 byte client identifier
stream_conn.sendall(identifier)
# Receive the 2 byte OK message
ok_message = b''
while len(ok_message) < 2:
    ok_message += stream_conn.recv(2 - len(ok_message))
# Connection successful
print('Connected to stream server')</pre>
```

10.2 Remote Procedures

Remote procedures are arranged into groups called services. These act as a single-level namespacing to keep things organized. Each service has a unique name used to identify it, and within a service each procedure has a unique name.

10.2.1 Invoking Remote Procedures

Remote procedures are invoked by sending a request message to the RPC server, and waiting for a response message. These messages are encoded as Protocol Buffer messages.

The request message contains the name of the procedure to invoke, and the values of any arguments to pass it. The response message contains the value returned by the procedure (if any) and any errors that were encountered.

Requests are processed in order of receipt. The next request will not be processed until the previous one completes and it's response has been received by the client. When there are multiple client connections, the requests are processed in round-robin order.

10.2.2 Anatomy of a Request

A request is sent to the server using a Request Protocol Buffer message with the following format:

```
message Request {
    string service = 1;
    string procedure = 2;
    repeated Argument arguments = 3;
}

message Argument {
    uint32 position = 1;
    bytes value = 2;
}
```

The fields are:

- service The name of the service in which the remote procedure is defined.
- procedure The name of the remote procedure to invoke.
- arguments A sequence of Argument messages containing the values of the procedure's arguments. The fields are:
 - position The zero-indexed position of the of the argument in the procedure's signature.
 - value The value of the argument, encoded in Protocol Buffer format.

The Argument messages have a position field to allow values for default arguments to be omitted. See *Protocol Buffer Encoding* for details on how to serialize the argument values.

10.2.3 Anatomy of a Response

A response is sent to the client using a Response Protocol Buffer message with the following format:

```
message Response {
  double time = 1;
  bool has_error = 2;
  string error = 3;
  bool has_return_value = 4;
  bytes return_value = 5;
}
```

The fields are:

- time The universal time (in seconds) when the request completed processing.
- has_error True if there was an error executing the remote procedure.
- error If has_error is true, contains a description of the error.
- has_return_value True if the remote procedure returned a value.
- return_value If has_return_value is true and has_error is false, contains the value returned by the remote procedure, encoded in protocol buffer format.

See Protocol Buffer Encoding for details on how to unserialize the return value.

10.2.4 Encoding and Sending Requests and Responses

To send a request:

- 1. Encode a Request message using the Protocol Buffer Encoding.
- 2. Send the size in bytes of the encoded Request message, encoded as a Protocol Buffer varint.
- 3. Send the message data.

To receive a response:

- 1. Read a Protocol Buffer varint, which contains the length of the Response message data in bytes.
- 2. Receive and decode the Response message.

10.2.5 Example RPC invocation

The following Python script invokes the GetStatus procedure from the *KRPC service* using an already established connection to the server (the rpc_conn variable).

The krpc.schema.KRPC package contains the Protocol Buffer message formats Request, Response and Status compiled to python code using the Protocol Buffer compiler. The EncodeVarint and DecodeVarint functions are used to encode/decode integers to/from the Protocol Buffer varint format.

```
def EncodeVarint(value):
 return krpc.Encoder.encode(value,krpc.types.ValueType("int32"))
def DecodeVarint(data) :
  return krpc.Decoder.decode(data,krpc.types.ValueType("int32"))
# Create Request message
request = krpc.schema.KRPC.Request()
request.service = 'KRPC'
request.procedure = 'GetStatus'
# Encode and send the request
data = request.SerializeToString()
header = EncodeVarint(len(data))
rpc_conn.sendall(header + data)
# Receive the size of the response data
data = b''
while True:
   data += rpc_conn.recv(1)
   try:
        size = DecodeVarint(data)
       break
   except IndexError:
       pass
# Receive the response data
dat.a = b''
while len(data) < size:</pre>
   data += rpc_conn.recv(size - len(data))
# Decode the response message
response = krpc.schema.KRPC.Response()
response.ParseFromString(data)
# Check for an error response
```

```
if response.has_error:
    print('ERROR:', response.error)

# Decode the return value as a Status message
else:
    status = krpc.schema.KRPC.Status()
    assert response.has_return_value
    status.ParseFromString(response.return_value)

# Print out the version string from the Status message
    print(status.version)
```

10.3 Protocol Buffer Encoding

Values passed as arguments or received as return values are encoded using the Protocol Buffer serialization format:

- Documentation for this encoding can be found here: https://developers.google.com/protocol-buffers/docs/encoding
- Protocol Buffer serialization libraries are available for C++/Java/Python here: http://code.google.com/p/protobuf/downloads/list
- There are implementations available for most popular languages here: http://code.google.com/p/protobuf/wiki/ThirdPartyAddOns

10.4 Streams

Streams allow the client to repeatedly execute a Remote Procedure Call on the server and receive its results, without needing to repeatedly call the Remote Procedure Call directly, avoiding the communication overhead that this would involve.

A stream is created on the server by calling *AddStream* which returns a unique identifier for the stream. Once a client is finished with a stream, it can remove it from the server by calling *RemoveStream* with the stream's identifier. Streams are automatically removed when the client that created it disconnects from the server. Streams are local to each client. There is no way to share a stream between clients.

The results of the RPCs for each stream are sent to the client over the Stream Server's TCP/IP connection, as repeated *stream messages*. The RPC for each stream is invoked every fixed update. Updates are sent to clients when the value returned by the RPC changes to minimize network traffic.

10.4.1 Anatomy of a Stream Message

A stream message is sent to the client using a StreamMessage Protocol Buffer message with the following format:

```
message StreamMessage {
   repeated StreamResponse responses = 1;
}
```

This message contains a list of StreamResponse messages, one for each stream that exists on the server for that client, with the following format:

```
message StreamResponse {
  uint32 id = 1;
  Response response = 2;
}
```

The fields are:

- id The identifier of the stream. This is the value returned by AddStream when the stream is created.
- response A Response message containing the result of the stream's RPC. This is identical to the Response message returned when calling the RPC directly. See *Anatomy of a Response* for details on the format and contents of this message.

10.5 KRPC Service

The server provides a service called KRPC containing procedures that are used to retrieve information about the server and add/remove streams.

10.5.1 GetStatus

The GetStatus procedure returns status information about the server. It returns a Protocol Buffer message with the format:

```
message Status {
  string version = 1;
  uint64 bytes_read = 2;
  uint64 bytes_written = 3;
  float bytes read rate = 4;
  float bytes_written_rate = 5;
  uint64 rpcs_executed = 6;
  float rpc_rate = 7;
  bool one_rpc_per_update = 8;
  uint32 max_time_per_update = 9;
  bool adaptive_rate_control = 10;
  bool blocking recv = 11;
  uint32 recv_timeout = 12;
  float time_per_rpc_update = 13;
  float poll_time_per_rpc_update = 14;
  float exec_time_per_rpc_update = 15;
  uint32 stream_rpcs = 16;
  uint64 stream rpcs executed = 17;
  float stream_rpc_rate = 18;
  float time_per_stream_update = 19;
```

The version field contains the version string of the server. The remaining fields contain performance information about the server.

10.5.2 GetServices

The GetServices procedure returns a Protocol Buffer message containing information about all of the services and procedures provided by the server. It also provides type information about each procedure, in the form of *attributes*. The format of the message is:

```
message Services {
   repeated Service services = 1;
}
```

This contains a single field, which is a list of Service messages with information about each service provided by the server. The content of these Service messages are *documented below*.

10.5.3 AddStream

The AddStream procedure adds a new stream to the server. It takes a single argument containing the RPC to invoke, encoded as a Request object. See *Anatomy of a Request* for the format and contents of this object. See *Streams* for more information on working with streams.

10.5.4 RemoveStream

The RemoveStream procedure removes a stream from the server. It takes a single argument – the identifier of the stream to be removed. This is the identifier returned when the stream was added by calling *AddStream*. See *Streams* for more information on working with streams.

10.6 Service Description Message

The *GetServices procedure* returns information about all of the services provided by the server. Details about a service are given by a Service message, with the format:

```
message Service {
   string name = 1;
   repeated Procedure procedures = 2;
   repeated Class classes = 3;
   repeated Enumeration enumerations = 4;
   string documentation = 5;
}
```

The fields are:

- name The name of the service.
- procedures A list of Procedure messages, one for each procedure defined by the service.
- classes A list of Class messages, one for each KRPCClass defined by the service.
- enumerations A list of Enumeration messages, one for each KRPCEnum defined by the service.
- documentation Documentation for the service, as C# XML documentation.

Note: See the Extending kRPC documentation for more details about KRPCClass and KRPCEnum.

10.6.1 Procedures

Details about a procedure are given by a Procedure message, with the format:

```
message Procedure {
    string name = 1;
    repeated Parameter parameters = 2;
    bool has_return_type = 3;
    string return_type = 4;
    repeated string attributes = 5;
    string documentation = 6;
}

message Parameter {
    string name = 1;
    string type = 2;
    bool has_default_argument = 3;
    bytes default_argument = 4;
}
```

The fields are:

- name The name of the procedure.
- parameters A list of Parameter messages containing details of the procedure's parameters, with the following fields:
 - name The name of the parameter, to allow parameter passing by name.
 - type The *type* of the parameter.
 - has_default_argument True if the parameter has a default value.
 - default_argument If has_default_argument is true, contains the value of the default value of the parameter, encoded using Protocol Buffer format.
- has_return_type True if the procedure returns a value.
- return_type If has_return_type is true, contains the return type of the procedure.
- attributes The procedure's attributes.
- documentation Documentation for the procedure, as C# XML documentation.

10.6.2 Classes

Details about each KRPCClass are specified in a Class message, with the format:

```
message Class {
   string name = 1;
   string documentation = 2;
}
```

The fields are:

- name The name of the class.
- documentation Documentation for the class, as C# XML documentation.

10.6.3 Enumerations

Details about each KRPCEnum are specified in an Enumeration message, with the format:

```
message Enumeration {
   string name = 1;
   repeated EnumerationValue values = 2;
   string documentation = 3;
}

message EnumerationValue {
   string name = 1;
   int32 value = 2;
   string documentation = 3;
}
```

The fields are:

- name The name of the enumeration.
- values A list of EnumerationValue messages, indicating the values that the enumeration can be assigned. The fields are:
 - name The name associated with the value for the enumeration.
 - value The possible value for the enumeration as a 32-bit integer.
 - documentation Documentation for the enumeration value, as C# XML documentation.
- documentation Documentation for the enumeration, as C# XML documentation.

10.6.4 Attributes

Additional type information about a procedure is encoded as a list of attributes, and included in the Procedure message. For example, if the procedure implements a method for a class (see *proxy objects*) this fact will be specified in the attributes.

The following attributes specify what the procedure implements:

- Property.Get (property-name)
 - Indicates that the procedure is a property getter (for the service) with the given property-name.
- Property.Set (property-name)
 - Indicates that the procedure is a property setter (for the service) with the given property-name.
- Class.Method(class-name, method-name)
 - Indicates that the procedure is a method for a class with the given class-name and method-name.
- Class.StaticMethod(class-name, method-name)
 - Indicates that the procedure is a static method for a class with the given class-name and method-name.
- Class.Property.Get(class-name,property-name)
 - Indicates that the procedure is a property getter for a class with the given class-name and property-name.
- Class.Property.Set (class-name, property-name)
 - Indicates that the procedure is a property setter for a class with the given class-name and property-name.

The following attributes specify more details about the return and parameter types of the procedure.

• ReturnType.type-name

Specifies the actual *return type* of the procedure, if it differs to the type specified in the Procedure message. For example, this is used with *proxy objects*.

• ParameterType (parameter-position).type-name

Specifies the actual *parameter type* of the procedure, if it differs to the type of the corresponding parameter specified in the Parameter message. For example, this is used with *proxy objects*.

10.6.5 Type Names

The GetServices procedure returns type information about parameters and return values as strings. Type names can be any of the following:

- A Protocol Buffer value type. One of double, float, int32, int64, uint32, uint64, bool, string or bytes
- A KRPCClass, in the format Class (ClassName)
- A KRPCEnum, in the format Enum (ClassName)
- A Protocol Buffer message type, in the format KRPC.MessageType. Only message types defined in krpc.proto are permitted.

10.6.6 Proxy Objects

kRPC allows procedures to create objects on the server, and passes unique identifiers for them to the client. This allows the client to create a *proxy* object for the actual object, whose methods and properties make remote procedure calls to the server. Object identifiers have type uint64.

When a procedure returns a proxy object, the procedure will have the attribute ReturnType.Class(ClassName) where ClassName is the name of the class.

When a procedure takes a proxy object as a parameter, the procedure will have the attribute ParameterType(n).Class(ClassName) where n is the position of the parameter and ClassName is the name of the class.

CHAPTER

ELEVEN

INTERNALS OF KRPC

11.1 Server Performance Settings

kRPC receives RPCs, execute them and returns results when the mod's FixedUpdate method is invoked. kRPC may execute multiple RPCs within a single call to FixedUpdate and this behavior is controlled by several configurable settings:

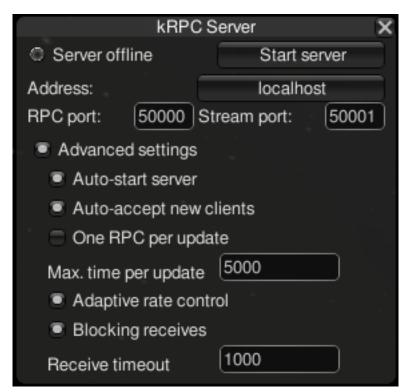


Fig. 11.1: Server window showing the advanced settings.

- 1. Auto-start server: When enabled, the server will start automatically when the game loads.
- 2. **Auto-accept new clients**: When enabled, new client connections are automatically allowed. When disabled, a pop-up is displayed asking whether the new client connection should be allowed.
- 3. One RPC per update: When enabled, the server will execute at most one RPC per client per update.
- 4. **Maximum time per update**: This is the maximum number of nanoseconds that the server will spend processing RPCs in the call to FixedUpdte. This is used to limit the time taken by the server per frame. A high value, for

example 20000 ns, will allow the server to process many RPCs at the expense of the game's framerate. A low value, for example 1000 ns, won't allow the server to execute many RPCs per update, but will allow the game to run at a much higher framerate.

- 5. **Adaptive rate control**: If enabled, the server will automatically adjust the maximum time per update parameter, so that the game runs at a minimum of 60 FPS.
- 6. **Blocking receives**: If enabled, when the server checks for new RPCs from clients, it will wait for a up to a fixed amount of time.
- 7. **Receive timeout**: When blocking receives are enabled, this is the maximum amount of time the server will wait for a new RPC from a client.

PYTHON MODULE INDEX

k

krpc,300 krpc.client,300 krpc.stream,301

404 Python Module Index

k

krpc, 220

406 Lua Module Index

Symbols	AlarmAction.do_nothing_delete_when_passed (in mod-
call() (krpc.stream.Stream method), 301	ule KerbalAlarmClock), 297, 378
۸	AlarmAction.kill_warp (in module KerbalAlarmClock), 297, 378
A	AlarmAction.kill_warp_only (in module KerbalAlarm-
abort (SpaceCenter.Control attribute), 247, 328	Clock), 297, 378
acceleration (InfernalRobotics.Servo attribute), 291, 373	AlarmAction.message_only (in module KerbalAlarm-
action (KerbalAlarmClock.Alarm attribute), 294, 375	Clock), 297, 378
actions (SpaceCenter.Module attribute), 261, 343	AlarmAction.pause_game (in module KerbalAlarm-
activate_next_stage() (SpaceCenter.Control method), 248, 330	Clock), 297, 378
ActivateNextStage() (Java method), 176	$alarms\ (Kerbal Alarm Clock. Kerbal Alarm Clock\ attribute),$
Active (Java field), 192	293, 375
active (SpaceCenter.Engine attribute), 265, 347	alarms_with_type() (KerbalAlarm-
active (SpaceCenter.Light attribute), 270, 351	Clock.KerbalAlarmClock static method), 293, 375
active (SpaceCenter.ReactionWheel attribute), 274, 356	AlarmsWithType(AlarmType) (Java method), 213
active (SpaceCenter.ResourceHarvester attribute), 273,	AlarmType (class in KerbalAlarmClock), 295, 377
355	AlarmType (Java enum), 215
active (SpaceCenter.Sensor attribute), 275, 356	AlarmType.apoapsis (in module KerbalAlarmClock),
active() (SpaceCenter.ResourceConverter method), 272, 354	295, 377
Active(int) (Java method), 193	AlarmType.ascending_node (in module KerbalAlarm-
active_vessel (SpaceCenter.SpaceCenter attribute), 220,	Clock), 295, 377
302	AlarmType.closest (in module KerbalAlarmClock), 296,
add_node() (SpaceCenter.Control method), 248, 330	377 AlarmType.contract (in module KerbalAlarmClock), 296,
add_stream() (krpc.client.Client method), 300	377
AddNode(double, float, float, float) (Java method), 176	AlarmType.contract_auto (in module KerbalAlarm-
addStream(Class, String, Object) (Java method), 155	Clock), 296, 377
addStream(RemoteObject, String, Object) (Java method), 155	AlarmType.crew (in module KerbalAlarmClock), 296,
Adjacent (Java field), 202	377
aerodynamic_force (SpaceCenter.Flight attribute), 240,	AlarmType.descending_node (in module KerbalAlarm-
322	Clock), 296, 377
Alarm (class in KerbalAlarmClock), 294, 375	AlarmType.distance (in module KerbalAlarmClock), 296,
Alarm (Java class), 213	377 AlarmType.earth_time (in module KerbalAlarmClock),
alarm_with_name() (KerbalAlarm-	296, 378
Clock.KerbalAlarmClock static method),	AlarmType.launch_rendevous (in module KerbalAlarm-
293, 375	Clock), 296, 378
AlarmAction (class in KerbalAlarmClock), 296, 378	AlarmType.maneuver (in module KerbalAlarmClock),
AlarmAction (Java enum), 216 AlarmAction.do_nothing (in module KerbalAlarmClock),	295, 377
296, 378	AlarmType.maneuver_auto (in module KerbalAlarm-
270, 370	Clock) 295 377

AlarmType.periapsis (in module KerbalAlarmClock), 295, 377	BedrockHeight(double, double) (Java method), 166 BedrockPosition(double, double, ReferenceFrame) (Java
AlarmType.raw (in module KerbalAlarmClock), 295, 377	method), 167
AlarmType.soi_change (in module KerbalAlarmClock), 296, 378	bodies (SpaceCenter.SpaceCenter attribute), 221, 302 body (SpaceCenter.Orbit attribute), 243, 324
AlarmType.soi_change_auto (in module KerbalAlarm-	brakes (SpaceCenter.Control attribute), 247, 328
Clock), 296, 378	Broken (Java field), 191, 193, 196
AlarmType.transfer (in module KerbalAlarmClock), 296,	broken (SpaceCenter.ReactionWheel attribute), 274, 356
378	burn_vector() (SpaceCenter.Node method), 282, 364
AlarmType.transfer_modelled (in module KerbalAlarm-Clock), 296, 378	BurnVector(ReferenceFrame) (Java method), 203
AlarmWithName(String) (Java method), 213	C
All (GameScene attribute), 389	can_rails_warp_at() (SpaceCenter.SpaceCenter static
all (SpaceCenter.Parts attribute), 250, 332	method), 222, 303
amount() (SpaceCenter.Resources method), 281, 363	can_restart (SpaceCenter.Engine attribute), 267, 349
Amount(String) (Java method), 202	can_shutdown (SpaceCenter.Engine attribute), 267, 349
angle_of_attack (SpaceCenter.Flight attribute), 241, 323	CanRailsWarpAt(int) (Java method), 156
angular_velocity() (SpaceCenter.CelestialBody method),	Capacity (Java field), 194
237, 318	CelestialBody (class in SpaceCenter), 233, 314
angular_velocity() (SpaceCenter.Vessel method), 232,	CelestialBody (Java class), 166
313	center_of_mass (SpaceCenter.Flight attribute), 238, 320
Angular Velocity (Reference Frame) (Java method), 165,	children (SpaceCenter.Part attribute), 254, 335
169	clear_drawing() (SpaceCenter.SpaceCenter static
anti_normal (SpaceCenter.Flight attribute), 239, 321	method), 224, 305
anti_radial (SpaceCenter.Flight attribute), 240, 321	clear_target() (SpaceCenter.SpaceCenter static method),
AntiNormal (Java field), 177	221, 302
AntiRadial (Java field), 177	ClearDrawing() (Java method), 159
AntiTarget (Java field), 177	ClearTarget() (Java method), 155
Apoapsis (Java field), 215	Client (class in krpc), 220
apoapsis (SpaceCenter.Orbit attribute), 243, 325	Client (class in krpc), 220 Client (class in krpc.client), 300
apoapsis_altitude (SpaceCenter.Orbit attribute), 243, 325	close() (Java method), 155
argument_of_periapsis (SpaceCenter.Orbit attribute),	close() (krpc.Client method), 220
245, 326	close() (krpc.client.Client method), 301
AscendingNode (Java field), 215	Closest (Java field), 215
atmosphere_density (SpaceCenter.Flight attribute), 240,	Comms (class in SpaceCenter), 284, 366
321	Comms (Java class), 204
atmosphere_depth (SpaceCenter.CelestialBody attribute),	comms (SpaceCenter. Vessel attribute), 226, 307
235, 317	config_speed (InfernalRobotics.Servo attribute), 291, 373
auto_pilot (SpaceCenter.Vessel attribute), 225, 307	connect() (in module krpc), 220, 300
AutoPilot (class in SpaceCenter), 286, 367	Connection (Java class), 154
AutoPilot (Java class), 206	Contract (Java field), 215
available_thrust (SpaceCenter.Engine attribute), 266, 347	Contract (Java field), 215 ContractAuto (Java field), 215
available_thrust (SpaceCenter.Vessel attribute), 227, 308	Control (class in SpaceCenter), 246, 327
axially_attached (SpaceCenter.Part attribute), 254, 336	Control (Java class), 174
	control (SpaceCenter. Vessel attribute), 225, 307
В	Control Group (class in Infernal Robotics), 289, 370
ballistic_coefficient (SpaceCenter.Flight attribute), 242, 324	ControlGroup (Java class), 209
	controlling (SpaceCenter.Parts attribute), 251, 332
Base (Java field), 165	core_temperature (SpaceCenter.ResourceHarvester at-
bedrock_altitude (SpaceCenter.Flight attribute), 237, 319	tribute), 274, 355
bedrock_height() (SpaceCenter.CelestialBody method),	cost (SpaceCenter.Part attribute), 253, 335
234, 315	count (SpaceCenter.ResourceConverter attribute), 272,
bedrock_position() (SpaceCenter.CelestialBody method), 235, 316	353

create_alarm() (KerbalAlarmClock.KerbalAlarmClock static method), 293, 375	docked_part (SpaceCenter.DockingPort attribute), 263, 344
CreateAlarm(AlarmType, String, double) (Java method),	Docking (Java field), 188
213	docking_port (SpaceCenter.Part attribute), 257, 339
Crew (Java field), 215	docking_port_with_name() (SpaceCenter.Parts method),
crossfeed (SpaceCenter.Part attribute), 257, 338	252, 333
current_speed (InfernalRobotics.Servo attribute), 291,	docking_ports (SpaceCenter.Parts attribute), 251, 333
373	DockingPort (class in SpaceCenter), 262, 344
current_stage (SpaceCenter.Control attribute), 248, 329	DockingPort (Java class), 186
Cut (Java field), 192	DockingPortState (class in SpaceCenter), 264, 346
	DockingPortState (Java enum), 187
D	DockingPortState.docked (in module SpaceCenter), 265,
Debris (Java field), 165	346
Decouple() (Java method), 186	DockingPortState.docking (in module SpaceCenter), 265,
decouple() (SpaceCenter.Decoupler method), 262, 344	346
decouple_stage (SpaceCenter.Part attribute), 254, 336	DockingPortState.moving (in module SpaceCenter), 265,
decoupled (SpaceCenter.Decoupler attribute), 262, 344	346
Decoupler (class in SpaceCenter), 262, 343	DockingPortState.ready (in module SpaceCenter), 264,
Decoupler (Java class), 186	346
decoupler (SpaceCenter.Part attribute), 257, 339	DockingPortState.shielded (in module SpaceCenter), 265,
decouplers (SpaceCenter.Parts attribute), 251, 333	346
delta_v (SpaceCenter.Node attribute), 282, 364	DockingPortState.undocking (in module SpaceCenter),
density() (SpaceCenter.Resources static method), 281,	265, 346
363	DockingPortWithName(String) (Java method), 179
Density(String) (Java method), 202	DoNothing (Java field), 216
Deploy() (Java method), 192	DoNothingDeleteWhenPassed (Java field), 216
deploy() (SpaceCenter.Parachute method), 270, 352	drag (SpaceCenter.Flight attribute), 240, 322
deploy_altitude (SpaceCenter.Parachute attribute), 270,	drag_coefficient (SpaceCenter.Flight attribute), 242, 324
352	draw_direction() (SpaceCenter.SpaceCenter static
deploy_min_pressure (SpaceCenter.Parachute attribute),	method), 224, 305
270, 352	draw_line() (SpaceCenter.SpaceCenter static method),
Deployed (Java field), 190–192	224, 305
deployed (SpaceCenter.LandingGear attribute), 268, 350	DrawDirection(org.javatuples.Triplet, ReferenceFrame,
deployed (SpaceCenter.LandingLeg attribute), 269, 350	org.javatuples.Triplet, float) (Java method), 158
deployed (SpaceCenter.Parachute attribute), 270, 352	DrawLine(org.javatuples.Triplet, org.javatuples.Triplet,
deployed (SpaceCenter.Radiator attribute), 271, 353	ReferenceFrame, org.javatuples.Triplet) (Java
deployed (SpaceCenter.ResourceHarvester attribute),	method), 158
273, 355	dry_mass (SpaceCenter.Part attribute), 255, 337
deployed (SpaceCenter.SolarPanel attribute), 275, 357	dry_mass (SpaceCenter.Vessel attribute), 226, 307
Deploying (Java field), 190, 191	dynamic_pressure (SpaceCenter.Flight attribute), 240,
DescendingNode (Java field), 215	321
direction (SpaceCenter.Flight attribute), 239, 320	E
direction() (SpaceCenter.CelestialBody method), 237,	
318	EarthTime (Java field), 215
direction() (SpaceCenter.DockingPort method), 263, 345	eccentric_anomaly (SpaceCenter.Orbit attribute), 245,
direction() (SpaceCenter.Node method), 284, 366	327
direction() (SpaceCenter.Part method), 259, 341	eccentricity (SpaceCenter.Orbit attribute), 245, 326
direction() (SpaceCenter. Vessel method), 232, 313	Editor (GameScene attribute), 389
Direction(ReferenceFrame) (Java method), 165, 169, 183,	EditorSPH (GameScene attribute), 389
187, 204	EditorVAB (GameScene attribute), 389
Disengage() (Java method), 206	elevation (SpaceCenter.Flight attribute), 238, 319
disengage() (SpaceCenter.AutoPilot method), 286, 367	energy_flow (SpaceCenter.SolarPanel attribute), 276, 357
Distance (Java field), 215	Engage() (Java method), 206
Docked (Java field), 165, 188	engage() (SpaceCenter.AutoPilot method), 286, 367

Engine (class in SpaceCenter), 265, 347	GetActionGroup(int) (Java method), 176
Engine (Java class), 188	getActions() (Java method), 185
engine (SpaceCenter.Part attribute), 257, 339	getActive() (Java method), 188, 191, 194, 195
engines (SpaceCenter.Parts attribute), 252, 334	getActiveVessel() (Java method), 155
epoch (SpaceCenter.Orbit attribute), 245, 327	getAerodynamicForce() (Java method), 171
equatorial_radius (SpaceCenter.CelestialBody attribute),	getAlarms() (Java method), 213
234, 315	getAll() (Java method), 178
equivalent_air_speed (SpaceCenter.Flight attribute), 241,	getAngleOfAttack() (Java method), 172
323	getAntiNormal() (Java method), 170
error (SpaceCenter.AutoPilot attribute), 286, 367	getAntiRadial() (Java method), 170
Escaping (Java field), 165	getApoapsis() (Java method), 173
events (SpaceCenter.Module attribute), 260, 342	getApoapsisAltitude() (Java method), 173
expanded (InfernalRobotics.ControlGroup attribute), 289,	getArgumentOfPeriapsis() (Java method), 174
371	getAtmosphereDensity() (Java method), 170
Extended (Java field), 193, 196	getAtmosphereDepth() (Java method), 167
Extending (Java field), 193, 196	getAutoPilot() (Java method), 159
external_temperature (SpaceCenter.Part attribute), 255,	getAvailableThrust() (Java method), 160, 189
337	getAxiallyAttached() (Java method), 180
extraction_rate (SpaceCenter.ResourceHarvester at-	getBallisticCoefficient() (Java method), 172
tribute), 273, 355	getBedrockAltitude() (Java method), 169
tribute), 273, 333	getBodies() (Java method), 155
F	getBody() (Java method), 173
	getBrakes() (Java method), 175
far_available (SpaceCenter.SpaceCenter attribute), 224,	getBroken() (Java method), 175 getBroken() (Java method), 195
305	· · · · · · · · · · · · · · · · · · ·
fields (SpaceCenter.Module attribute), 260, 342	getCanRestart() (Java method), 189
Flight (class in SpaceCenter), 237, 319	getCanShutdown() (Java method), 189
Flight (GameScene attribute), 388	getCenterOfMass() (Java method), 170
Flight (Java class), 169	getChildren() (Java method), 180
flight() (SpaceCenter.Vessel method), 225, 306	getComms() (Java method), 160
Flight(ReferenceFrame) (Java method), 159	getConfigSpeed() (Java method), 211
flow_mode() (SpaceCenter.Resources static method),	getControl() (Java method), 159
281, 363	getControlling() (Java method), 178
FlowMode(String) (Java method), 202	getCoreTemperature() (Java method), 195
Flying (Java field), 165	getCost() (Java method), 180
forward (SpaceCenter.Control attribute), 247, 329	getCount() (Java method), 193
$forward_key \ (Infernal Robotics. Control Group \ attribute),$	getCrossfeed() (Java method), 182
289, 371	getCurrentSpeed() (Java method), 211
fuel_lines_from (SpaceCenter.Part attribute), 257, 339	getCurrentStage() (Java method), 176
fuel_lines_to (SpaceCenter.Part attribute), 257, 339	getDecoupled() (Java method), 186
•	getDecoupler() (Java method), 182
G	getDecouplers() (Java method), 179
g (SpaceCenter.SpaceCenter attribute), 221, 303	getDecoupleStage() (Java method), 181
g_force (SpaceCenter.Flight attribute), 237, 319	getDeltaV() (Java method), 203
GameScene (built-in class), 388	getDeployAltitude() (Java method), 192
gear (SpaceCenter.Control attribute), 247, 328	getDeployed() (Java method), 190–192, 194, 196
get() (Java method), 155	getDeployMinPressure() (Java method), 192
get_action_group() (SpaceCenter.Control method), 248,	getDirection() (Java method), 170
330	getDockedPart() (Java method), 186
get_field() (SpaceCenter.Module method), 260, 342	getDockingPort() (Java method), 182
get_status() (krpc.client.KRPC method), 301	getDockingPorts() (Java method), 179
get_status() (krpc.KRPC method), 220	getDrag() (Java method), 171
getAbort() (Java method), 175	getDragCoefficient() (Java method), 172
getAcceleration() (Java method), 211	getDryMass() (Java method), 160, 181
getAction() (Java method), 213	getDynamicPressure() (Java method), 170
generion() (Java memou), 213	

getEccentricAnomaly() (Java method), 174	getLaunchClamp() (Java method), 183
getEccentricity() (Java method), 174	getLaunchClamps() (Java method), 179
getElevation() (Java method), 169	getLift() (Java method), 171
getEnergyFlow() (Java method), 196	getLiftCoefficient() (Java method), 172
getEngine() (Java method), 182	getLight() (Java method), 183
getEngines() (Java method), 179	getLights() (Java method), 175, 180
getEpoch() (Java method), 174	getLongitude() (Java method), 170
getEquatorialRadius() (Java method), 166	getLongitudeOfAscendingNode() (Java method), 174
getEquivalentAirSpeed() (Java method), 171	getMach() (Java method), 171
getError() (Java method), 206	getMargin() (Java method), 213
getEvents() (Java method), 185	getMass() (Java method), 160, 166, 181
getExpanded() (Java method), 209	getMassless() (Java method), 181
getExternalTemperature() (Java method), 181	getMaxConfigPosition() (Java method), 210
getExtractionRate() (Java method), 194	getMaximumRailsWarpFactor() (Java method), 156
getFARAvailable() (Java method), 158	getMaxPosition() (Java method), 211
GetField(String) (Java method), 184	getMaxRollSpeed() (Java method), 207
getFields() (Java method), 184	getMaxRotationSpeed() (Java method), 207
getForward() (Java method), 176	getMaxSkinTemperature() (Java method), 181
getForwardKey() (Java method), 209	getMaxTemperature() (Java method), 181
getFuelLinesFrom() (Java method), 182	getMaxThrust() (Java method), 160, 189
getFuelLinesTo() (Java method), 182	getMaxVacuumThrust() (Java method), 160, 189
getG() (Java method), 156	getMeanAltitude() (Java method), 169
getGear() (Java method), 175	getMeanAnomaly() (Java method), 174
getGForce() (Java method), 169	getMeanAnomalyAtEpoch() (Java method), 174
getGimballed() (Java method), 190	getMET() (Java method), 159
getGimbalLimit() (Java method), 190	getMinConfigPosition() (Java method), 210
getGimbalLocked() (Java method), 190	getMinPosition() (Java method), 210
getGimbalRange() (Java method), 190	getModules() (Java method), 182
getGravitationalParameter() (Java method), 166	getName() (Java method), 159, 166, 180, 184, 186, 209,
getHasAtmosphere() (Java method), 167	210, 214
getHasAtmosphericOxygen() (Java method), 167	getNames() (Java method), 201
getHasConnection() (Java method), 205	getNextOrbit() (Java method), 174
getHasConnectionToGroundStation() (Java method), 205	getNodes() (Java method), 177
getHasFlightComputer() (Java method), 205	
getHasFuel() (Java method), 189	getNonRotatingReferenceFrame() (Java method), 168
	getNormal() (Java method), 170, 202
getHasLocalControl() (Java method), 205	getNotes() (Java method), 214
getHasShield() (Java method), 186	getOptimumCoreTemperature() (Java method), 195
getHeading() (Java method), 170	getOrbit() (Java method), 159, 166, 204
getHorizontalSpeed() (Java method), 170	getOrbitalReferenceFrame() (Java method), 161, 168, 204
getID() (Java method), 214	getParachute() (Java method), 183
getImpactTolerance() (Java method), 181	getParachutes() (Java method), 180
getImpulse() (Java method), 186	getParent() (Java method), 180
getInclination() (Java method), 174	getPart() (Java method), 184, 186, 188, 190–196
getIsAxisInverted() (Java method), 211	getParts() (Java method), 160
getIsFreeMoving() (Java method), 211	getPeriapsis() (Java method), 173
getIsFuelLine() (Java method), 182	getPeriapsisAltitude() (Java method), 173
getIsLocked() (Java method), 211	getPeriod() (Java method), 173
getIsMoving() (Java method), 211	getPhysicsWarpFactor() (Java method), 156
getKerbinSeaLevelSpecificImpulse() (Java method), 160,	getPitch() (Java method), 170, 175
189	getPitchTorque() (Java method), 195
getLandingGear() (Java method), 179, 182	getPosition() (Java method), 210
getLandingLeg() (Java method), 182	getPowerUsage() (Java method), 191, 195
getLandingLegs() (Java method), 179	getPrograde() (Java method), 170, 202
getLatitude() (Java method), 169	getPropellantRatios() (Java method), 176, 202

getPropellants() (Java method), 189	getSpeedMode() (Java method), 175
getRadial() (Java method), 170, 203	getSpeedOfSound() (Java method), 171
getRadiallyAttached() (Java method), 181	getSphereOfInfluence() (Java method), 167
getRadiator() (Java method), 183	getStage() (Java method), 181
getRadiators() (Java method), 180	getStallFraction() (Java method), 172
getRadius() (Java method), 173	getState() (Java method), 186, 190, 192, 194, 196
getRailsWarpFactor() (Java method), 156	getStaticAirTemperature() (Java method), 172
getRCS() (Java method), 175	getStaticPressure() (Java method), 171
getReactionWheel() (Java method), 183	getSunExposure() (Java method), 196
getReactionWheels() (Java method), 180	getSurfaceAltitude() (Java method), 169
getReengageDistance() (Java method), 186	getSurfaceGravity() (Java method), 166
getReferenceFrame() (Java method), 161, 167, 183, 187,	getSurfaceReferenceFrame() (Java method), 161
204, 206	getSurfaceVelocityReferenceFrame() (Java method), 163
getRemaining() (Java method), 214	getTarget() (Java method), 159
getRemainingDeltaV() (Java method), 203	getTargetBody() (Java method), 155
getRemoteTechAvailable() (Java method), 158	getTargetDirection() (Java method), 206
getRepeat() (Java method), 214	getTargetDockingPort() (Java method), 155
getRepeatPeriod() (Java method), 214	getTargetRoll() (Java method), 206
getResourceConverter() (Java method), 183	getTargetVessel() (Java method), 155
getResourceConverters() (Java method), 180	getTemperature() (Java method), 181
getResourceHarvester() (Java method), 183	getTerminalVelocity() (Java method), 171
getResourceHarvesters() (Java method), 180	getThermalConductionFlux() (Java method), 182
getResources() (Java method), 160, 182	getThermalConvectionFlux() (Java method), 182
getRetrograde() (Java method), 170	getThermalEfficiency() (Java method), 195
getReverseKey() (Java method), 209	getThermalInternalFlux() (Java method), 182
getRight() (Java method), 176	getThermalMass() (Java method), 181
getRoll() (Java method), 170, 176	getThermalRadiationFlux() (Java method), 182
getRollError() (Java method), 206	getThermalResourceMass() (Java method), 182
getRollSpeedMultiplier() (Java method), 207	getThermalSkinMass() (Java method), 181
getRollTorque() (Java method), 195	getThermalSkinToInternalFlux() (Java method), 182
getRoot() (Java method), 178	getThrottle() (Java method), 175, 189
getRotation() (Java method), 170	getThrottleLocked() (Java method), 173, 189
getRotation() (Java method), 176 getRotationalPeriod() (Java method), 166	getThrust() (Java method), 160, 189
getRotationalSpeed() (Java method), 166	getThrustLimit() (Java method), 189 getThrustLimit() (Java method), 189
getRotationSpeedMultiplier() (Java method), 207 getSAS() (Java method), 175, 206	getThrustSpecificFuelConsumption() (Java method), 172 getTime() (Java method), 213
	9
getSASMode() (Java method), 175, 207	getTimeTo() (Java method), 203
getSatellites() (Java method), 166	getTimeToApoapsis() (Java method), 173
getSemiMajorAxis() (Java method), 173	getTimeToPeriapsis() (Java method), 174
getSemiMinorAxis() (Java method), 173	getTimeToSOIChange() (Java method), 174
getSensor() (Java method), 183	getTitle() (Java method), 180
getSensors() (Java method), 180	getTotalAirTemperature() (Java method), 172
getServoGroups() (Java method), 208	getType() (Java method), 159, 214
getServos() (Java method), 209	getUp() (Java method), 176
getShielded() (Java method), 186	getUT() (Java method), 156, 203
getSideslipAngle() (Java method), 172	getVacuumSpecificImpulse() (Java method), 160, 189
getSignalDelay() (Java method), 205	getValue() (Java method), 195
getSignalDelayToGroundStation() (Java method), 205	getVelocity() (Java method), 170
getSituation() (Java method), 159	getVerticalSpeed() (Java method), 170
getSkinTemperature() (Java method), 181	getVessel() (Java method), 180, 214
getSolarPanel() (Java method), 183	getVessels() (Java method), 155
getSolarPanels() (Java method), 180	getWarpFactor() (Java method), 156
getSpecificImpulse() (Java method), 160, 189	getWarpMode() (Java method), 156
getSpeed() (Java method), 170, 173, 209, 211	getWarpRate() (Java method), 156

getWheelSteering() (Java method), 176 getWheelThrottle() (Java method), 176 getXferOriginBody() (Java method), 214 getXferTargetBody() (Java method), 214 getYaw() (Java method), 175 getYawTorque() (Java method), 195 gimbal_limit (SpaceCenter.Engine attribute), 268, 349 gimbal_locked (SpaceCenter.Engine attribute), 268, 349 gimbal_range (SpaceCenter.Engine attribute), 267, 349 gimballed (SpaceCenter.Engine attribute), 267, 349 gravitational_parameter (SpaceCenter.CelestialBody attribute), 233, 315	Inputs(int) (Java method), 194 InStage(int) (Java method), 179 is_axis_inverted (InfernalRobotics.Servo attribute), 292, 373 is_free_moving (InfernalRobotics.Servo attribute), 291, 373 is_fuel_line (SpaceCenter.Part attribute), 257, 338 is_locked (InfernalRobotics.Servo attribute), 292, 373 is_moving (InfernalRobotics.Servo attribute), 291, 373 K KerbalAlarmClock (class in KerbalAlarmClock), 293,
H	374 KerbalAlarmClock (Java class), 213
has_action() (SpaceCenter.Module method), 261, 343 has_atmosphere (SpaceCenter.CelestialBody attribute), 235, 316 has_atmospheric_oxygen (SpaceCenter.CelestialBody attribute), 235, 317	kerbin_sea_level_specific_impulse (SpaceCenter.Engine attribute), 266, 348 kerbin_sea_level_specific_impulse (SpaceCenter.Vessel attribute), 227, 308 KillWarp (Java field), 216
has_connection (SpaceCenter.Comms attribute), 285, 366 has_connection_to_ground_station (SpaceCenter.Comms attribute), 285, 366 has_event() (SpaceCenter.Module method), 261, 342	KillWarpOnly (Java field), 216 KRPC (class in krpc), 220 KRPC (class in krpc.client), 301 krpc (krpc.Client attribute), 220
has_field() (SpaceCenter.Module method), 260, 342 has_flight_computer (SpaceCenter.Comms attribute), 284, 366 has_fuel (SpaceCenter.Engine attribute), 267, 348	krpc (krpc.client.Client attribute), 301 krpc (module), 220, 300 krpc.client (module), 300 krpc.client (package), 154
has_local_control (SpaceCenter.Comms attribute), 284, 366	krpc.stream (module), 301 krpc::Client (C++ class), 90
has_resource() (SpaceCenter.Resources method), 280, 362 has_shield (SpaceCenter.DockingPort attribute), 263, 345	krpc::connect (C++ function), 89 krpc::services::AddStream <t> (C++ class), 90 krpc::services::AddStream<t>::operator() (C++ func-</t></t>
HasAction(String) (Java method), 185 HasEvent(String) (Java method), 185 HasField(String) (Java method), 184 HasResource(String) (Java method), 201	tion), 90 krpc::services::AddStream <t>::remove (C++ function), 90 krpc::services::InfernalRobotics (C++ class), 143</t>
heading (SpaceCenter.Flight attribute), 239, 320 highlight (InfernalRobotics.Servo attribute), 290, 372 horizontal_speed (SpaceCenter.Flight attribute), 238, 320	krpc::services::InfernalRobotics::ControlGroup (C++ class), 144 krpc::services::InfernalRobotics::ControlGroup::expanded
	(C++ function), 144 krpc::services::InfernalRobotics::ControlGroup::forward_key
id (KerbalAlarmClock.Alarm attribute), 294, 376 Idle (Java field), 194 impact_tolerance (SpaceCenter.Part attribute), 255, 337 impulse (SpaceCenter.Decoupler attribute), 262, 344	(C++ function), 144 krpc::services::InfernalRobotics::ControlGroup::move_center (C++ function), 144 krpc::services::InfernalRobotics::ControlGroup::move_left
in_decouple_stage() (SpaceCenter.Parts method), 251, 333 in_stage() (SpaceCenter.Parts method), 251, 333	(C++ function), 144 krpc::services::InfernalRobotics::ControlGroup::move_next_preset (C++ function), 144
inclination (SpaceCenter.Orbit attribute), 245, 326 InDecoupleStage(int) (Java method), 179 InfernalRobotics (class in InfernalRobotics), 288, 370	krpc::services::InfernalRobotics::ControlGroup::move_prev_preset (C++ function), 145 krpc::services::InfernalRobotics::ControlGroup::move_right
InfernalRobotics (Java class), 208 inputs() (SpaceCenter.ResourceConverter method), 272, 354	(C++ function), 144 krpc::services::InfernalRobotics::ControlGroup::name (C++ function), 144

krpc::services::InfernalRobotics::ControlGroup::reverse_ke	y (C++	function), 146	
(C++ function), 144			(C++
krpc::services::InfernalRobotics::ControlGroup::servo_witl		ion), 146	
(C++ function), 144		InfernalRobotics::Servo::name	(C++
krpc::services::InfernalRobotics::ControlGroup::servos	funct	ion), 145	
(C++ function), 144	krpc::services::	InfernalRobotics::Servo::position	(C++
krpc::services::InfernalRobotics::ControlGroup::set_expand		ion), 145	`
(C++ function), 144		InfernalRobotics::Servo::set_acceler	ation
krpc::services::InfernalRobotics::ControlGroup::set_forwar	-	function), 146	
(C++ function), 144		InfernalRobotics::Servo::set_current	speed
krpc::services::InfernalRobotics::ControlGroup::set_name	-	function), 146	_ 1
(C++ function), 144		InfernalRobotics::Servo::set_highlig	ht
krpc::services::InfernalRobotics::ControlGroup::set_reverse	-	function), 145	
(C++ function), 144		InfernalRobotics::Servo::set_is_axis_	inverted
krpc::services::InfernalRobotics::ControlGroup::set_speed	-	function), 146	
(C++ function), 144		InfernalRobotics::Servo::set_is_lock	ed
krpc::services::InfernalRobotics::ControlGroup::speed	-	function), 146	···
(C++ function), 144		InfernalRobotics::Servo::set_max_pe	osition
krpc::services::InfernalRobotics::ControlGroup::stop		function), 145	osition
(C++ function), 145		InfernalRobotics::Servo::set_min_po	neition
krpc::services::InfernalRobotics::InfernalRobotics (C++	-	function), 145	osition
function), 143		InfernalRobotics::Servo::set_name	$(C \perp \perp$
krpc::services::InfernalRobotics::Servo (C++ class), 145		ion), 145	Стт
krpc::services::InfernalRobotics::Servo::acceleration		InfernalRobotics::Servo::set_speed	(C
(C++ function), 146	-	ion), 145	(CTT
krpc::services::InfernalRobotics::Servo::config_speed			(C++
(C++ function), 145		InfernalRobotics::Servo::speed ion), 145	(C++
krpc::services::InfernalRobotics::Servo::current_speed		InfernalRobotics::Servo::stop (C++	func
(C++ function), 146	tion),		Tulic-
krpc::services::InfernalRobotics::Servo::is_axis_inverted		InfernalRobotics::servo_group_with	nomo
±	-		_name
(C++ function), 146 krpc::services::InfernalRobotics::Servo::is_free_moving		function), 143	(C++
	-	InfernalRobotics::servo_groups	(C++
(C++ function), 146		ion), 143	(C + 1
krpc::services::InfernalRobotics::Servo::is_locked (C++	-	InfernalRobotics::servo_with_name	(C++
function), 146		ion), 143	
krpc::services::InfernalRobotics::Servo::is_moving (C++	-	KerbalAlarmClock (C++ class), 147	
function), 146	-	KerbalAlarmClock::Alarm (C++ o	ciass),
krpc::services::InfernalRobotics::Servo::max_config_positi		V . 1 . 1 A 1 C1 1 A 1	(C
(C++ function), 145	-	KerbalAlarmClock::Alarm::action	(C++
krpc::services::InfernalRobotics::Servo::max_position		ion), 148	c
(C++ function), 145	-	KerbalAlarmClock::Alarm::id (C++	func-
krpc::services::InfernalRobotics::Servo::min_config_position			(C
(C++ function), 145	-	KerbalAlarmClock::Alarm::margin	(C++
krpc::services::InfernalRobotics::Servo::min_position		ion), 148	(C
(C++ function), 145	-		(C++
krpc::services::InfernalRobotics::Servo::move_center		ion), 148	.
(C++ function), 146	-		(C++
krpc::services::InfernalRobotics::Servo::move_left (C++		ion), 148	
function), 146		KerbalAlarmClock::Alarm::remainir	ng
krpc::services::InfernalRobotics::Servo::move_next_preset		function), 149	(C
(C++ function), 146		KerbalAlarmClock::Alarm::remove	(C++
krpc::services::InfernalRobotics::Servo::move_prev_preset		ion), 149	(C
(C++ function), 146	_	KerbalAlarmClock::Alarm::repeat	(C++
krpc::services::InfernalRobotics::Servo::move right	funct	ion), 149	

krpc::services::KerbalAlarmClock::Alarm::repeat_period (C++ function), 149	krpc::services::KerbalAlarmClock::AlarmType::apoapsis (C++ enumerator), 149
krpc::services::KerbalAlarmClock::Alarm::set_action (C++ function), 148	krpc::services::KerbalAlarmClock::AlarmType::ascending_node (C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::set_margin	krpc::services::KerbalAlarmClock::AlarmType::closest
(C++ function), 148	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::set_name	krpc::services::KerbalAlarmClock::AlarmType::contract
(C++ function), 148	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::set_notes	krpc::services::KerbalAlarmClock::AlarmType::contract_auto
(C++ function), 149	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::set_repeat	krpc::services::KerbalAlarmClock::AlarmType::crew
(C++ function), 149	(C++ enumerator), 150
	oldrpc::services::KerbalAlarmClock::AlarmType::descending_node
(C++ function), 149	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::set_time	krpc::services::KerbalAlarmClock::AlarmType::distance
(C++ function), 148	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::set_vessel	krpc::services::KerbalAlarmClock::AlarmType::earth_time
(C++ function), 149	(C++ enumerator), 150
	_krpty:services::KerbalAlarmClock::AlarmType::launch_rendevous
(C++ function), 149	(C++ enumerator), 150
	_kopky::services::KerbalAlarmClock::AlarmType::maneuver
(C++ function), 149	(C++ enumerator), 149
	krpc::services::KerbalAlarmClock::AlarmType::maneuver_auto
function), 148	(C++ enumerator), 149
krpc::services::KerbalAlarmClock::Alarm::type (C++	krpc::services::KerbalAlarmClock::AlarmType::periapsis
function), 148	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::Alarm::vessel (C++	
function), 149	(C++ enumerator), 149
	dkrpc::services::KerbalAlarmClock::AlarmType::soi_change
(C++ function), 149	(C++ enumerator), 150
	lkrpc::services::KerbalAlarmClock::AlarmType::soi_change_auto
(C++ function), 149	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::alarm_with_name	krpc::services::KerbalAlarmClock::AlarmType::transfer
(C++ function), 147	(C++ enumerator), 150
	krpc::services::KerbalAlarmClock::AlarmType::transfer_modellec
enum), 151	(C++ enumerator), 150
krpc::services::KerbalAlarmClock::AlarmAction::do_nothi	
* * * * * * * * * * * * * * * * * * * *	function), 148
krpc::services::KerbalAlarmClock::AlarmAction::do_nothi	C+1
(C++ enumerator), 151	(C++ function), 147
krpc::services::KerbalAlarmClock::AlarmAction::kill_war	
(C++ enumerator), 151	krpc::services::KRPC::get_status (C++ function), 90
krpc::services::KerbalAlarmClock::AlarmAction::kill_war	
(C++ enumerator), 151	krpc::services::SpaceCenter (C++ class), 91
krpc::services::KerbalAlarmClock::AlarmAction::message	
(C++ enumerator), 151	tion), 91
$krpc::services::KerbalAlarmClock::AlarmAction::pause_gazeriches.$	
(C++ enumerator), 151	krpc::services::SpaceCenter::AutoPilot::disengage (C++
krpc::services::KerbalAlarmClock::alarms (C++ func-	function), 140
tion), 147	krpc::services::SpaceCenter::AutoPilot::engage (C++
krpc::services::KerbalAlarmClock::alarms_with_type	function), 140
(C++ function), 148	krpc::services::SpaceCenter::AutoPilot::error (C++ func-
krpc::services::KerbalAlarmClock::AlarmType (C++	tion), 141
enum), 149	krpc::services::SpaceCenter::AutoPilot::max_roll_speed

(C++ function), 142		rices::SpaceCenter::CelestialBody::bedrock_position
krpc::services::SpaceCenter::AutoPilot::max_rotation_spee		(C++ function), 102
(C++ function), 142		rices::SpaceCenter::CelestialBody::direction
krpc::services::SpaceCenter::AutoPilot::reference_frame		(C++ function), 104
(C++ function), 141		rices::SpaceCenter::CelestialBody::equatorial_radius
krpc::services::SpaceCenter::AutoPilot::roll_error (C++		(C++ function), 102
function), 141		rices::SpaceCenter::CelestialBody::gravitational_parameter
krpc::services::SpaceCenter::AutoPilot::roll_speed_multipl		(C++ function), 102
(C++ function), 142		rices::SpaceCenter::CelestialBody::has_atmosphere
krpc::services::SpaceCenter::AutoPilot::rotation_speed_mu		(C++ function), 103
(C++ function), 142		rices::SpaceCenter::CelestialBody::has_atmospheric_oxygen
krpc::services::SpaceCenter::AutoPilot::sas (C++ func-		(C++ function), 103
tion), 141		rices::SpaceCenter::CelestialBody::mass (C++ function), 101
krpc::services::SpaceCenter::AutoPilot::sas_mode (C++		
function), 141		rices::SpaceCenter::CelestialBody::msl_position (C++ function), 102
krpc::services::SpaceCenter::AutoPilot::set_max_roll_spee (C++ function), 142		
		rices::SpaceCenter::CelestialBody::name (C++ function), 101
krpc::services::SpaceCenter::AutoPilot::set_max_rotation_s (C++ function), 142		rices::SpaceCenter::CelestialBody::non_rotating_reference_frame
krpc::services::SpaceCenter::AutoPilot::set_pid_parameters		(C++ function), 103
(C++ function), 142		rices::SpaceCenter::CelestialBody::orbit (C++
krpc::services::SpaceCenter::AutoPilot::set_reference_fram		function), 101
(C++ function), 141		rices::SpaceCenter::CelestialBody::orbital_reference_frame
krpc::services::SpaceCenter::AutoPilot::set_roll_speed_mu		(C++ function), 104
(C++ function), 142		rices::SpaceCenter::CelestialBody::position
krpc::services::SpaceCenter::AutoPilot::set_rotation_speed	-	• • • • • • • • • • • • • • • • • • • •
(C++ function), 142		rices::SpaceCenter::CelestialBody::reference_frame
krpc::services::SpaceCenter::AutoPilot::set_sas (C++		(C++ function), 103
function), 141		rices::SpaceCenter::CelestialBody::rotation
krpc::services::SpaceCenter::AutoPilot::set_sas_mode		(C++ function), 104
(C++ function), 141		rices::SpaceCenter::CelestialBody::rotational_period
krpc::services::SpaceCenter::AutoPilot::set_target_direction		(C++ function), 102
(C++ function), 141		rices::SpaceCenter::CelestialBody::rotational_speed
krpc::services::SpaceCenter::AutoPilot::set_target_roll		(C++ function), 102
(C++ function), 141		rices::SpaceCenter::CelestialBody::satellites
krpc::services::SpaceCenter::AutoPilot::target_direction		(C++ function), 101
(C++ function), 141		rices::SpaceCenter::CelestialBody::sphere_of_influence
krpc::services::SpaceCenter::AutoPilot::target_pitch_and_h		
(C++ function), 141		rices::SpaceCenter::CelestialBody::surface_gravity
krpc::services::SpaceCenter::AutoPilot::target_roll (C++	-	(C++ function), 102
function), 141		rices::SpaceCenter::CelestialBody::surface_height
krpc::services::SpaceCenter::AutoPilot::wait (C++ func-		(C++ function), 102
tion), 140		rices::SpaceCenter::CelestialBody::surface_position
krpc::services::SpaceCenter::bodies (C++ function), 91		(C++ function), 102
krpc::services::SpaceCenter::can_rails_warp_at (C++		rices::SpaceCenter::CelestialBody::velocity
function), 92		(C++ function), 104
		rices::SpaceCenter::clear_drawing (C++ func-
101	_	tion), 94
krpc::services::SpaceCenter::CelestialBody::angular_veloc		
(C++ function), 104		91
krpc::services::SpaceCenter::CelestialBody::atmosphere_de		
(C++ function), 103		rices::SpaceCenter::Comms::has_connection
krpc::services::SpaceCenter::CelestialBody::bedrock_heigh		(C++ function), 140
(C++ function), 102		rices::SpaceCenter::Comms::has_connection_to_ground_station

(C++ function), 140	krpc::services::SpaceCenter::Control::set_gear (C++
krpc::services::SpaceCenter::Comms::has_flight_computer	
(C++ function), 139	krpc::services::SpaceCenter::Control::set_lights (C++
krpc::services::SpaceCenter::Comms::has_local_control	function), 110
(C++ function), 139	<pre>krpc::services::SpaceCenter::Control::set_pitch</pre>
krpc::services::SpaceCenter::Comms::signal_delay (C++	· · · · · · · · · · · · · · · · · · ·
function), 140	krpc::services::SpaceCenter::Control::set_rcs (C++ func-
krpc::services::SpaceCenter::Comms::signal_delay_to_group (C++ function), 140	krpc::services::SpaceCenter::Control::set_right (C++
krpc::services::SpaceCenter::Comms::signal_delay_to_vest	
(C++ function), 140	krpc::services::SpaceCenter::Control::set_roll (C++ func-
krpc::services::SpaceCenter::Control (C++ class), 110	tion), 111
krpc::services::SpaceCenter::Control::abort (C++ func-	krpc::services::SpaceCenter::Control::set_sas (C++ func-
tion), 111	tion), 110
krpc::services::SpaceCenter::Control::activate_next_stage	
(C++ function), 111	(C++ function), 110
krpc::services::SpaceCenter::Control::add_node (C++	krpc::services::SpaceCenter::Control::set_speed_mode
function), 112	(C++ function), 110
krpc::services::SpaceCenter::Control::brakes (C++ function), 110	<pre>krpc::services::SpaceCenter::Control::set_throttle (C++ function), 111</pre>
krpc::services::SpaceCenter::Control::current_stage	krpc::services::SpaceCenter::Control::set_up (C++ func-
(C++ function), 111	tion), 111
krpc::services::SpaceCenter::Control::forward (C++	krpc::services::SpaceCenter::Control::set_wheel_steering
function), 111	(C++ function), 111
krpc::services::SpaceCenter::Control::gear (C++ func-	$krpc::services::SpaceCenter::Control::set_wheel_throttle$
tion), 110	(C++ function), 111
krpc::services::SpaceCenter::Control::get_action_group	krpc::services::SpaceCenter::Control::set_yaw (C++
(C++ function), 111	function), 111
krpc::services::SpaceCenter::Control::lights (C++ function), 110	krpc::services::SpaceCenter::Control::speed_mode (C++ function), 110
krpc::services::SpaceCenter::Control::nodes (C++ func-	krpc::services::SpaceCenter::Control::throttle (C++ func-
tion), 112	tion), 111
krpc::services::SpaceCenter::Control::pitch (C++ func-	krpc::services::SpaceCenter::Control::toggle_action_group
tion), 111	(C++ function), 112
krpc::services::SpaceCenter::Control::rcs (C++ function), 110	krpc::services::SpaceCenter::Control::up (C++ function), 111
krpc::services::SpaceCenter::Control::remove_nodes (C++ function), 112	krpc::services::SpaceCenter::Control::wheel_steering (C++ function), 111
krpc::services::SpaceCenter::Control::right (C++ function), 111	krpc::services::SpaceCenter::Control::wheel_throttle (C++ function), 111
krpc::services::SpaceCenter::Control::roll (C++ func-	krpc::services::SpaceCenter::Control::yaw (C++ func-
tion), 111	tion), 111
krpc::services::SpaceCenter::Control::sas (C++ function),	krpc::services::SpaceCenter::Decoupler (C++ class), 120
110	krpc::services::SpaceCenter::Decoupler::decouple (C++
krpc::services::SpaceCenter::Control::sas_mode (C++	function), 121
function), 110	krpc::services::SpaceCenter::Decoupler::decoupled (C++
krpc::services::SpaceCenter::Control::set_abort (C++	function), 121
function), 111	krpc::services::SpaceCenter::Decoupler::impulse (C++
krpc::services::SpaceCenter::Control::set_action_group	function), 121
(C++ function), 112	krpc::services::SpaceCenter::Decoupler::part (C++ func-
krpc::services::SpaceCenter::Control::set_brakes (C++	tion), 120
function), 110	$krpc::services::SpaceCenter::DockingPort \ \ (C++ \ \ class),$
krpc::services::SpaceCenter::Control::set_forward (C++	121
function) 111	krpc··services··SpaceCenter··DockingPort··direction

(C++ function), 121 krpc::services::SpaceCenter::DockingPort::docked_part	krpc::services::SpaceCenter::Engine::gimbal_limit (C++ function), 125
	krpc::services::SpaceCenter::Engine::gimbal_locked
(C++ function), 121	
krpc::services::SpaceCenter::DockingPort::has_shield	(C++ function), 124
(C++ function), 121	krpc::services::SpaceCenter::Engine::gimbal_range (C++
krpc::services::SpaceCenter::DockingPort::name (C++	function), 124
function), 121	krpc::services::SpaceCenter::Engine::gimballed (C++
krpc::services::SpaceCenter::DockingPort::part (C++	function), 124
function), 121	krpc::services::SpaceCenter::Engine::has_fuel (C++
krpc::services::SpaceCenter::DockingPort::position (C++	function), 124
function), 121	krpc::services::SpaceCenter::Engine::kerbin_sea_level_specific_impulse
krpc::services::SpaceCenter::DockingPort::reengage_distar	
(C++ function), 121	krpc::services::SpaceCenter::Engine::max_thrust (C++
krpc::services::SpaceCenter::DockingPort::reference_frame	
(C++ function), 122	krpc::services::SpaceCenter::Engine::max_vacuum_thrust
krpc::services::SpaceCenter::DockingPort::rotation (C++	(C++ function), 124
function), 121	krpc::services::SpaceCenter::Engine::part (C++ func-
krpc::services::SpaceCenter::DockingPort::set_name	tion), 123
(C++ function), 121	krpc::services::SpaceCenter::Engine::propellant_ratios
krpc::services::SpaceCenter::DockingPort::set_shielded	(C++ function), 124
(C++ function), 121	krpc::services::SpaceCenter::Engine::propellants (C++
krpc::services::SpaceCenter::DockingPort::shielded	function), 124
(C++ function), 121	krpc::services::SpaceCenter::Engine::set_active (C++
krpc::services::SpaceCenter::DockingPort::state (C++	function), 123
function), 121	krpc::services::SpaceCenter::Engine::set_gimbal_limit
krpc::services::SpaceCenter::DockingPort::undock (C++	(C++ function), 125
function), 121	krpc::services::SpaceCenter::Engine::set_gimbal_locked
krpc::services::SpaceCenter::DockingPortState (C++	(C++ function), 125
enum), 122	krpc::services::SpaceCenter::Engine::set_thrust_limit
krpc::services::SpaceCenter::DockingPortState::docked	(C++ function), 124
(C++ enumerator), 122	krpc::services::SpaceCenter::Engine::specific_impulse
krpc::services::SpaceCenter::DockingPortState::docking	(C++ function), 124
(C++ enumerator), 122	krpc::services::SpaceCenter::Engine::throttle (C++ func-
krpc::services::SpaceCenter::DockingPortState::moving	tion), 124
(C++ enumerator), 123	krpc::services::SpaceCenter::Engine::throttle_locked
krpc::services::SpaceCenter::DockingPortState::ready	(C++ function), 124
(C++ enumerator), 122	krpc::services::SpaceCenter::Engine::thrust (C++ func-
krpc::services::SpaceCenter::DockingPortState::shielded	tion), 123
(C++ enumerator), 123	krpc::services::SpaceCenter::Engine::thrust_limit (C++
krpc::services::SpaceCenter::DockingPortState::undocking	function), 124
(C++ enumerator), 123	krpc::services::SpaceCenter::Engine::vacuum_specific_impulse
krpc::services::SpaceCenter::draw_direction (C++ func-	(C++ function), 124
tion), 93	krpc::services::SpaceCenter::far_available (C++ func-
krpc::services::SpaceCenter::draw_line (C++ function),	tion), 93
94	krpc::services::SpaceCenter::Flight (C++ class), 104
krpc::services::SpaceCenter::Engine (C++ class), 123	krpc::services::SpaceCenter::Flight::aerodynamic_force
krpc::services::SpaceCenter::Engine::active (C++ func-	(C++ function), 106
tion), 123	krpc::services::SpaceCenter::Flight::angle_of_attack
krpc::services::SpaceCenter::Engine::available_thrust	(C++ function), 107
(C++ function), 123	krpc::services::SpaceCenter::Flight::anti_normal (C++
krpc::services::SpaceCenter::Engine::can_restart (C++	function), 106
function), 124	krpc::services::SpaceCenter::Flight::anti_radial (C++
krpc::services::SpaceCenter::Engine::can_shutdown	function), 106
(C++ function), 124	krpc::services::SpaceCenter::Flight::atmosphere_density
	- · · · · · · · · · · · · · · · · · · ·

(C++ function), 106	tion), 105
krpc::services::SpaceCenter::Flight::ballistic_coefficient	krpc::services::SpaceCenter::Flight::speed_of_sound
(C++ function), 107	(C++ function), 106
krpc::services::SpaceCenter::Flight::bedrock_altitude	krpc::services::SpaceCenter::Flight::stall_fraction (C++
(C++ function), 105	function), 107
krpc::services::SpaceCenter::Flight::center_of_mass	krpc::services::SpaceCenter::Flight::static_air_temperature
(C++ function), 105	(C++ function), 107
krpc::services::SpaceCenter::Flight::direction (C++ func-	<pre>krpc::services::SpaceCenter::Flight::static_pressure (C++</pre>
tion), 105	function), 106
<pre>krpc::services::SpaceCenter::Flight::drag (C++ function),</pre>	krpc::services::SpaceCenter::Flight::surface_altitude
106	(C++ function), 105
krpc::services::SpaceCenter::Flight::drag_coefficient	krpc::services::SpaceCenter::Flight::terminal_velocity
(C++ function), 107	(C++ function), 107
krpc::services::SpaceCenter::Flight::dynamic_pressure	krpc::services::SpaceCenter::Flight::thrust_specific_fuel_consumption
(C++ function), 106	(C++ function), 108
krpc::services::SpaceCenter::Flight::elevation (C++ func-	krpc::services::SpaceCenter::Flight::total_air_temperature
tion), 105	(C++ function), 107
$krpc::services::SpaceCenter::Flight::equivalent_air_speed$	krpc::services::SpaceCenter::Flight::velocity (C++ func-
(C++ function), 107	tion), 105
krpc::services::SpaceCenter::Flight::g_force (C++ func-	krpc::services::SpaceCenter::Flight::vertical_speed (C++
tion), 105	function), 105
krpc::services::SpaceCenter::Flight::heading (C++ func-	krpc::services::SpaceCenter::g (C++ function), 91
tion), 105	krpc::services::SpaceCenter::LandingGear (C++ class),
krpc::services::SpaceCenter::Flight::horizontal_speed	125
(C++ function), 105	krpc::services::SpaceCenter::LandingGear::deployed
krpc::services::SpaceCenter::Flight::latitude (C++ func-	(C++ function), 125
tion), 105	krpc::services::SpaceCenter::LandingGear::part (C++
krpc::services::SpaceCenter::Flight::lift (C++ function),	function), 125
106	krpc::services::SpaceCenter::LandingGear::set_deployed
krpc::services::SpaceCenter::Flight::lift_coefficient (C++	(C++ function), 125
function), 107	krpc::services::SpaceCenter::LandingGear::state (C++
krpc::services::SpaceCenter::Flight::longitude (C++	function), 125
function), 105	krpc::services::SpaceCenter::LandingGearState (C++
krpc::services::SpaceCenter::Flight::mach (C++ func-	enum), 125
tion), 107	krpc::services::SpaceCenter::LandingGearState::deployed
$krpc::services::SpaceCenter::Flight::mean_altitude \ (C++$	(C++ enumerator), 125
function), 105	krpc::services::SpaceCenter::LandingGearState::deploying
$krpc::services::SpaceCenter::Flight::normal \ \ (C++\ \ func-$	(C++ enumerator), 125
tion), 105	krpc::services::SpaceCenter::LandingGearState::retracted
krpc::services::SpaceCenter::Flight::pitch (C++ func-	(C++ enumerator), 125
tion), 105	krpc::services::SpaceCenter::LandingGearState::retracting
krpc::services::SpaceCenter::Flight::prograde (C++ func-	(C++ enumerator), 125
tion), 105	krpc::services::SpaceCenter::LandingLeg (C++ class),
krpc::services::SpaceCenter::Flight::radial (C++ func-	125
tion), 106	krpc::services::SpaceCenter::LandingLeg::deployed
krpc::services::SpaceCenter::Flight::retrograde (C++	(C++ function), 125
function), 105	krpc::services::SpaceCenter::LandingLeg::part (C++
$krpc::services::SpaceCenter::Flight::roll\ (C+\!\!\!\!+ function),$	function), 125
105	krpc::services::SpaceCenter::LandingLeg::set_deployed
krpc::services::SpaceCenter::Flight::rotation (C++ func-	(C++ function), 125
tion), 105	krpc::services::SpaceCenter::LandingLeg::state (C++
krpc::services::SpaceCenter::Flight::sideslip_angle (C++	function), 125
function), 107	krpc::services::SpaceCenter::LandingLegState (C++
krpc::services::SpaceCenter::Flight::speed (C++ func-	enum), 125

krpc::services::SpaceCenter::LandingLegState::broken (C++ enumerator), 126 krpc::services::SpaceCenter::LandingLegState::deployed (C++ enumerator), 125 krpc::services::SpaceCenter::LandingLegState::deploying (C++ enumerator), 126 krpc::services::SpaceCenter::LandingLegState::repairing (C++ enumerator), 126 krpc::services::SpaceCenter::LandingLegState::retracted (C++ enumerator), 126 krpc::services::SpaceCenter::LandingLegState::retracting (C++ enumerator), 126 krpc::services::SpaceCenter::launch_vessel_from_sph (C++ function), 91 krpc::services::SpaceCenter::launch_vessel_from_vab (C++ function), 91 krpc::services::SpaceCenter::LaunchClamp (C++ class), krpc::services::SpaceCenter::LaunchClamp::part function), 126 krpc::services::SpaceCenter::LaunchClamp::release (C++ function), 126 krpc::services::SpaceCenter::Light (C++ class), 126 krpc::services::SpaceCenter::Light::active (C++ function), 126 krpc::services::SpaceCenter::Light::part (C++ function), krpc::services::SpaceCenter::Light::power_usage function), 126 krpc::services::SpaceCenter::Light::set_active (C++function), 126 krpc::services::SpaceCenter::maximum_rails_warp_factor krpc::services::SpaceCenter::Node::set_normal (C++ function), 92 krpc::services::SpaceCenter::Module (C++ class), 119 krpc::services::SpaceCenter::Module::actions (C++ function), 120 krpc::services::SpaceCenter::Module::events (C++ function), 120 krpc::services::SpaceCenter::Module::fields (C++ function), 119 krpc::services::SpaceCenter::Module::get_field (C++function), 119 krpc::services::SpaceCenter::Module::has_action (C++function), 120 krpc::services::SpaceCenter::Module::has_event (C++function), 120 krpc::services::SpaceCenter::Module::has_field (C++function), 119 krpc::services::SpaceCenter::Module::name (C++ function), 119 krpc::services::SpaceCenter::Module::part (C++ function), 119 krpc::services::SpaceCenter::Module::set_action (C++ krpc::services::SpaceCenter::Orbit::eccentric_anomaly

function), 120

krpc::services::SpaceCenter::Module::trigger_event (C++ function), 120 krpc::services::SpaceCenter::Node (C++ class), 137 krpc::services::SpaceCenter::Node::burn_vector function), 138 krpc::services::SpaceCenter::Node::delta v (C++ funckrpc::services::SpaceCenter::Node::direction (C++ function), 139 krpc::services::SpaceCenter::Node::normal (C++ function), 137 krpc::services::SpaceCenter::Node::orbit (C++ function), krpc::services::SpaceCenter::Node::orbital_reference_frame (C++ function), 139 krpc::services::SpaceCenter::Node::position (C++ function), 139 krpc::services::SpaceCenter::Node::prograde (C++ function), 137 krpc::services::SpaceCenter::Node::radial (C++ function), 137 krpc::services::SpaceCenter::Node::reference_frame (C++ function), 139 krpc::services::SpaceCenter::Node::remaining_burn_vector (C++ function), 138 krpc::services::SpaceCenter::Node::remaining_delta_v (C++ function), 138 krpc::services::SpaceCenter::Node::remove (C++ function), 138 krpc::services::SpaceCenter::Node::set_delta_v (C++function), 138 (C++function), 137 krpc::services::SpaceCenter::Node::set_prograde (C++function), 137 krpc::services::SpaceCenter::Node::set_radial (C++ function), 138 krpc::services::SpaceCenter::Node::set_ut (C++ function), 138 krpc::services::SpaceCenter::Node::time_to (C++ function), 138 krpc::services::SpaceCenter::Node::ut (C++ function), krpc::services::SpaceCenter::Orbit (C++ class), 108 krpc::services::SpaceCenter::Orbit::apoapsis (C++ function), 108 krpc::services::SpaceCenter::Orbit::apoapsis_altitude (C++ function), 108 krpc::services::SpaceCenter::Orbit::argument_of_periapsis (C++ function), 109 krpc::services::SpaceCenter::Orbit::body (C++ function),

420 Index

(C++ function), 109

krpc::services::SpaceCenter::Orbit::eccentricity (C++	tion), 126
function), 109	krpc::services::SpaceCenter::ParachuteState (C++ enum),
krpc::services::SpaceCenter::Orbit::epoch (C++ func-	127
tion), 109	krpc::services::SpaceCenter::ParachuteState::active (C++
krpc::services::SpaceCenter::Orbit::inclination (C++	enumerator), 127
function), 109	krpc::services::SpaceCenter::ParachuteState::cut (C++
krpc::services::SpaceCenter::Orbit::longitude_of_ascendin	g_node enumerator), 127
(C++ function), 109	krpc::services::SpaceCenter::ParachuteState::deployed
krpc::services::SpaceCenter::Orbit::mean_anomaly (C++	(C++ enumerator), 127
function), 109	krpc::services::SpaceCenter::ParachuteState::semi_deployed
krpc::services::SpaceCenter::Orbit::mean_anomaly_at_epo	
(C++ function), 109	krpc::services::SpaceCenter::ParachuteState::stowed
krpc::services::SpaceCenter::Orbit::next_orbit (C++	(C++ enumerator), 127
function), 110	krpc::services::SpaceCenter::Part (C++ class), 115
krpc::services::SpaceCenter::Orbit::periapsis (C++ func-	krpc::services::SpaceCenter::Part::axially_attached (C++
tion), 108	function), 116
krpc::services::SpaceCenter::Orbit::periapsis_altitude	krpc::services::SpaceCenter::Part::children (C++ func-
(C++ function), 108	tion), 115
krpc::services::SpaceCenter::Orbit::period (C++ func-	krpc::services::SpaceCenter::Part::cost (C++ function),
tion), 109	115
	krpc::services::SpaceCenter::Part::crossfeed (C++ func-
tion), 108	tion), 117
	tidurpc::services::SpaceCenter::Part::decouple_stage (C++
(C++ function), 109	function), 116
	alkrpc::services::SpaceCenter::Part::decoupler (C++ func-
(C++ function), 109	tion), 117
krpc::services::SpaceCenter::Orbit::semi_major_axis	krpc::services::SpaceCenter::Part::direction (C++ func-
(C++ function), 108	tion), 118
krpc::services::SpaceCenter::Orbit::semi_minor_axis	krpc::services::SpaceCenter::Part::docking_port (C++
(C++ function), 108	function), 117
krpc::services::SpaceCenter::Orbit::speed (C++ func-	krpc::services::SpaceCenter::Part::dry_mass (C++ func-
tion), 109	tion), 116
krpc::services::SpaceCenter::Orbit::time_to_apoapsis	krpc::services::SpaceCenter::Part::engine (C++ function),
(C++ function), 109	117
krpc::services::SpaceCenter::Orbit::time_to_periapsis	krpc::services::SpaceCenter::Part::external_temperature
(C++ function), 109	(C++ function), 116
krpc::services::SpaceCenter::Orbit::time_to_soi_change	krpc::services::SpaceCenter::Part::fuel_lines_from (C++
(C++ function), 109	function), 117
krpc::services::SpaceCenter::Parachute (C++ class), 126	krpc::services::SpaceCenter::Part::fuel_lines_to (C++
krpc::services::SpaceCenter::Parachute::deploy (C++	function), 117
function), 126	krpc::services::SpaceCenter::Part::impact_tolerance
krpc::services::SpaceCenter::Parachute::deploy_altitude	(C++ function), 116
(C++ function), 127	krpc::services::SpaceCenter::Part::is_fuel_line (C++
krpc::services::SpaceCenter::Parachute::deploy_min_press	
(C++ function), 127	krpc::services::SpaceCenter::Part::landing_gear (C++
krpc::services::SpaceCenter::Parachute::deployed (C++	function), 118
function), 126	krpc::services::SpaceCenter::Part::landing_leg (C++
krpc::services::SpaceCenter::Parachute::part (C++ func-	function), 118
tion), 126	krpc::services::SpaceCenter::Part::launch_clamp (C++
krpc::services::SpaceCenter::Parachute::set_deploy_altitud	
(C++ function), 127	krpc::services::SpaceCenter::Part::light (C++ function),
krpc::services::SpaceCenter::Parachute::set_deploy_min_p	
(C++ function), 127	krpc::services::SpaceCenter::Part::mass (C++ function),
krpc::services::SpaceCenter::Parachute::state (C++ func-	116

krpc:services:SpaceCenter:Part:max_lemperature (C++ function), 116 krpc:services:SpaceCenter:Part:max_lemperature (C++ function), 117 krpc:services:SpaceCenter:Part:max_lemperature (C++ function), 118 krpc:services:SpaceCenter:Part:mame (C++ function), 115 krpc:services:SpaceCenter:Part:mame (C++ function), 118 krpc:services:SpaceCenter:Part:parachute (C++ function), 118 krpc:services:SpaceCenter:Part:partitle (C++ function), 118 krpc:services:SpaceCenter:Part:radialor (C++ function), 118 krpc:services:SpaceCenter:Part:radialor (C++ function), 118 krpc:services:SpaceCenter:Part:radialor (C++ function), 118 krpc:services:SpaceCenter:Part:radialor (C++ function), 118 krpc:services:SpaceCenter:Part:ration_wheel (C++ function), 118 krpc:services:SpaceCenter:Part:stage (C++ function), 118 krpc:services:SpaceCenter:Part:stage (C++ function), 118 krpc:services:SpaceCenter:Part:stage (C++ function), 116 krpc:services:SpaceCenter:Part:stage (C++ function), 116 krpc:services:SpaceCenter:Part:stage (C++ function), 116 krpc:services:SpaceCenter:Part:stage (C++ function), 115 krpc:services:SpaceCenter:Part:stage (C++	krpc::services::SpaceCenter::Part::massless (C++ function), 116	krpc::services::SpaceCenter::Part::thermal_skin_mass (C++ function), 117
(C++ function), 116 krpc::services:SpaceCenter:Part::modules (C++ function), 117 krpc::services:SpaceCenter:Part::part::modules (C++ function), 118 krpc::services:SpaceCenter:Part::part::parachute (C++ function), 118 krpc::services:SpaceCenter:Part::part::parachute (C++ function), 118 krpc::services:SpaceCenter:Part::part::parachute (C++ function), 118 krpc::services:SpaceCenter:Part::pa		
kpc::services::SpaceCenter::Part::mack_remperature (C++ function), 116 krpc::services::SpaceCenter::Part::mame (C++ function), 117 krpc::services::SpaceCenter::Part::parent (C++ function), 118 krpc::services::SpaceCenter::Part::parent (C++ function), 118 krpc::services::SpaceCenter::Part::parent (C++ function), 118 krpc::services::SpaceCenter::Part::parent (C++ function), 118 krpc::services::SpaceCenter::Part::part::		
krpc::services::SpaceCenter::Part::modules (C++ function), 115 krpc::services::SpaceCenter::Part::parachute (C++ function), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radiator (C++ function), 118 krpc::services::SpaceCenter::Part::resource_marvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource_bravester (C++ function), 118 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::indecouple_stage (C++ function), 114 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::indecouple_stage (C++ function), 114 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::indecouple_stage (C++ function), 114 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::indecouple_stage (C++ function), 114 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::indecouple_stage (C++ function), 115 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::indecouple_stage (C++ function), 115 krpc::services::SpaceCenter::Part::services::SpaceCenter::Part::resource_stage (C++ function), 115 krp	krpc::services::SpaceCenter::Part::max_temperature	krpc::services::SpaceCenter::Part::title (C++ function),
krpc::services::SpaceCenter::Part::parachute (C++ function), 115 krpc::services::SpaceCenter::Part::parachute (C++ function), 118 krpc::services::SpaceCenter::Part::parachute (C++ function), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 114 krpc::services::SpaceCente	krpc::services::SpaceCenter::Part::modules (C++ func-	
tion), [18] krpc::services::SpaceCenter::Part::parent (C++ function), 115 krpc::services::SpaceCenter::Part::part::part::part::radiator (C++ function), 118 krpc::services::SpaceCenter::Part::radiator (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 116 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part:	krpc::services::SpaceCenter::Part::name (C++ function),	krpc::services::SpaceCenter::Part::vessel (C++ function),
krpc::services::SpaceCenter::Part::position (C++ function), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::radialor (C++ function), 118 krpc::services::SpaceCenter::Part::radialor (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource (C++ function), 117 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 116 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_ra	krpc::services::SpaceCenter::Part::parachute (C++ func-	krpc::services::SpaceCenter::Parts (C++ class), 113
krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource (C++ function), 117 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radi	tion), 118	krpc::services::SpaceCenter::Parts::all (C++ function),
krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::radialor (C++ function), 118 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_barvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource function, 114 krpc::services::SpaceCenter::Part::n_decouple_stage (C++ function), 114 krpc::services::SpaceCenter::Part::in_decouple_stage (C++ function), 114 krpc::services::SpaceCenter::Part::in_stage (C++ function), 114 krpc::services::SpaceCenter::Part::landing_gear (C++ function), 114 krpc::services::SpaceCenter::Part::landing_gear (C++ function), 115 krpc::services::SpaceCenter::Part::landing_legs (C++ function), 115 krpc::services::SpaceCenter::Part::lendendles_with_name (C++ function), 115 krpc::services::SpaceCenter::Part::lendendles_with_name (C++ function), 116 krpc::services::SpaceCenter::Part::lendendles_with_name (C++ function), 115 krpc::services::SpaceCenter::Part::lendendles_with_name (C++ fu	krpc::services::SpaceCenter::Part::parent (C++ function),	113
tion), 118 krpc::services::SpaceCenter::Part::radially_attached (C++ function), 116 krpc::services::SpaceCenter::Part::radiator (C++ function), 118 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource (C++ function), 118 krpc::services::SpaceCenter::Part::resource (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux	115	krpc::services::SpaceCenter::Parts::controlling (C++
krpc::services::SpaceCenter::Part::radially_attached (C++ function), 118 krpc::services::SpaceCenter::Part::radialor (C++ function), 118 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 117 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 117 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::shin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::servi	krpc::services::SpaceCenter::Part::position (C++ func-	function), 114
(C++ function), 116 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 114 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_anass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_anass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_resource_mass	tion), 118	krpc::services::SpaceCenter::Parts::decouplers (C++
krpc::services::SpaceCenter::Part::radiator (C++ function), 118 krpc::services::SpaceCenter::Part::reference frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource frame (C++ function), 114 krpc::services::SpaceCenter::Part::resource frame (C++ function), 114 krpc::services::SpaceCenter::Part::sensor (C++ function), 114 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::solar_panel (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krp	krpc::services::SpaceCenter::Part::radially_attached	
tion), 118 krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter		krpc::services::SpaceCenter::Parts::docking_port_with_name
krpc::services::SpaceCenter::Part::reaction_wheel (C++ function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource (C++ function), 117 krpc::services::SpaceCenter::Part::resource (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_radiation_flux (krpc::services::SpaceCenter::Part::radiator (C++ func-	
function), 118 krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 117 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::shin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mas (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mas (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mas (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mas (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function		
krpc::services::SpaceCenter::Part::reference_frame (C++ function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource (C++ function), 114 krpc::services::SpaceCenter::Part::resource (C++ function), 114 krpc::services::SpaceCenter::Part::resource (C++ function), 114 krpc::services::SpaceCenter::Part::resource (C++ function), 115 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::solar_panel (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::	krpc::services::SpaceCenter::Part::reaction_wheel (C++	
function), 118 krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 118 krpc::services::SpaceCenter::Part::rosources (C++ function), 118 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::slanding_legs (C++ function), 115 krpc::services::SpaceCenter::Part::landing_legs (C++ function), 115 krpc::services::SpaceCenter::Parts::landing_legs (C++ function), 115 krpc::services::SpaceCenter::Parts::landi		krpc::services::SpaceCenter::Parts::engines (C++ func-
krpc::services::SpaceCenter::Part::resource_converter (C++ function), 118 krpc::services::SpaceCenter::Part::resource harvester (C++ function), 117 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_aradiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_harvesters (C++	krpc::services::SpaceCenter::Part::reference_frame (C++	
C++ function), 118 krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::rotation (C++ function), 117 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 115 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::sloar_panel (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Par		
krpc::services::SpaceCenter::Part::resource_harvester (C++ function), 118 krpc::services::SpaceCenter::Part::resources (C++ function), 117 krpc::services::SpaceCenter::Part::resources (C++ function), 118 krpc::services::SpaceCenter::Part::scnsor (C++ function), 118 krpc::services::SpaceCenter::Part::scnsor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::therma	krpc::services::SpaceCenter::Part::resource_converter	
krpc::services::SpaceCenter::Part::resources (C++ function), 117 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::standing_legs (C++ function), 118 krpc::services::SpaceCenter::Part::standing_legs (C++ function), 118 krpc::services::SpaceCenter::Part::standing_legs (C++ function), 115 krpc::services::SpaceCenter::Part::standing_legs (C++ function), 115 krpc::services::SpaceCenter::Part::standing_legs (C++ function), 115 krpc::services::SpaceCenter::Parts::landing_legs (C++ function), 115 krpc::services::SpaceCenter::Parts::madiators (C++ function), 115 krpc::services::SpaceCenter::Parts::readiators (C++ function), 115 krpc::services::SpaceCenter::Parts::readiators (C++ functio		
krpc::services::SpaceCenter::Part::resources (C++ function), 117 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::sloar_panel (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_naveters krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter:		
tion), 117 krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Parts::launch_clamps (C++ function), 118 krpc::services::SpaceCenter::Parts::lights (C++ function), 118 krpc::services::SpaceCenter::Parts::lights (C++ function), 116 krpc::services::SpaceCenter::Parts::lights (C++ function), 116 krpc::services::SpaceCenter::Parts::modules_with_name (C++ function), 118 krpc::services::SpaceCenter::Parts::parachutes (C++ function), 116 krpc::services::SpaceCenter::Parts::parachutes (C++ function), 115 krpc::services::SpaceCenter::Parts::radiators (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_converters krpc::services::SpaceCenter::Parts::resource_converters krpc::services::SpaceCenter::Parts::resource_converters krpc::services::SpaceCenter::Parts::resource_harvesters krpc::services::SpaceCente		
krpc::services::SpaceCenter::Part::rotation (C++ function), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::solar_panel (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::tatage (C++ function), 115 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::resource_harvesters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_harvesters (C++ function), 115 krpc::services::SpaceCenter::		
tion), 118 krpc::services::SpaceCenter::Part::sensor (C++ function), 118 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 114 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Parts::radiators (C++ function), 115 krpc::services::SpaceCenter::Parts::radiators (C++ function), 115 krpc::services::SpaceCenter::Parts::reaction_wheels krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::resource_converters (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::servic		
krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::solar_panel krpc::services::SpaceCenter::Part::solar_panel krpc::services::SpaceCenter::Part::stage (C++ function), 118 krpc::services::SpaceCenter::Part::thermal_conduction, 119 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_nabs		
krpc::services::SpaceCenter::Parts::kin_temperature		
krpc::services::SpaceCenter::Part::skin_temperature (C++ function), 116 krpc::services::SpaceCenter::Part::solar_panel (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 115 krpc::services::SpaceCenter::Part::temperature (C++ function), 116 krpc::services::SpaceCenter::Part::radiators (C++ function), 115 krpc::services::SpaceCenter::Part::reaction_wheels krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_harvesters (C++ function), 117 krpc::services::SpaceCenter::Parts::resource_harvesters (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::servi		
(C++ function), 116 krpc::services::SpaceCenter::Parts::solar_panel (C++ function), 118 krpc::services::SpaceCenter::Parts::stage (C++ function), 116 krpc::services::SpaceCenter::Parts::stage (C++ function), 116 krpc::services::SpaceCenter::Parts::temperature (C++ function), 116 krpc::services::SpaceCenter::Parts::temperature (C++ function), 116 krpc::services::SpaceCenter::Parts::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::resource_harvesters (C++ function), 117 krpc::services::SpaceCenter::Parts::root (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 kr		
krpc::services::SpaceCenter::Part::solar_panel (C++ function), 118 krpc::services::SpaceCenter::Part::stage (C++ function), 115 krpc::services::SpaceCenter::Part::temperature (C++ tion), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 114 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ functi		
function), 118 krpc::services::SpaceCenter::Parts::tage (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ tion), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceC		
krpc::services::SpaceCenter::Part::stage (C++ function), 116 krpc::services::SpaceCenter::Part::temperature (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::Spa		
krpc::services::SpaceCenter::Parts::radiators (C++ funckrpc::services::SpaceCenter::Parts::reaction_wheels krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 115	<i>"</i>	
krpc::services::SpaceCenter::Part::temperature (C++ tion), 115		
function), 116 krpc::services::SpaceCenter::Parts::thermal_conduction_flux (C++ function), 115		
krpc::services::SpaceCenter::Part::thermal_conduction_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::Space		
(C++ function), 117 krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 k	· · · · · · · · · · · · · · · · · · ·	
krpc::services::SpaceCenter::Part::thermal_convection_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 115 krpc::services::SpaceCenter::Parts::resource_harvesters (C++ function), 115 krpc::services::SpaceCenter::Parts::root (C++ function), 113 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115		
(C++ function), 117 krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function		
krpc::services::SpaceCenter::Part::thermal_internal_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115		
(C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::sensors (C++ function), 115 krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ function), 114		
krpc::services::SpaceCenter::Part::thermal_mass (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Part::thermal_resource_mass (C++ function), 115 krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::thermal_resource_mass function), 114		
function), 117 krpc::services::SpaceCenter::Parts::thermal_radiation_flux (C++ function), 117 krpc::services::SpaceCenter::Parts::thermal_radiation_flux krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ function), 114		
krpc::services::SpaceCenter::Part::thermal_radiation_flux		
(C++ function), 117 krpc::services::SpaceCenter::Parts::set_controlling (C++ krpc::services::SpaceCenter::Parts::set_controlling (C++ function), 114		
krpc::services::SpaceCenter::Part::thermal_resource_mass function), 114		
	(C++ function), 117	krpc::services::SpaceCenter::Parts::solar_panels (C++

function), 115	krpc::services::SpaceCenter::ResourceConverter::active
krpc::services::SpaceCenter::Parts::with_module (C++	(C++ function), 128
function), 114	krpc::services::SpaceCenter::ResourceConverter::count
krpc::services::SpaceCenter::Parts::with_name (C++	(C++ function), 128
function), 114	krpc::services::SpaceCenter::ResourceConverter::inputs
krpc::services::SpaceCenter::Parts::with_title (C++ func-	(C++ function), 128
tion), 114	krpc::services::SpaceCenter::ResourceConverter::name
krpc::services::SpaceCenter::physics_warp_factor (C++	(C++ function), 128
function), 92	krpc::services::SpaceCenter::ResourceConverter::outputs
krpc::services::SpaceCenter::Radiator (C++ class), 127	(C++ function), 128
krpc::services::SpaceCenter::Radiator::deployed (C++	krpc::services::SpaceCenter::ResourceConverter::part
function), 127	(C++ function), 128
krpc::services::SpaceCenter::Radiator::part (C++ func-	krpc::services::SpaceCenter::ResourceConverter::start
tion), 127	(C++ function), 128
krpc::services::SpaceCenter::Radiator::set_deployed	krpc::services::SpaceCenter::ResourceConverter::state
(C++ function), 127	(C++ function), 128
krpc::services::SpaceCenter::Radiator::state (C++ func-	krpc::services::SpaceCenter::ResourceConverter::status_info
tion), 127	
	(C++ function), 128
krpc::services::SpaceCenter::RadiatorState (C++ enum),	krpc::services::SpaceCenter::ResourceConverter::stop
127	(C++ function), 128
krpc::services::SpaceCenter::RadiatorState::broken (C++	krpc::services::SpaceCenter::ResourceConverterState
enumerator), 127	(C++ enum), 129
krpc::services::SpaceCenter::RadiatorState::extended	krpc::services::SpaceCenter::ResourceConverterState::capacity
(C++ enumerator), 127	(C++ enumerator), 129
krpc::services::SpaceCenter::RadiatorState::extending	krpc::services::SpaceCenter::ResourceConverterState::idle
(C++ enumerator), 127	(C++ enumerator), 129
krpc::services::SpaceCenter::RadiatorState::retracted	krpc::services::SpaceCenter::ResourceConverterState::missing_resource
(C++ enumerator), 127	(C++ enumerator), 129
krpc::services::SpaceCenter::RadiatorState::retracting (C++ enumerator), 127	krpc::services::SpaceCenter::ResourceConverterState::running (C++ enumerator), 129
krpc::services::SpaceCenter::rails_warp_factor (C++	krpc::services::SpaceCenter::ResourceConverterState::storage_full
function), 92	(C++ enumerator), 129
<pre>krpc::services::SpaceCenter::ReactionWheel (C++ class),</pre>	krpc::services::SpaceCenter::ResourceConverterState::unknown
130	(C++ enumerator), 129
krpc::services::SpaceCenter::ReactionWheel::active (C++ function), 130	krpc::services::SpaceCenter::ResourceFlowMode (C++ enum), 137
krpc::services::SpaceCenter::ReactionWheel::broken	krpc::services::SpaceCenter::ResourceFlowMode::adjacent
(C++ function), 130	(C++ enumerator), 137
krpc::services::SpaceCenter::ReactionWheel::part (C++	
function), 130	(C++ enumerator), 137
krpc::services::SpaceCenter::ReactionWheel::pitch_torque	
(C++ function), 130	(C++ enumerator), 137
krpc::services::SpaceCenter::ReactionWheel::roll_torque	krpc::services::SpaceCenter::ResourceFlowMode::vessel
(C++ function), 130	(C++ enumerator), 137
krpc::services::SpaceCenter::ReactionWheel::set_active	krpc::services::SpaceCenter::ResourceHarvester (C++
(C++ function), 130	class), 129
krpc::services::SpaceCenter::ReactionWheel::yaw_torque	krpc::services::SpaceCenter::ResourceHarvester::active
(C++ function), 130	(C++ function), 129
krpc::services::SpaceCenter::ReferenceFrame (C++	krpc::services::SpaceCenter::ResourceHarvester::core_temperature
class), 140	(C++ function), 129
krpc::services::SpaceCenter::remote_tech_available	krpc::services::SpaceCenter::ResourceHarvester::deployed
(C++ function), 93	(C++ function), 129
krpc::services::SpaceCenter::ResourceConverter (C++ class), 128	krpc::services::SpaceCenter::ResourceHarvester::extraction_rate (C++ function), 129

krpc::services::SpaceCenter::ResourceHarvester::opti			
(C++ function), 129		krpc::services::SpaceCenter::Sensor::value (C++	func-
krpc::services::SpaceCenter::ResourceHarvester::part		tion), 130	
(C++ function), 129		1 – –	(C++
$krpc::services::SpaceCenter::ResourceHarvester::set_$			
(C++ function), 129		krpc::services::SpaceCenter::set_physics_warp_fact	or
$krpc::services::SpaceCenter::ResourceHarvester::set_$			
(C++ function), 129		krpc::services::SpaceCenter::set_rails_warp_factor	(C++
krpc::services::SpaceCenter::ResourceHarvester::states and the states are also as also as the states are also as		function), 92	
(C++ function), 129		krpc::services::SpaceCenter::set_target_body (C++	func-
krpc::services::SpaceCenter::ResourceHarvester::therefore the support of the property of the			
(C++ function), 129		krpc::services::SpaceCenter::set_target_docking_po	rt
krpc::services::SpaceCenter::Resources (C++ class),		(C++ function), 91	
	C++		(C++
function), 136		function), 91	
1 1		krpc::services::SpaceCenter::SolarPanel (C++ class)	
function), 137		krpc::services::SpaceCenter::SolarPanel::deployed	(C++
krpc::services::SpaceCenter::Resources::flow_mode		function), 130	
(C++ function), 137		krpc::services::SpaceCenter::SolarPanel::energy_flo	W
krpc::services::SpaceCenter::Resources::has_resource	e	(C++ function), 130	
(C++ function), 136		krpc::services::SpaceCenter::SolarPanel::part (C++	func-
krpc::services::SpaceCenter::Resources::max (C++ fu		tion), 130	
tion), 136		krpc::services::SpaceCenter::SolarPanel::set_deploy	ed
-	C++	(C++ function), 130	
function), 136		krpc::services::SpaceCenter::SolarPanel::state	(C++
$krpc::services::SpaceCenter::SASMode\ (C++\ enum),$		function), 130	
$krpc::services::SpaceCenter::SASMode::anti_normal$		krpc::services::SpaceCenter::SolarPanel::sun_expos	ure
(C++ enumerator), 112		(C++ function), 131	
krpc::services::SpaceCenter::SASMode::anti_radial		1	(C++
(C++ enumerator), 112		enum), 131	
krpc::services::SpaceCenter::SASMode::anti_target		krpc::services::SpaceCenter::SolarPanelState::broke	en
(C++ enumerator), 113		(C++ enumerator), 131	
krpc::services::SpaceCenter::SASMode::maneuver (0	C++	krpc::services::SpaceCenter::SolarPanelState::extender	ded
enumerator), 112		(C++ enumerator), 131	
1 1	C++	krpc::services::SpaceCenter::SolarPanelState::exten	ding
enumerator), 112	_	(C++ enumerator), 131	
krpc::services::SpaceCenter::SASMode::prograde (0	C++		ted
enumerator), 112	_	(C++ enumerator), 131	
	C++	krpc::services::SpaceCenter::SolarPanelState::retrac	eting
enumerator), 112	~	(C++ enumerator), 131	
krpc::services::SpaceCenter::SASMode::retrograde (0	C++	krpc::services::SpaceCenter::SpaceCenter (C++	func-
enumerator), 112		tion), 91	
krpc::services::SpaceCenter::SASMode::stability_ass	sist	krpc::services::SpaceCenter::SpeedMode (C++ en	num),
(C++ enumerator), 112		113	<i>(</i> G
krpc::services::SpaceCenter::SASMode::target (C++ e merator), 112	enu-	krpc::services::SpaceCenter::SpeedMode::orbit enumerator), 113	(C++
krpc::services::SpaceCenter::Sensor (C++ class), 130		krpc::services::SpaceCenter::SpeedMode::surface	(C++
krpc::services::SpaceCenter::Sensor::active (C++ fu		enumerator), 113	
tion), 130		krpc::services::SpaceCenter::SpeedMode::target	(C++
krpc::services::SpaceCenter::Sensor::part (C++ functi		enumerator), 113	
130		krpc::services::SpaceCenter::target_body (C++ func	tion),
krpc::services::SpaceCenter::Sensor::power_usage (91	
function), 130		krpc::services::SpaceCenter::target_docking_port	(C++
krpc::services::SpaceCenter::Sensor::set_active (0	C++	function), 91	

krpc::services::SpaceCenter::target_vessel (C++ tion), 91	func-	krpc::services::SpaceCenter::Vessel::rotation (C++ function), 100
krpc::services::SpaceCenter::transform_direction function), 93	(C++	krpc::services::SpaceCenter::Vessel::set_name (C++ function), 94
krpc::services::SpaceCenter::transform_position function), 92	(C++	krpc::services::SpaceCenter::Vessel::set_target (C++ function), 95
krpc::services::SpaceCenter::transform_rotation function), 93	(C++	krpc::services::SpaceCenter::Vessel::set_type (C++ function), 94
krpc::services::SpaceCenter::transform_velocity function), 93	(C++	krpc::services::SpaceCenter::Vessel::situation (C++ function), 94
krpc::services::SpaceCenter::ut (C++ function), 91 krpc::services::SpaceCenter::Vessel (C++ class), 94	ı	krpc::services::SpaceCenter::Vessel::specific_impulse (C++ function), 96
krpc::services::SpaceCenter::Vessel::angular_veloc (C++ function), 100		krpc::services::SpaceCenter::Vessel::surface_reference_frame (C++ function), 98
krpc::services::SpaceCenter::Vessel::auto_pilot function), 95	(C++	krpc::services::SpaceCenter::Vessel::surface_velocity_reference_frame (C++ function), 98
krpc::services::SpaceCenter::Vessel::available_thru (C++ function), 95	st	krpc::services::SpaceCenter::Vessel::target (C++ function), 95
krpc::services::SpaceCenter::Vessel::comms (C++ tion), 95	func-	krpc::services::SpaceCenter::Vessel::thrust (C++ function), 95
tion), 95		krpc::services::SpaceCenter::Vessel::type (C++ function), 94
krpc::services::SpaceCenter::Vessel::direction function), 100		krpc::services::SpaceCenter::Vessel::vacuum_specific_impulse (C++ function), 96
krpc::services::SpaceCenter::Vessel::dry_mass function), 95	(C++	krpc::services::SpaceCenter::Vessel::velocity (C++ function), 100
krpc::services::SpaceCenter::Vessel::flight (C++ tion), 94	func-	krpc::services::SpaceCenter::vessels (C++ function), 91 krpc::services::SpaceCenter::VesselSituation (C++
krpc::services::SpaceCenter::Vessel::kerbin_sea_lev	vel_spec	
(C++ function), 96		krpc::services::SpaceCenter::VesselSituation::docked
krpc::services::SpaceCenter::Vessel::mass (C++ tion), 95	func-	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselSituation::escaping
krpc::services::SpaceCenter::Vessel::max_thrust function), 95	(C++	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselSituation::flying
krpc::services::SpaceCenter::Vessel::max_vacuum_ (C++ function), 96	_thrust	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselSituation::landed
krpc::services::SpaceCenter::Vessel::met (C++ fune 94	ction),	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselSituation::orbiting
krpc::services::SpaceCenter::Vessel::name (C++ tion), 94	func-	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselSituation::pre_launch
krpc::services::SpaceCenter::Vessel::orbit (C++ tion), 95	func-	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselSituation::splashed
krpc::services::SpaceCenter::Vessel::orbital_referer (C++ function), 96	nce_frar	
krpc::services::SpaceCenter::Vessel::parts (C++ tion), 95	func-	(C++ enumerator), 101 krpc::services::SpaceCenter::VesselType (C++ enum),
krpc::services::SpaceCenter::Vessel::position (C++ tion), 100	func-	100 krpc::services::SpaceCenter::VesselType::base (C++ enu-
krpc::services::SpaceCenter::Vessel::reference_fram	ne	merator), 101
(C++ function), 96 krpc::services::SpaceCenter::Vessel::resources	(C++	krpc::services::SpaceCenter::VesselType::debris (C++ enumerator), 101
function), 95 krpc::services::SpaceCenter::Vessel::resources_in_c	decoupl	
(C++ function), 95		krpc::services::SpaceCenter::VesselType::probe (C++

enumerator), 101 krpc::services::SpaceCenter::VesselType::rover (C++	LandingLegState.deployed (in module SpaceCenter), 269, 351
enumerator), 101	LandingLegState.deploying (in module SpaceCenter),
krpc::services::SpaceCenter::VesselType::ship (C++ enu-	269, 351
merator), 100	LandingLegState.repairing (in module SpaceCenter),
krpc::services::SpaceCenter::VesselType::station (C++	269, 351
enumerator), 101	LandingLegState.retracted (in module SpaceCenter), 269,
krpc::services::SpaceCenter::warp_factor (C++ function), 92	351 LandingLegState.retracting (in module SpaceCenter),
krpc::services::SpaceCenter::warp_mode (C++ function),	269, 351 latitude (SpaceCenter.Flight attribute), 238, 319
krpc::services::SpaceCenter::warp_rate (C++ function), 92	launch_clamp (SpaceCenter.Part attribute), 258, 340 launch_clamps (SpaceCenter.Parts attribute), 252, 334
krpc::services::SpaceCenter::warp_to (C++ function), 92	launch_vessel_from_sph() (SpaceCenter.SpaceCenter
krpc::services::SpaceCenter::WarpMode (C++ enum), 94	static method), 221, 302
krpc::services::SpaceCenter::WarpMode::none (C++ enu-	launch_vessel_from_vab() (SpaceCenter.SpaceCenter
merator), 94	static method), 221, 302
krpc::services::SpaceCenter::WarpMode::physics (C++	LaunchClamp (class in SpaceCenter), 269, 351
enumerator), 94	LaunchClamp (Java class), 191
krpc::services::SpaceCenter::WarpMode::rails (C++ enu-	LaunchRendevous (Java field), 215
merator), 94	LaunchVesselFromSPH(String) (Java method), 156
KRPCClass (built-in class), 385	LaunchVesselFromVAB(String) (Java method), 155
KRPCEnum (built-in class), 387	lift (SpaceCenter.Flight attribute), 240, 322
KRPCMethod (built-in class), 386	lift_coefficient (SpaceCenter.Flight attribute), 242, 324
KRPCProcedure (built-in class), 385	Light (class in SpaceCenter), 270, 351
KRPCProperty (built-in class), 386	Light (Java class), 191
KRPCService (built-in class), 384	light (SpaceCenter.Part attribute), 258, 340
1	lights (SpaceCenter.Control attribute), 247, 328
L	lights (SpaceCenter.Parts attribute), 252, 334
Landed (Java field), 166	longitude (SpaceCenter.Flight attribute), 238, 319
Lander (Java field), 165	longitude_of_ascending_node (SpaceCenter.Orbit at-
landing_gear (SpaceCenter.Part attribute), 257, 339	tribute), 245, 326
landing_gear (SpaceCenter.Parts attribute), 252, 334	M
landing_leg (SpaceCenter.Part attribute), 258, 339	
landing_legs (SpaceCenter.Parts attribute), 252, 334	mach (SpaceCenter.Flight attribute), 241, 322
LandingGear (class in SpaceCenter), 268, 350	Maneuver (Java field), 177, 215
LandingGear (Java class), 190	ManeuverAuto (Java field), 215
LandingGearState (class in SpaceCenter), 268, 350	margin (KerbalAlarmClock.Alarm attribute), 294, 375
LandingGearState (Java enum), 190	mass (SpaceCenter.CelestialBody attribute), 233, 315
LandingGearState.deployed (in module SpaceCenter),	mass (SpaceCenter.Part attribute), 255, 336
268, 350 Landing CoarState deploying (in module SpaceCenter)	mass (SpaceCenter. Vessel attribute), 226, 307
LandingGearState.deploying (in module SpaceCenter),	massless (SpaceCenter.Part attribute), 255, 336
268, 350 LandingGearState.retracted (in module SpaceCenter),	max() (SpaceCenter.Resources method), 281, 362 Max(String) (Java method), 202
268, 350	max_config_position (InfernalRobotics.Servo attribute),
LandingGearState.retracting (in module SpaceCenter), 268, 350	291, 372 max_position (InfernalRobotics.Servo attribute), 291,
LandingLeg (class in SpaceCenter), 269, 350	372
LandingLeg (Java class), 190	max_roll_speed (SpaceCenter.AutoPilot attribute), 287,
LandingLegState (class in SpaceCenter), 269, 351	369
LandingLegState (Java enum), 191	max_rotation_speed (SpaceCenter.AutoPilot attribute),
LandingLegState.broken (in module SpaceCenter), 269,	287, 369
351	max_skin_temperature (SpaceCenter.Part attribute), 255,
	337

max_temperature (SpaceCenter.Part attribute), 255, 337	method), 167
max_thrust (SpaceCenter.Engine attribute), 266, 348	N.I.
max_thrust (SpaceCenter.Vessel attribute), 227, 308	N
max_vacuum_thrust (SpaceCenter.Engine attribute), 266, 348	name (InfernalRobotics.ControlGroup attribute), 289, 370
max_vacuum_thrust (SpaceCenter.Vessel attribute), 227, 308	name (InfernalRobotics.Servo attribute), 290, 372 name (KerbalAlarmClock.Alarm attribute), 294, 376
maximum_rails_warp_factor (SpaceCenter.SpaceCenter	name (SpaceCenter.CelestialBody attribute), 233, 314
attribute), 222, 303	name (SpaceCenter.DockingPort attribute), 262, 344
mean_altitude (SpaceCenter.Flight attribute), 237, 319	name (SpaceCenter.Module attribute), 259, 341
mean_anomaly (SpaceCenter.Orbit attribute), 245, 327	name (SpaceCenter.Part attribute), 253, 335
$mean_anomaly_at_epoch \ (SpaceCenter.Orbit \ attribute),$	name (SpaceCenter. Vessel attribute), 225, 306
245, 326	name() (SpaceCenter.ResourceConverter method), 272,
MessageOnly (Java field), 216	353
met (SpaceCenter. Vessel attribute), 225, 306	Name(int) (Java method), 193
min_config_position (InfernalRobotics.Servo attribute),	names (SpaceCenter.Resources attribute), 280, 362
290, 372	newInstance() (Java method), 154
min_position (InfernalRobotics.Servo attribute), 291, 372 MissingResource (Java field), 194	newInstance(String) (Java method), 154
Module (class in SpaceCenter), 259, 341	newInstance(String, InetAddress) (Java method), 154
Module (Java class), 184	newInstance(String, InetAddress, int, int) (Java method),
modules (SpaceCenter.Part attribute), 257, 339	newInstance(String, String) (Java method), 154
modules_with_name() (SpaceCenter.Parts method), 251,	newInstance(String, String) (Java method), 154
333	next_orbit (SpaceCenter.Orbit attribute), 246, 327
ModulesWithName(String) (Java method), 179	Node (class in SpaceCenter), 282, 363
$move_center() \ (Infernal Robotics. Control Group \ method),$	Node (Java class), 202
290, 371	nodes (SpaceCenter.Control attribute), 249, 330
move_center() (InfernalRobotics.Servo method), 292, 373	non_rotating_reference_frame (SpaceCenter.CelestialBody attribute), 235, 317
move_left() (InfernalRobotics.ControlGroup method),	None (Java field), 159, 202
290, 371	Normal (Java field), 177
move_left() (InfernalRobotics.Servo method), 292, 373	normal (SpaceCenter.Flight attribute), 239, 321
move_next_preset() (InfernalRobotics.ControlGroup	normal (SpaceCenter.Node attribute), 282, 363
method), 290, 371	notes (KerbalAlarmClock.Alarm attribute), 294, 376
move_next_preset() (InfernalRobotics.Servo method), 292, 374	0
$move_prev_preset() \qquad (Infernal Robotics. Control Group$	optimum_core_temperature (SpaceCen-
method), 290, 371	ter.ResourceHarvester attribute), 274, 355
move_prev_preset() (InfernalRobotics.Servo method),	Orbit (class in SpaceCenter), 243, 324
292, 374	Orbit (Java class), 173
move_right() (InfernalRobotics.ControlGroup method),	Orbit (Java field), 177
290, 371 move_right() (InfernalRobotics.Servo method), 292, 373	orbit (SpaceCenter.CelestialBody attribute), 233, 315
move_to() (InfernalRobotics.Servo method), 292, 374	orbit (SpaceCenter.Node attribute), 283, 365
MoveCenter() (Java method), 210, 211	orbit (SpaceCenter. Vessel attribute), 225, 306
MoveLeft() (Java method), 210, 211	orbital_reference_frame (SpaceCenter.CelestialBody at-
MoveNextPreset() (Java method), 210, 211	tribute), 236, 318 orbital_reference_frame (SpaceCenter.Node attribute),
MovePrevPreset() (Java method), 210, 211	283, 365
MoveRight() (Java method), 210, 211	orbital_reference_frame (SpaceCenter.Vessel attribute),
MoveTo(float, float) (Java method), 212	228, 309
Moving (Java field), 188	Orbiting (Java field), 166
msl_position() (SpaceCenter.CelestialBody method), 234, 316	outputs() (SpaceCenter.ResourceConverter method), 273, 354
MSLPosition(double, double, ReferenceFrame) (Java	Outputs(int) (Java method), 194

P	Position(ReferenceFrame) (Java method), 163, 168, 183,
Parachute (class in SpaceCenter), 270, 352	186, 204
Parachute (Java class), 192	power_usage (SpaceCenter.Light attribute), 270, 351
parachute (SpaceCenter.Part attribute), 258, 340	power_usage (SpaceCenter.Sensor attribute), 275, 357
parachutes (SpaceCenter.Parts attribute), 252, 334	PreLaunch (Java field), 166
ParachuteState (class in SpaceCenter), 271, 352	Probe (Java field), 165
ParachuteState (Java enum), 192	Prograde (Java field), 177
ParachuteState.active (in module SpaceCenter), 271, 352	prograde (SpaceCenter.Flight attribute), 239, 321
ParachuteState.cut (in module SpaceCenter), 271, 352	prograde (SpaceCenter.Node attribute), 282, 363
ParachuteState.deployed (in module SpaceCenter), 271, 352	propellant_ratios (SpaceCenter.Engine attribute), 267, 348
ParachuteState.semi_deployed (in module SpaceCenter), 271, 352	propellants (SpaceCenter.Engine attribute), 267, 348
ParachuteState.stowed (in module SpaceCenter), 271, 352	Q
parent (SpaceCenter.Part attribute), 253, 335	Quaternion (C++ class), 143
Part (class in SpaceCenter), 253, 335	Quaternion (class in SpaceCenter), 288, 369
Part (Java class), 180	_
part (SpaceCenter.Decoupler attribute), 262, 343	R
part (SpaceCenter.DockingPort attribute), 262, 344	Radial (Java field), 177
part (SpaceCenter.Engine attribute), 265, 347	radial (SpaceCenter.Flight attribute), 239, 321
part (SpaceCenter.LandingGear attribute), 268, 350	radial (SpaceCenter.Node attribute), 282, 364
part (SpaceCenter.LandingLeg attribute), 269, 350	radially_attached (SpaceCenter.Part attribute), 254, 336
part (SpaceCenter.LaunchClamp attribute), 269, 351	Radiator (class in SpaceCenter), 271, 353
part (SpaceCenter.Light attribute), 270, 351	Radiator (Java class), 192
part (SpaceCenter.Module attribute), 260, 341	radiator (SpaceCenter.Part attribute), 258, 340
part (SpaceCenter.Parachute attribute), 270, 352	radiators (SpaceCenter.Parts attribute), 252, 334
part (SpaceCenter.Radiator attribute), 271, 353	RadiatorState (class in SpaceCenter), 271, 353
part (SpaceCenter.ReactionWheel attribute), 274, 356	RadiatorState (Java enum), 192
part (SpaceCenter.ResourceConverter attribute), 272, 353	RadiatorState.broken (in module SpaceCenter), 271, 353
part (SpaceCenter.ResourceHarvester attribute), 273, 355	RadiatorState.extended (in module SpaceCenter), 271,
part (SpaceCenter.Sensor attribute), 275, 356	353
part (SpaceCenter.SolarPanel attribute), 275, 357	RadiatorState.extending (in module SpaceCenter), 271,
Parts (class in SpaceCenter), 250, 332	353
Parts (Java class), 178	RadiatorState.retracted (in module SpaceCenter), 271,
parts (SpaceCenter.Vessel attribute), 226, 307	353
PauseGame (Java field), 216	RadiatorState.retracting (in module SpaceCenter), 271,
Periapsis (Java field), 215	353
periapsis (SpaceCenter.Orbit attribute), 243, 325	radius (SpaceCenter.Orbit attribute), 244, 325
periapsis_altitude (SpaceCenter.Orbit attribute), 243, 325	Rails (Java field), 159
period (SpaceCenter.Orbit attribute), 244, 326	rails_warp_factor (SpaceCenter.SpaceCenter attribute),
Physics (Java field), 159	222, 303
physics_warp_factor (SpaceCenter.SpaceCenter at-	Raw (Java field), 215
tribute), 222, 303	rcs (SpaceCenter.Control attribute), 247, 328
pitch (SpaceCenter.Control attribute), 247, 329	reaction_wheel (SpaceCenter.Part attribute), 258, 340
pitch (SpaceCenter.Flight attribute), 239, 320	reaction_wheels (SpaceCenter.Parts attribute), 253, 334
pitch_torque (SpaceCenter.ReactionWheel attribute),	ReactionWheel (class in SpaceCenter), 274, 356
274, 356	ReactionWheel (Java class), 195
position (InfernalRobotics.Servo attribute), 290, 372	Ready (Java field), 188
position() (SpaceCenter.CelestialBody method), 236, 318	reengage_distance (SpaceCenter.DockingPort attribute),
position() (SpaceCenter.DockingPort method), 263, 345	263, 345
position() (SpaceCenter.Node method), 284, 365	reference_frame (SpaceCenter.AutoPilot attribute), 286,
position() (SpaceCenter.Part method), 259, 340	368
position() (SpaceCenter.Vessel method), 230, 313	reference_frame (SpaceCenter.CelestialBody attribute),
	235 317

reference_frame (SpaceCenter.DockingPort attribute), 264, 345	ResourceConverterState.running (in module SpaceCenter), 273, 354
reference_frame (SpaceCenter.Node attribute), 283, 365	ResourceConverterState.storage_full (in module Space-
reference_frame (SpaceCenter.Part attribute), 259, 341	Center), 273, 354
reference_frame (SpaceCenter. Vessel attribute), 227, 308	ResourceConverterState.unknown (in module SpaceCen-
reference_plane_direction() (SpaceCenter.Orbit static	ter), 273, 355
method), 245, 327	ResourceFlowMode (class in SpaceCenter), 281, 363
reference_plane_normal() (SpaceCenter.Orbit static	ResourceFlowMode (Java enum), 202
method), 245, 327 ReferenceFrame (class in SpaceCenter), 285, 367	ResourceFlowMode.adjacent (in module SpaceCenter), 281, 363
ReferenceFrame (Java class), 205	ResourceFlowMode.none (in module SpaceCenter), 281,
ReferencePlaneDirection(ReferenceFrame) (Java	363
method), 174 Reference Plane Normal (Reference Frame) (Java method)	ResourceFlowMode.stage (in module SpaceCenter), 281, 363
ReferencePlaneNormal(ReferenceFrame) (Java method),	
174	ResourceFlowMode.vessel (in module SpaceCenter),
Release() (Java method), 191	281, 363
release() (SpaceCenter.LaunchClamp method), 269, 351	ResourceHarvester (class in SpaceCenter), 273, 355
remaining (KerbalAlarmClock.Alarm attribute), 294, 376	ResourceHarvester (Java class), 194
remaining_burn_vector() (SpaceCenter.Node method),	Resources (class in SpaceCenter), 280, 362
283, 364	Resources (Java class), 201
remaining_delta_v (SpaceCenter.Node attribute), 282,	resources (SpaceCenter.Part attribute), 256, 338
364	resources (SpaceCenter.Vessel attribute), 226, 307
RemainingBurnVector(ReferenceFrame) (Java method), 203	resources_in_decouple_stage() (SpaceCenter.Vessel method), 226, 307
remote_tech_available (SpaceCenter.SpaceCenter attribute), 224, 305	ResourcesInDecoupleStage(int, boolean) (Java method), 160
RemoteObject (Java class), 155	Retracted (Java field), 190, 191, 193, 196
Remove() (Java method), 204, 214	Retracting (Java field), 190, 191, 193, 196
remove() (Java method), 155	Retrograde (Java field), 177
remove() (Kerbal Alarm Clock. Alarm method), 295, 377	retrograde (SpaceCenter.Flight attribute), 239, 321
remove() (krpc.stream.Stream method), 301	reverse_key (InfernalRobotics.ControlGroup attribute),
remove() (SpaceCenter.Node method), 283, 365	289, 371
remove_nodes() (SpaceCenter.Control method), 249, 330	right (SpaceCenter.Control attribute), 248, 329
RemoveNodes() (Java method), 177	roll (SpaceCenter.Control attribute), 247, 329
Repairing (Java field), 191	roll (SpaceCenter.Flight attribute), 239, 320
repeat (KerbalAlarmClock.Alarm attribute), 294, 376	roll_error (SpaceCenter.AutoPilot attribute), 286, 368
repeat_period (KerbalAlarmClock.Alarm attribute), 295, 376	roll_speed_multiplier (SpaceCenter.AutoPilot attribute), 287, 369
resource_converter (SpaceCenter.Part attribute), 258, 340	roll_torque (SpaceCenter.ReactionWheel attribute), 275,
resource_converters (SpaceCenter.Parts attribute), 252,	356
334	root (SpaceCenter.Parts attribute), 250, 332
resource_harvester (SpaceCenter.Part attribute), 258, 340	rotation (SpaceCenter.Flight attribute), 238, 320
resource_harvesters (SpaceCenter.Parts attribute), 253,	rotation() (SpaceCenter.CelestialBody method), 237, 318
334	rotation() (SpaceCenter.DockingPort method), 263, 345
ResourceConverter (class in SpaceCenter), 272, 353	rotation() (SpaceCenter.Part method), 259, 341
ResourceConverter (Java class), 193	rotation() (SpaceCenter. Vessel method), 232, 313
ResourceConverterState (class in SpaceCenter), 273, 354	Rotation(ReferenceFrame) (Java method), 165, 169, 183,
ResourceConverterState (Java enum), 194	187
ResourceConverterState.capacity (in module SpaceCenter), 273, 354	rotation_speed_multiplier (SpaceCenter.AutoPilot attribute), 287, 369
ResourceConverterState.idle (in module SpaceCenter), 273, 354	rotational_period (SpaceCenter.CelestialBody attribute), 233, 315
ResourceConverterState.missing_resource (in module	rotational_speed (SpaceCenter.CelestialBody attribute),
SpaceCenter), 273, 354	234, 315

Rover (Java field), 165 Running (Java field), 194	SetActionGroup(int, boolean) (Java method), 176 setActive(boolean) (Java method), 188, 191, 194, 195
S	setActiveVessel(Vessel) (Java method), 155 setBrakes(boolean) (Java method), 175
	setControlling(Part) (Java method), 178
sas (SpaceCenter.AutoPilot attribute), 287, 368	setCurrentSpeed(float) (Java method), 211
sas (SpaceCenter.Control attribute), 246, 328	setDeltaV(float) (Java method), 203
sas_mode (SpaceCenter.AutoPilot attribute), 287, 368	setDeployAltitude(float) (Java method), 192
sas_mode (SpaceCenter.Control attribute), 246, 328	setDeployed(boolean) (Java method), 190–192, 194, 196
SASMode (class in SpaceCenter), 249, 330	setDeployMinPressure(float) (Java method), 192
SASMode (Java enum), 177	setExpanded(boolean) (Java method), 209
SASMode.anti_normal (in module SpaceCenter), 249,	setForward(float) (Java method), 176
331	setForwardKey(String) (Java method), 209
SASMode.anti_radial (in module SpaceCenter), 249, 331	setGear(boolean) (Java method), 175
SASMode.anti_target (in module SpaceCenter), 249, 331	setGimbalLimit(float) (Java method), 190
SASMode.maneuver (in module SpaceCenter), 249, 330	setGimbalLocked(boolean) (Java method), 190
SASMode.normal (in module SpaceCenter), 249, 331	setHighlight(boolean) (Java method), 210
SASMode.prograde (in module SpaceCenter), 249, 331	setIsAxisInverted(boolean) (Java method), 211
SASMode.radial (in module SpaceCenter), 249, 331	setIsLocked(boolean) (Java method), 211
SASMode.retrograde (in module SpaceCenter), 249, 331	setLights(boolean) (Java method), 175
SASMode.stability_assist (in module SpaceCenter), 249,	setMargin(double) (Java method), 213
330	setMaxPosition(float) (Java method), 211
SASMode.target (in module SpaceCenter), 249, 331	setMaxRollSpeed(float) (Java method), 207
satellites (SpaceCenter.CelestialBody attribute), 233, 314	setMaxRotationSpeed(float) (Java method), 207
semi_major_axis (SpaceCenter.Orbit attribute), 244, 325	setMinPosition(float) (Java method), 210
semi_minor_axis (SpaceCenter.Orbit attribute), 244, 325	setName(String) (Java method), 159, 186, 209, 210, 214
SemiDeployed (Java field), 192	setNormal(float) (Java method), 202
Sensor (class in SpaceCenter), 275, 356	setNotes(String) (Java method), 214
Sensor (Java class), 195	setPhysicsWarpFactor(int) (Java method), 156
sensor (SpaceCenter.Part attribute), 258, 340	SetPIDParameters(float, float, float) (Java method), 207
sensors (SpaceCenter.Parts attribute), 253, 335	setPitch(float) (Java method), 175
Servo (class in InfernalRobotics), 290, 372	setPrograde(float) (Java method), 202
Servo (Java class), 210	setRadial(float) (Java method), 203
servo_group_with_name() (Infernal-	setRailsWarpFactor(int) (Java method), 156
Robotics.InfernalRobotics static method),	setRCS(boolean) (Java method), 175
288, 370	setReferenceFrame(ReferenceFrame) (Java method), 206
servo_groups (InfernalRobotics.InfernalRobotics at-	setRepeat(boolean) (Java method), 214
tribute), 288, 370	setRepeatPeriod(double) (Java method), 214
servo_with_name() (InfernalRobotics.ControlGroup	setReverseKey(String) (Java method), 209
method), 289, 371	setRight(float) (Java method), 176
servo_with_name() (InfernalRobotics.InfernalRobotics	setRoll(float) (Java method), 176
static method), 288, 370	setRollSpeedMultiplier(float) (Java method), 207
ServoGroupWithName(String) (Java method), 208	setRotationSpeedMultiplier(float) (Java method), 207
servos (InfernalRobotics.ControlGroup attribute), 289,	setSAS(boolean) (Java method), 175, 207
371 Samus With Name (Strings) (Java mathed) 200, 210	setSASMode(SASMode) (Java method), 175, 207
ServoWithName(String) (Java method), 209, 210	setShielded(boolean) (Java method), 186
set_action() (SpaceCenter.Module method), 261, 343 set_action_group() (SpaceCenter.Control method), 248,	setSpeed(float) (Java method), 209, 211
set_action_group() (spaceCenter.Control method), 248, 330	setSpeedMode(SpeedMode) (Java method), 175
	setTarget(Vessel) (Java method), 159
set_pid_parameters() (SpaceCenter.AutoPilot method),	setTargetBody(CelestialBody) (Java method), 155
287, 369 set Abort (boolean) (Java method), 175	setTargetDirection(org.javatuples.Triplet) (Java method),
setAbort(boolean) (Java method), 175 setAcceleration(float) (Java method), 211	206
setAction(AlarmAction) (Java method), 213	setTargetDockingPort(DockingPort) (Java method), 155
SetAction(String, boolean) (Java method), 185	setTargetRoll(float) (Java method), 206

setTargetVessel(Vessel) (Java method), 155	speed (SpaceCenter.Flight attribute), 238, 320
setThrottle(float) (Java method), 175	speed (SpaceCenter.Orbit attribute), 244, 326
setThrustLimit(float) (Java method), 189	speed_mode (SpaceCenter.Control attribute), 246, 328
setTime(double) (Java method), 214	speed_of_sound (SpaceCenter.Flight attribute), 241, 322
setType(VesselType) (Java method), 159	SpeedMode (class in SpaceCenter), 249, 331
setUp(float) (Java method), 176	SpeedMode (Java enum), 177
setUT(double) (Java method), 203	SpeedMode.orbit (in module SpaceCenter), 249, 331
setVessel(SpaceCenter.Vessel) (Java method), 214	SpeedMode.surface (in module SpaceCenter), 249, 331
setWheelSteering(float) (Java method), 176	SpeedMode.target (in module SpaceCenter), 250, 331
setWheelThrottle(float) (Java method), 176	sphere_of_influence (SpaceCenter.CelestialBody at-
setXferOriginBody(SpaceCenter.CelestialBody) (Java	tribute), 235, 316
method), 214	Splashed (Java field), 166
setXferTargetBody(SpaceCenter.CelestialBody) (Java	Stability Assist (Java field), 177
method), 214	Stage (Java field), 202
setYaw(float) (Java method), 176	stage (SpaceCenter.Part attribute), 254, 336
Shielded (Java field), 188	stall_fraction (SpaceCenter.Flight attribute), 242, 323
shielded (SpaceCenter.DockingPort attribute), 263, 345	start() (SpaceCenter. ResourceConverter method), 272,
Ship (Java field), 165	354
sideslip_angle (SpaceCenter.Flight attribute), 241, 323	Start(int) (Java method), 193
signal_delay (SpaceCenter.Comms attribute), 285, 366	state (SpaceCenter.DockingPort attribute), 263, 344
	state (SpaceCenter.LandingGear attribute), 268, 350
signal_delay_to_ground_station (SpaceCenter.Comms at-	· · ·
tribute), 285, 367	state (SpaceCenter.LandingLeg attribute), 269, 350
signal_delay_to_vessel() (SpaceCenter.Comms method),	state (SpaceCenter.Parachute attribute), 270, 352
285, 367	state (SpaceCenter.Radiator attribute), 271, 353
SignalDelayToVessel(Vessel) (Java method), 205	state (SpaceCenter.ResourceHarvester attribute), 273, 355
situation (SpaceCenter. Vessel attribute), 225, 306	state (SpaceCenter.SolarPanel attribute), 275, 357
skin_temperature (SpaceCenter.Part attribute), 255, 337	state() (SpaceCenter.ResourceConverter method), 272,
SOIChange (Java field), 215	354
SOIChangeAuto (Java field), 216	State(int) (Java method), 193
solar_panel (SpaceCenter.Part attribute), 259, 340	static_air_temperature (SpaceCenter.Flight attribute),
solar_panels (SpaceCenter.Parts attribute), 253, 335	242, 323
SolarPanel (class in SpaceCenter), 275, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method),
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276,	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method),
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276,	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276,	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276,	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272,
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276,	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 357	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 358 SpaceCenter (class in SpaceCenter), 220, 302	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301 Stream (Java class), 155
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 358 SpaceCenter (class in SpaceCenter), 220, 302 SpaceCenter (GameScene attribute), 388	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301 Stream (Java class), 155 stream() (krpc.client.Client method), 300
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 358 SpaceCenter (class in SpaceCenter), 220, 302 SpaceCenter (GameScene attribute), 388 SpaceCenter (Java class), 155	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301 Stream (Java class), 155 stream() (krpc.client.Client method), 300 SubOrbital (Java field), 166
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 358 SpaceCenter (class in SpaceCenter), 220, 302 SpaceCenter (GameScene attribute), 388 SpaceCenter (Java class), 155 specific_impulse (SpaceCenter.Engine attribute), 266,	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301 Stream (Java class), 155 stream() (krpc.client.Client method), 300 SubOrbital (Java field), 166 sun_exposure (SpaceCenter.SolarPanel attribute), 276,
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 358 SpaceCenter (class in SpaceCenter), 220, 302 SpaceCenter (GameScene attribute), 388 SpaceCenter (Java class), 155 specific_impulse (SpaceCenter.Engine attribute), 266, 348	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301 Stream (Java class), 155 stream() (krpc.client.Client method), 300 SubOrbital (Java field), 166 sun_exposure (SpaceCenter.SolarPanel attribute), 276, 357
SolarPanel (class in SpaceCenter), 275, 357 SolarPanel (Java class), 196 SolarPanelState (class in SpaceCenter), 276, 357 SolarPanelState (Java enum), 196 SolarPanelState.broken (in module SpaceCenter), 276, 358 SolarPanelState.extended (in module SpaceCenter), 276, 357 SolarPanelState.extending (in module SpaceCenter), 276, 358 SolarPanelState.retracted (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 357 SolarPanelState.retracting (in module SpaceCenter), 276, 358 SpaceCenter (class in SpaceCenter), 220, 302 SpaceCenter (GameScene attribute), 388 SpaceCenter (Java class), 155 specific_impulse (SpaceCenter.Engine attribute), 266, 348 specific_impulse (SpaceCenter.Vessel attribute), 227, 308	static_pressure (SpaceCenter.Flight attribute), 240, 321 Station (Java field), 165 status_info() (SpaceCenter.ResourceConverter method), 272, 354 StatusInfo(int) (Java method), 193 stop() (InfernalRobotics.ControlGroup method), 290, 372 stop() (InfernalRobotics.Servo method), 292, 374 Stop() (Java method), 210, 212 stop() (SpaceCenter.ResourceConverter method), 272, 354 Stop(int) (Java method), 193 StorageFull (Java field), 194 Stowed (Java field), 192 Stream (class in krpc.stream), 301 Stream (Java class), 155 stream() (krpc.client.Client method), 300 SubOrbital (Java field), 166 sun_exposure (SpaceCenter.SolarPanel attribute), 276, 357 Surface (Java field), 177

surface_height() (SpaceCenter.CelestialBody method), 234, 315	time_to (SpaceCenter.Node attribute), 283, 365 time_to_apoapsis (SpaceCenter.Orbit attribute), 244, 326
surface_position() (SpaceCenter.CelestialBody method), 234, 316	time_to_periapsis (SpaceCenter.Orbit attribute), 244, 326 time_to_soi_change (SpaceCenter.Orbit attribute), 246,
surface_reference_frame (SpaceCenter.Vessel attribute),	327
228, 309	title (SpaceCenter.Part attribute), 253, 335
surface_velocity_reference_frame (SpaceCenter.Vessel attribute), 230, 311	toggle_action_group() (SpaceCenter.Control method), 248, 330
SurfaceHeight(double, double) (Java method), 166	ToggleActionGroup(int) (Java method), 176
SurfacePosition(double, double, ReferenceFrame) (Java method), 167	total_air_temperature (SpaceCenter.Flight attribute), 242, 323
Т	TrackingStation (GameScene attribute), 388
	Transfer (Java field), 216
Target (Java field), 177	TransferModelled (Java field), 216
target (SpaceCenter. Vessel attribute), 225, 306	transform_direction() (SpaceCenter.SpaceCenter static
target_body (SpaceCenter.SpaceCenter attribute), 221,	method), 223, 304
302	transform_position() (SpaceCenter.SpaceCenter static
target_direction (SpaceCenter.AutoPilot attribute), 286,	method), 223, 304 transform_rotation() (SpaceCenter.SpaceCenter static
368 target_docking_port (SpaceCenter.SpaceCenter at-	method), 223, 304
tribute), 221, 302	transform_velocity() (SpaceCenter.SpaceCenter static
target_pitch_and_heading() (SpaceCenter.AutoPilot	method), 223, 305
method), 286, 368	TransformDirection(org.javatuples.Triplet, Reference-
target_roll (SpaceCenter.AutoPilot attribute), 287, 368	Frame, ReferenceFrame) (Java method),
target_vessel (SpaceCenter.SpaceCenter attribute), 221,	157
302	TransformPosition(org.javatuples.Triplet, Reference-
TargetPitchAndHeading(float, float) (Java method), 206	Frame, ReferenceFrame) (Java method),
temperature (SpaceCenter.Part attribute), 255, 337	157
terminal_velocity (SpaceCenter.Flight attribute), 241, 323	TransformRotation(org.javatuples.Quartet, Reference-
thermal_conduction_flux (SpaceCenter.Part attribute), 256, 338	Frame, ReferenceFrame) (Java method), 157
thermal_convection_flux (SpaceCenter.Part attribute),	TransformVelocity(org.javatuples.Triplet, org.javatuples.Triplet, ReferenceFrame,
256, 338 thermal_efficiency (SpaceCenter.ResourceHarvester at-	ReferenceFrame) (Java method), 158
tribute), 274, 355	trigger_event() (SpaceCenter.Module method), 261, 343
thermal_internal_flux (SpaceCenter.Part attribute), 256,	TriggerEvent(String) (Java method), 185
338	type (KerbalAlarmClock.Alarm attribute), 294, 376
thermal_mass (SpaceCenter.Part attribute), 256, 337	type (SpaceCenter. Vessel attribute), 225, 306
thermal_radiation_flux (SpaceCenter.Part attribute), 256, 338	U
thermal_resource_mass (SpaceCenter.Part attribute), 256,	Undock() (Java method), 186
338	undock() (SpaceCenter.DockingPort method), 263, 344
thermal_skin_mass (SpaceCenter.Part attribute), 256, 337	Undocking (Java field), 188
thermal_skin_to_internal_flux (SpaceCenter.Part at-	Unknown (Java field), 194
tribute), 256, 338	up (SpaceCenter.Control attribute), 248, 329
throttle (SpaceCenter Control attribute), 247, 329	ut (SpaceCenter.Node attribute), 283, 364
throttle (SpaceCenter.Engine attribute), 267, 349 throttle_locked (SpaceCenter.Engine attribute), 267, 349	ut (SpaceCenter.SpaceCenter attribute), 221, 302
thrust (SpaceCenter.Engine attribute), 266, 347	V
thrust (SpaceCenter.Vessel attribute), 226, 308	
thrust_limit (SpaceCenter.Engine attribute), 266, 348	vacuum_specific_impulse (SpaceCenter.Engine attribute), 266, 348
thrust_specific_fuel_consumption (SpaceCenter.Flight attribute), 243, 324	vacuum_specific_impulse (SpaceCenter.Vessel attribute), 227, 308
time (KerbalAlarmClock.Alarm attribute), 294, 375	value (SpaceCenter Sensor attribute) 275 357

Vector3 (C++ class), 142 Vector3 (class in SpaceCenter), 288, 369 velocity (SpaceCenter.Flight attribute), 238, 319 velocity() (SpaceCenter.CelestialBody method), 236, 318 velocity() (SpaceCenter.Part method), 259, 341 velocity() (SpaceCenter.Vessel method), 232, 313 Velocity(ReferenceFrame) (Java method), 163, 169, 183	WarpMode.physics (in module SpaceCenter), 224, 306 WarpMode.rails (in module SpaceCenter), 224, 305 WarpTo(double, float, float) (Java method), 156 wheel_steering (SpaceCenter.Control attribute), 248, 329 wheel_throttle (SpaceCenter.Control attribute), 248, 329 with_module() (SpaceCenter.Parts method), 251, 333 with_name() (SpaceCenter.Parts method), 251, 332
vertical_speed (SpaceCenter.Flight attribute), 238, 320 Vessel (class in SpaceCenter), 225, 306 Vessel (Java class), 159	with_title() (SpaceCenter.Parts method), 251, 333 WithModule(String) (Java method), 179 WithName(String) (Java method), 178
Vessel (Java field), 202	WithTitle(String) (Java method), 178
vessel (KerbalAlarmClock.Alarm attribute), 295, 376	Χ
vessel (SpaceCenter.Part attribute), 253, 335	
vessels (SpaceCenter.SpaceCenter attribute), 221, 302	xfer_origin_body (KerbalAlarmClock.Alarm attribute)
VesselSituation (class in SpaceCenter), 232, 314 VesselSituation (Java enum), 165	295, 376
VesselSituation.docked (in module SpaceCenter), 232, 314	xfer_target_body (KerbalAlarmClock.Alarm attribute) 295, 377
VesselSituation.escaping (in module SpaceCenter), 232,	Υ
314	yaw (SpaceCenter.Control attribute), 247, 329
VesselSituation.flying (in module SpaceCenter), 232, 314 VesselSituation.landed (in module SpaceCenter), 232, 314	yaw_torque (SpaceCenter.ReactionWheel attribute), 274 356
VesselSituation.orbiting (in module SpaceCenter), 233, 314	
VesselSituation.pre_launch (in module SpaceCenter), 233, 314	
VesselSituation.splashed (in module SpaceCenter), 233, 314	
VesselSituation.sub_orbital (in module SpaceCenter), 233, 314	
VesselType (class in SpaceCenter), 232, 313	
VesselType (Java enum), 165	
VesselType.base (in module SpaceCenter), 232, 314	
VesselType.debris (in module SpaceCenter), 232, 314	
VesselType.lander (in module SpaceCenter), 232, 314	
VesselType.probe (in module SpaceCenter), 232, 314	
VesselType.rover (in module SpaceCenter), 232, 314 VesselType.ship (in module SpaceCenter), 232, 314	
VesselType.station (in module SpaceCenter), 232, 314 VesselType.station (in module SpaceCenter), 232, 314	
W	
Wait() (Java method), 206	
wait() (SpaceCenter.AutoPilot method), 286, 367	
warp_factor (SpaceCenter.SpaceCenter attribute), 222,	
303	
warp_mode (SpaceCenter.SpaceCenter attribute), 221, 303	
warp_rate (SpaceCenter.SpaceCenter attribute), 222, 303	
warp_to() (SpaceCenter.SpaceCenter static method), 222, 304	
WarpMode (class in SpaceCenter), 224, 305	
WarpMode (Java enum), 159	
WarpMode.none (in module SpaceCenter), 224, 306	