Kontrol System 2

untoldwind

May 11, 2024

CONTENTS

An autopilot scripting system for Kerbal Space Program 2

CHAPTER

ONE

QUICKSTART

1.1 Installation

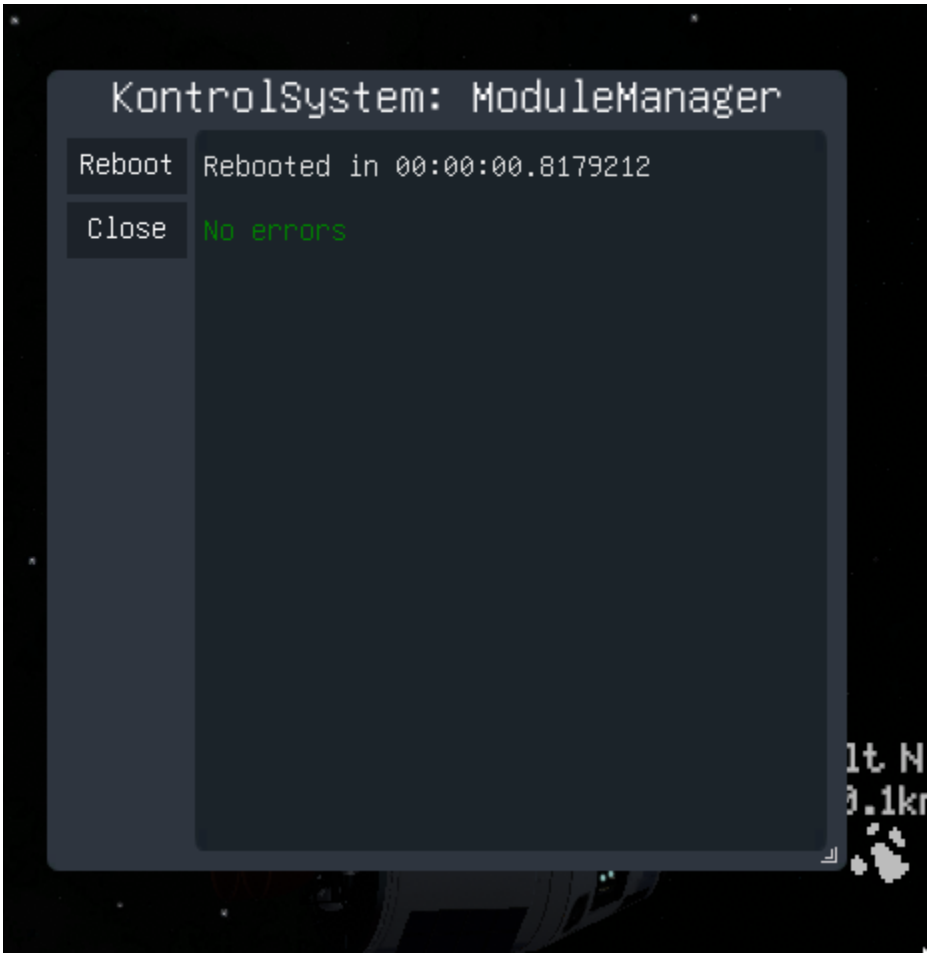
- Install BepInEx and SpaceWarp v0.4.0
  - … or just SpaceWarp with bundled BepInEx
- Unpack the one of the releases in the same directory as the KSP2 executable
  - Directory should look like:

```
KSP2_x64.exe
BepInEx
|- plugins
|  |- KontrolSystem2
|  |  |- KontrolSystemSpaceWarpMod.dll
|  |  |- ...
|  |- SpaceWarp
|  |- ...
|- ...
```

In flight there should now be an additional menu entry:

which should open the main dialog to start scripts:

The “Manage” button shows all errors if the system failed to reboot. If everything is ok it should be like this:



And there is a console where scripts can write helpful (or not so helpful) stuff to:


KontrolSystem: Console
Hello world
My vessel is: New Workspace-1
Autopilot: true StabilityAssist
UT: 2911.15641198136 pos: [-406344.582246133, 1412
UT: 2912.15641195901 pos: [-408188.219328599, 1413
UT: 2913.15641193666 pos: [-410027.559307182, 1413
UT: 2914.17641191386 pos: [-411899.239526652, 1414
UT: 2915.17641189151 pos: [-413729.841475756, 1414
>>>>> ERROR <<<<<<<<<
Module hello_world_flight terminated with:
Aborted by pilot
Clear
Close

CHAPTER

TWO

THE TO2 LANGUAGE

TO2 should be read as "twelve-O-2" resp `1202`. If you have some historic knowledge about the first moon landing you probably get the reference.

Core features:

- TO2 is strictly typed (as you do not want a typo to mess up your expensive rocket)
- It is also functional by nature, which is a nice thing to have these days
- When rebooting the Kontrol-System every script is recompiled
  - … to IL code for maximum acceptable performance
- By default every function is asynchronous so that it can run as a Unity-coroutine
  - … i.e. a running script does not block the main thread of the game (which would be rather bad)

2.1 Literal values

2.1.1 Boolean

To create a boolean value there are just the usual two keywords:

- `true`
- `false`

2.1.2 Integer

Integers can be created via:

- Decimals, e.g. `1234`
- Hexadecimals with a `0x` prefix, e.g `0x12abcd`
- Octal numbers with a `0o` prefix, e.g `0o1234`
- Binary numbers with a `0b` prefix, e.g. `0b010111`

To improve readability of large numbers an _ can be added between any digit:

- e.g. one million: `1_000_000`
- Some readable hex: `0x1111_ffff`
- Some binary: `0b1111_0000_0101_1010`

2.1.3 Float

Floating point numbers can be created via:

- Decimals, e.g. `12.34`
- Exponential/scientific notation, e.g. `1.234e-5`

2.1.4 Strings

String values can be created using double quotes, e.g.

```
"Hello world"
```

Inside a string the following escape sequences are allowed:

- `\\` to insert a backslash
- `\"` to insert a double-quote (without terminating the string itself)
- `\n` a line feed
- `\r` a carriage return
- `\t` a tabulator

2.1.5 String interpolation

In addition to static strings, c-sharp like string interpolation (i.e. strings with placeholders) are supported by starting with a `$"`, e.g.

```
const a = 1234
const b = "Hello"

$"{b} world {a}"
```

will produce the string `Hello world 1234`

Like in c-sharp, the placeholders can formatted via the `{<interpolationExpression>[,<alignment>][:<formatString>]}` pattern, e.g.

```
const c = 12.34

$"a fixed point {c,10:N4}"
```

will produce the string `a fixed point 12.3400`

Refer to https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated for a detailed description of the various formatting options.

2.2 Conditionals

2.2.1 If

If conditions can be written as

```
if ( condition ) {
    ... expressions
}
```

or inline

```
if ( condition ) expression
```

Example

```
if(fb > fa) {
    let temp = ax
    ax = bx
    bx = temp
    temp = fa
    fa = fb
    fb = temp
}
```

2.2.2 If/Else

If/else can be written as

```
if ( condition ) {
    ... expressions
} else {
    ... expressions
}
```

or inline

```
if ( condition ) expression else expression
```

This can then be combined into an if/elseif/else construct

```
if ( condition1 ) {
    ... expressions
} else if ( condition2 ) {
    ... expressions
} else {
    ... expressions
}
```

2.2.3 Ternary

TO2 does not have a ternary expression, instead `if` and `if/else` are themselves an expression and have a result. I.e. it is possible to write

```
let a = if (b < 0) 12 else 34
```

where `a` will be the integer `12` if `b < 0` is true or otherwise the integer `34`.

If the else part is omitted the result becomes an *Option*.

```
let maybe_a = if (b < 0) 12
```

here `maybe_a` will be an `Option<int>` with the value `Some(12)` if `b < 0` is true or `None` otherwise.

2.3 Entrypoints

An entrypoint is just a TO2-script file (aka. module) with a main function. These will appear in the main UI window and can be started as a process.

There are different main functions for each game situation.

2.3.1 Flight view

The entrypoint for the flight view is:

```
use { CONSOLE } from ksp::console

pub fn main_flight() -> Unit = {
    CONSOLE.print_line("Hello world")
}
```

For convenience the currently active vessel can be injected as parameter:

```
use { CONSOLE } from ksp::console
use { Vessel } from ksp::vessel

pub fn main_flight(vessel: Vessel) -> Unit = {
    CONSOLE.print_line("Hello " + vessel.name)
}
```

Entrypoints can have additional parameters beyond the active vessel. These serve as inputs to the process and can be overwritten in-game. Currently, `int`, `float` and `bool` types are supported. You should provide a default value for your extra parameters, otherwise a zero value will be chosen.

```
use { CONSOLE } from ksp::console
use { Vessel } from ksp::vessel

pub fn main_flight(vessel: Vessel, apoapsis: int = 1000, inclination: float = 1.5,␣
↪circularize: bool = true) -> Unit = {
    CONSOLE.print_line("Hello " + vessel.name)
    CONSOLE.print_line("Launching to " + apoapsis.to_string() + "km, inclination" +␣
```

(continues on next page)

(continued from previous page)

```
↪inclination.to_string() + "°. circularize=" + circularize.to_string())
}
```

NOTE all the following entrypoints are essentially untested and might not work.

2.3.2 Tracking station

The entrypoint for the Tracking station is:

```
use { CONSOLE } from ksp::console

pub fn main_tracking() -> Unit = {
    CONSOLE.print_line("Hello world")
}
```

2.3.3 VAB

The entrypoint for the VAB is:

```
use { CONSOLE } from ksp::console

pub fn main_editor() -> Unit = {
    CONSOLE.print_line("Hello world")
}
```

2.3.4 KSC

The entrypoint for the KSC is:

```
use { CONSOLE } from ksp::console

pub fn main_ksc() -> Unit = {
    CONSOLE.print_line("Hello world")
}
```

2.4 Functions

Functions are the bread and butter of every programming language, so we are trying to have as little boilerplate as possible.

2.4.1 Definition

Basic syntax

```
fn the_function_name(param1: type1, param2: type2) -> return_type = {
    ... the implementation ...
}

// or

fn the_function_name(param1: type1, param2: type2) -> return_type = some_expression
```

as mentioned in the intro, this defines an asynchronous function that is safe to use (i.e. does not block the current thread).

If best performance is required (e.g. for some numerical stuff), it is also possible to define a synchronous function

```
sync fn the_function_name(param1: type1, param2: type2) -> return_type = {
    ... the implementation ...
}

// or

sync fn the_function_name(param1: type1, param2: type2) -> return_type = some_expression
```

NOTE: Be somewhat careful with potentially long running loops here as it might impact the game.

By default functions can only be used within the same file. If a function should be callable from other scripts it is necessary to add a pub prefix:

```
pub fn the_function_name(param1: type1, param2: type2) -> return_type = {
    ... the implementation ...
}
```

resp.

```
pub sync fn the_function_name(param1: type1, param2: type2) -> return_type = {
    ... the implementation ...
}
```

Examples

```
use { CONSOLE } from ksp::console

pub fn print_hello() -> Unit = {
    CONSOLE.print_line("Hello world")
}
```

is a public function `print_hello` with no parameters and no return value that prints "Hello world" to the console.

```
fn add_three(a : int, b : float, c : float) -> float = a.to_float + b + c
```

is a local function `add_three` that takes an integer and two floats, adds them all up and returns a float.

NOTE: Adding an integer with a float is not allowed, i.e. the integer has to be explicitly converted to a float first via `.to_float()`. This prevents any ambiguity of implicit type conversion.

2.4.2 Lambdas (anonymous functions)

Basic syntax

```
fn(param1: type1, param2: type2) -> expression
```

`expression` can be a simple expression or a block of code surrounded by `{ }`, whereas the last expression of that block will be the return value.

Note that lambdas are synchronous by default. Currently there is no support for asynchronous lambdas.

Examples

```
const add_three_lambda = fn(a : int, b : float, c : float) -> a.to_float + b + c
```

the `add_three` function written as a lambda expression.

2.4.3 Invoking

A function can be invoked in the regular fashon of most programming languages (expect the lisp-like crowd):

```
function_name(value1, value2)
```

One important caveat is that asynchronous functions can NOT be called within a synchronous function. The other way round is ok.

2.5 Loops

2.5.1 While

Loops of the "while condition is true" kind can be written in the regular way:

```
while ( condition ) {
    ... expressions
}
```

Example:

```
let a : int = 0
let b : int = 0

while(a < 10) {
    a = a + 1
    b = b + a
}
```

2.5.2 For

TO2 supports "for in" loops:

```
for(variable in collection) {
    ... expressions
}
```

If a simple "from 0 to n" loop is required one can use a *range* can be use as "collection":

```
let sum = 0

for(i in 0..10) {
    sum += i + 1
}
```

If the collection contains *tuples* or *records* a deconstruction can be used as well:

```
use { CONSOLE } from ksp::console

let arr = [(f: 1.0, s: "First"), (f: 2.2, s: "Second")]
let sum = 0.0

for((s, f) in arr) {
    CONSOLE.print_line(s)
    sum += f
}
```

2.6 Modules

A module is just a TO2-file containing some public (`pub`) functions or types.

Currently all scripts are located in the plugin folder itself: `BepInEx/plugins/KontrolSystem2/to2`

The name of the script becomes the name of the module. Subfolders will be prefixed with a `::` delimiter.

E.g.

```
to2/
| - intercept.to2                module name: "intercept"
| - std/
|   | - utils.to2                module name: "std::utils"
|   | - vac.to2                  module name: "std::vac"
|   | - numerics/
|   |   | - brent_optimize.to2   module name: "std::numerics::brent_optimize
```

2.6.1 Importing for modules

Public functions can be used fully qualified: E.g.:

```
std::utils::angle_to_360(520)
```

will call the `angle_to_360` function defined in the `std::utils` module.

Alternatively function can be import via the `use` keyword.

```
use { angle_to_360 } from std::utils

angle_to_360(520)
```

2.7 Types

2.7.1 Builtin

- `bool`
  - A boolean value
  - booleans have the following helper methods:
    - `.to_string()` converts the boolean to a `string`
- `int`
  - An integer value
  - Internally a 64-bit value is used, i.e. in C# this would be a `long`
  - integers have the following helper methods:
    - `.to_float` converts the integer to a `float`
    - `.to_string()` converts the integer to a `string`
- `float`
  - A floating point value
  - Internally a 64-bit value is used, i.e in C# this would be a `double`
  - floats have the following helper methods:
    - `.to_int` converts the float to an `int` (be truncating it)
    - `.to_string()` converts the float to a `string` (with all the decimals)
    - `.to_fixed(decimals)` converts the float to a `string` with number of decimals limited to `decimals`
- `string`
  - A string value
  - strings have the following helper methods:
    - `.length` will give the length of the string
    - `.contains("substring")` checks if the string contains "substring"
    - `.starts_with("prefix")` checks if the string starts with "prefix"

    * `.ends_with("suffix")` checks if the string ends with "suffix"
    * `.pad_left(number)` will left pad the string with spaces (does nothing if `number < length`)
    * `.pad_right(number` will right pad the string with spaces (does nothing if `number < length`)
    * `.to_lower()` will convert the string to lowercase
    * `.to_upper()` will convert the string to uppercase
- `Unit`
  - Something that is not relevant for processing.
  - Usually used to mark a function with no result, i.e. in C# this would be something like a `void`

2.7.2 Arrays

An array type is defined by just adding add `[]`:

```
element_type[]
```

E.g.

```
string[]
```

is an array of strings.

```
int[][]
```

is an array of arrays of integers.

More details can be found *here*.

2.7.3 Tuples

A tuple is an ordered list of types.

```
(first_type, second_type)

// or

(first_type, second_type, third_type, fourth_type)
```

E.g.

```
(int, string)
```

is a pair (2-tuple) of an integer and a string

```
(bool, float, string)
```

is a triplet (3-tuple) of a boolean, a float and a string

These are handy if a function has multiple result values.

E.g.

```
use { floor } from core::math

fn floor_remain(a : float) -> (int, float) = (floor(a).to_int, a - floor(a))
```

is a function that returns an integer and a float.

In assignments tuples can be deconstructed:

```
let (b, c) = floor_remain(12.34)
```

will define an integer variable `b` with value `12` and a float variable `c` with value `0.34`

2.7.4 Records

Records are similar to tuples just with the addition that every element gets a unique name.

```
( name1: first_type, name2: second_type, name3: third_type )
```

E.g.

```
( ra: int, rb: string, rc: float )
```

is a record with three elements, where the first is an integer named `ra`, second is a string named `rb` and third is float named `rc`.

This way the example above may provide more details about the return value:

```
use { floor } from core::math

fn floor_remain(a : float) -> (base: int, remainder: float) = (base: floor(a).to_int,␣
↪remainder: a - floor(a))
```

Like tuples records can be deconstructed as well. By default the names have to match exactly.

```
let (base, remainder) = floor_remain(12.34)
```

will define an integer variable `base` with value `12` and a float variable `remainder` with value `0.34`

If different variable names are desired this can be mapped by using an `@` operator.

```
let (b @ base, c @ remainder) = floor_remain(12.34)
```

will define an integer variable `b` with value `12` and a float variable `c` with value `0.34`

2.7.5 Type aliases

A type alias is just a convenient shorthand for a potentially more complex types.

```
type alias_name = type definition
```

e.g.

```
type MyParameters = ( ra: int, rb: string, rc: float )
```

created an alias `MyParameters` for the record type defined above.

Like functions type aliases can be made available to other scripts with the `pub` keyword:

```
pub type MyParameters = ( ra: int, rb: string, rc: float )
```

2.8 Operators

List of all supported operators ordered by precedence.

Precedence affects the order how things are evaluated. The most basic example is `a * b + c * d` in which case you want the parser to interpret it like `(a * b) + (c * d)` and not like `a * (b + (c * d))` (or some other variation).

2.8.1 Prefix: `-`, `!`

- `-` prefix (e.g. `-a`) is a negation, which can be done on `int`, `float`, `ksp::math::Vec2`, `ksp::math::Vec3`
- `!` prefix (e.g. `!condition`) is a logical not, which can be done on `bool`

2.8.2 Power of (version >= 0.4.3)

- `**` (e.g. `a ** b`) raises `a` to the power of `b`
  - `int ** int` result `int`
  - `float ** float` result `float`

2.8.3 Multiplication / division / modulo: `*`, `/`, `%`

- `*` (e.g. `a * b`) multiplies two values, which is allowed for
  - `int * int` result `int`
  - `float * float` result `float`
  - `float * ksp::math::Vec2` result `ksp::math::Vec2`
  - `ksp::math::Vec2 * float` result `ksp::math::Vec2`
  - `ksp::math::Vec2 * ksp::math::Vec2` result `float` (vector dot product)
  - `float * ksp::math::Vec3` result `ksp::math::Vec3`
  - `ksp::math::Vec3 * float` result `ksp::math::Vec3`
  - `ksp::math::Vec3 * ksp::math::Vec3` result `float` (vector dot product)
  - Note: You can not multiply `int` with `float`, one of them as to be converted via `.to_float` or `.to_int`
- `/` (e.g. `a / b`) divides first value by the second value, which is allowed for
  - `int / int` result `int` (Note: result will be truncated `1 / 2` is `0`)
  - `float / float` result `float`
  - `ksp::math::Vec2 / float` result `ksp::math::Vec2`
  - `ksp::math::Vec3 / float` result `ksp::math::Vec3`

  - Note: You can not device `int` with `float` or vice versa, one of them as to be converted via `.to_float` or `.to_int`

- `%` (e.g. `a % b`) gets the remainder of the division of the first value by the second value, which can on, which is allowed for
  - `int % int` result `int`
  - `float % float` result `float`

2.8.4 Addition / subtraction: +, -

- `+` (e.g. `a + b`) adds two values, which is allowed for
  - `int + int` result `int`
  - `float + float` result `float`
  - `string + string` result `string` (concatenate two strings)
  - `ksp::math::Vec2 + ksp::math::Vec2` result `ksp::math::Vec2`
  - `ksp::math::Vec3 + ksp::math::Vec3` result `ksp::math::Vec3`
  - Note: You can not add `int` with `float`, one of them as to be converted via `.to_float` or `.to_int`
- `-` (e.g. `a - b`) subtracts the second value from the first, which is allowed for
  - `int - int` result `int`
  - `float - float` result `float`
  - `ksp::math::Vec2 - ksp::math::Vec2` result `ksp::math::Vec2`
  - `ksp::math::Vec3 - ksp::math::Vec3` result `ksp::math::Vec3`
  - Note: You can not subtract `int` with `float`, one of them as to be converted via `.to_float` or `.to_int`

2.8.5 Bit-And / Bit-Or / Bit-Xor: &, |, ^

- `int` values can be combined with bit operations
  - `int | int` result `int`, bit-wise or
  - `int & int` result `int`, bit-wise and
  - `int ^ int` result `int`, bit-wise xor
- An optional value can be combined with `|` to provide a default value, see *Option* for details
- Two records can be merged with a `&`, see *Record*

2.8.6 Create range: `..`, `...`

Ranges can be created from two `int` value.

- `a .. b` creates a range from `a` to `b` exclusively (i.e. `>= a` and `< b`)
- `a ... b` creates a range from `a` to `b` inclusively (i.e. `>= a` and `<= b`)

2.8.7 Comparison: `==`, `!=`, `<`, `<=`, `>`, `>=`

- The result of a comparison is always a `bool`
- `==` and `!=` check the equality resp. not-equality of two values. The following types can be compared
  - `bool` with `bool`
  - `int` with `int`
  - `float` with `float`
  - `string` with `string`
  - `ksp::math::Vec2` with `ksp::math::Vec2`
  - `ksp::math::Vec3` with `ksp::math::Vec3`
- `<`, `<=`, `>`, `>=` compare the order of two values, i.e. less than, less of equal, greater than, greater or equal. The following types can be compared
  - `int` with `int`
  - `float` with `float`
  - `string` with `string` (standard string ordering)

2.8.8 Boolean and / Boolean or: `&&`, `||`

- `&&` combines two `bool` with an and
- `||` combines two `bool` with an or

Note that this operator short-circuits if the result is already determined by the first boolean. E.g. if you have a boolean value and a function returning a boolean:

```
fn some_func() -> bool = { ... }

let a : bool
```

then `a && some_func()` will not invoke `some_func` if `a` is `false`. Correspondingly `a || some_func()` will not invoke `some_func` if `a` is `true.

2.9 Special types

2.9.1 Arrays

An array type is defined by just adding adding a `[]` to an existing type:

```
element_type[]
```

E.g.

```
string[]
```

is an array of strings.

```
int[][]
```

is an array of arrays of integers.

An array can be initialized with a `[ element_list ]` notation:

```
let a : string[] = ["zero", "one", "two", "three"]
```

initializes the variable `a` with an array of string with the elements “zero”, “one”, “two”, “three”.

Elements of an array can be accessed with the usual index notation:

```
a[2]
```

will be the string “two”.

Properties

All arrays have a `length` property giving the the number of elements.

E.g.

```
["zero", "one", "two", "three"].length
```

will be 4.

Methods

reverse

The `.reverse()` method just reverts the array.

E.g.

```
["zero", "one", "two", "three"].revert()
```

will be `["three", "two", "one", "zero"]`.

map

The `map(converter)` function can be used to convert an array to another array, whereas `converter` is a generic function that converts each element into something else:

```
[0, 1, 2, 3].map(fn(i) -> i.to_string())
```

will be the array `["0", "1", "2", "3"]`.

```
["zero", "one", "two", "three"].map(fn(s) -> "the " + s)
```

will be `["the zero", "the one", "the two", "the three"]`

map_with_index

`map_with_index` is similar to `map`, with the extension that `convert` also gets the current index.

```
["zero", "one", "two", "three"].map(fn(s, idx) -> s + " = " + idx.to_string())
```

will be `["zero = 0", "one = 1", "two = 2", "three = 3"]`

find

`find(condition)` finds the fist element in the array satisfying a condition, whereas `condition` is a function taking an element and returning a bool. The result of `find` is an *Option*.

```
["zero", "one", "two", "three"].find(fn(s) -> s == "one")
```

will be `Some("one")` (i.e. an `Option<string>` which is defined and has value "one")

```
["zero", "one", "two", "three"].find(fn(s) -> s == "something")
```

will be `None`.

exists

`exists(condition)` just checks of one of the elements satisfies the given condition. It is a shorthand for `find(condition).defined`

filter

`filter(condition)` creates a new array containing all the elements satisfying a given condition.

```
["zero", "one", "two", "three"].filter(fn(s) -> s != "one")
```

will be `["zero", "two", "three"]`.

2.9.2 Cell

Variables in a function are local by nature. But sometimes it is necessary (or just convenient) to have a value that is shared across multiple function or threads.

For this TO2 has the `Cell` wrapper type.

E.g.

```
Cell<int>              // a (memory) cell containing an integer
Cell<string>          // a (memory) cell containing a string
Cell<(int, string)>  // a (memory) cell containing a tuple of integer and string
```

Creation

A cell can be created by using the builtin `Cell` functions (not to be confused the the type itself)

```
let cell_a = Cell(1234)
let cell_b = Cell("Hallo")
```

in which case

- `cell_a` is a cell containing the integer value `1234`
- `cell_b` is a cell containing the string “Hallo”

Accessing the value

The current value of a cell can be access via the `value` property:

```
cell_a.value     // will be the interger 1234
cell_b.value     // will be the string "Hallo"
```

The other way round the value of the cell can be overwritten by using the same property

```
cell_a.value = 2345    // now cell_a will have the value 2345
cell_b.value = "World" // now cell_b will have the value "World"
```

Since cells call potentially be modified in multiple thread, calculating with cell values can be tricky.

E.g.

```
cell_a.value = cell_a.value + 1
```

has the ever so slight chance that `cell_a` is modified by a different thread just between `cell_a.value` has been read but the addition has not yet occured, which might lead to unpredictable behaviour.

To mitigate this the `Cell` type has an `update` method taking a function that converts the current value to the new value.

I.e.

```
cell.update( fn(current) -> current + 1 )
```

is the correct way to implement a thread-safe counter.

2.9.3 Option

Some times values are optional, e.g. an expression like `array_of_string.find(fn(s) -> s == "hallo")` may or may not have a result.

To handle these there is an `Option<type>` wrapper.

E.g.

```
Option<int>            // an optional integer
Option<string>         // an optional string
Option<(int, string)>  // an optional tuple of integer and string
```

Creation

The simplest way to create/instantiate an optional value is to use the builtin `Some` and `None` functions:

```
let maybe_a : Option<int> = Some(2345)
let maybe_b : Option<int> = None()
```

in which case

- `maybe_a` is an optional integer that has the value `2345` (i.e. is defined)
- `maybe_b` is an optional integer that has no value (i.e. is not defined)

Alternative an optional value can be create as the result of an *if-expression*

```
let maybe_a = if (b < 0) 2345
```

in which case

- if `b < 0` is true `maybe_a` will be an optional integer with value `2345`
- if `b < 0` is not true `maybe_a` will be an optional integer with no value

Accessing the value

`Option` has the following properties to access the value directly:

- `optional_value.defined` a bool indicating if there is a value or not
- `optional_value.value` the actual value, if there is one
  - Be careful using this! Accessing the value of an option with `defined == false` will most likely lead to unexpected behaviour. At the very least wrap this in an if condition like

```
if (optional_value.defined) {
  optional.value  // use it in some way
}
```

A better way to access an optional value is to use the `|` operator:

```
let maybe_a : Option<int> = Some(2345)
let maybe_b : Option<int> = None()

maybe_a | 1234    // will be integer 2345 since maybe_a is defined
maybe_b | 1234    // will be integer 1234 since maybe_b is not defined
```

2.9.4 Range

In many cases is convenient to have an interval of integers. TO2 has two convenient shorthands for this:

- `min..max` a range of integers from `min` to `max` exclusive
  - E.g. `3..6` would be the numbers `[3, 4, 5]`
- `min...max` a range of integers from `min` to `max` inclusive
  - E.g. `3...6` would be the numbers `[3, 4, 5, 6]`

Most commonly this is used in *for loops*:

```
let k = 0

for(i in 0..10) {
    k = k + 1
}
```

Properties

A range has a `length` property. Therefor

```
(3..6).length
```

will be `3`

Methods

A range has a `map(converter)` function that can be used to convert to into an array, whereas `converter` is a generic function that converts an `int` into something else:

```
(0..4).map(fn(i) -> i * 2)
```

will be the array `[0, 2, 4, 6]`.

```
(0..4).map(fn(i) -> i.to_string())
```

will be the array `["0", "1", "2", "3"]`.

2.9.5 Result

Things will never go as planed and loosely speaking one has to deal with two types of situations: The things you can deal with and the thing you can not. (Mindblowing isn't it?)

The latter are usually stuff that is pretty much out of your control entirely. In programmers terms this usually leads to some sort of system exception that will crash the process (maybe by proxy since the entire system crashes).

The former are more interesting as these kind of situations could be categories into:

- Things one is not aware of and are therefore not dealt with
- Things one is aware of, but are to hard to be dealt with properly
- Things that actually are dealt with properly

Obviously the last category should be maximized. An essential part for this to raise awareness and make it slightly inconvenient to just ignore potential error cases.

For this TO2 has `Result` wrapper type that encapsulates a successful value and an error value.

E.g.

```
Result<int, string>           // if successful an integer, on error a string (containing␣
↪some sort of message)
Result<string, string>        // if successful a string value, on error also a string␣
↪but containing some sort of message
Result<(int, string), string> // tuple of integer and string ...
```

Note: In almost all cases a `string` is used in the error-case, potentially this should be replaced with something more sophisticated.

This allows function to return a value that might not be available for some specific reason. E.g. *`vessel.maneuver.next_node()`* might not have a result because there just is no maneuvering node for that vessel, or it is in the past, or some other reason

Creation

The simplest way to create/instantiate an optional value is to use the builtin `Ok` and `Err` functions:

```
let result_a : Result<int, string> = Ok(2345)
let result_b : Result<int, string> = Err("just not there")
```

in which case

- `result_a` is an integer result that has the value `2345` (i.e. successful)
- `result_a` is an integer that has the error message "just not there" (i.e. not successful)

Alternatively an *`Option`* can converted to the result via the `ok_or` method by providing an error message for the `None` case:

```
let maybe_a : Option<int> = Some(2345)
let maybe_b : Option<int> = None()

let result_a = maybe_a.ok_ok("just not there")
let result_b = maybe_b.ok_ok("just not there")
```

creates the same as above.

Accessing the value

`Result` has the following properties to access the value directly:

- `result_value.success` a bool indicating if the result was successful
- `result_value.value` the actual value, if the result was successful
- `result_value.error` the error value, if the result was not successful
  - Be careful using either of these! At the every least wrap it in an if condition

```
if (result_value.success) {
  result_value.value   // use it
} else {
  result_value.error   // maybe log the error
}
```

A more convenient way is to use the ? operator.

```
let a = result_value?
```

is a shorthand for

```
if ( !result_value.success ) {
   return Err(result.error)
}
let a = result_value.value
```

Obviously this only works if the surrounding function also returns a result of some kind, otherwise the ? operator will produce a compilation error

2.10 Structs

Sometime it is necessary to have a complex state that is shared between multiple functions and using a *Cell* is just to cumbersome. In that case it is possible to define a `struct` (in object-oriented terms this would be called a class).

```
struct NameOfStruct(param1: paramType1, param2: paramType2) {
    field1 : fieldType1 = initial_value
    field2 : fieldType1 = initial_value
}
```

will implicitly create a `NameOfStruct(param1: paramType1, param2: paramType2)` function creating a new instance. To make this available to other modules this can be prefixed with `pub`.

Example:

```
struct Counter(initial_name: string) {
    name  : string = initial_name
    count : int    = 0
}
```

then a new `Counter` struct can be created with

```
let my_counter = Counter("My counter")

CONSOLE.print_line("Name : " + my_counter.name)              // Will be "Name : My␣
↪counter"
CONSOLE.print_line("Count: " + my_counter.count.to_string)   // Will be "Count: 0"
```

It is possible to define methods to operate on that struct:

```
impl NameOfStruct {
  fn method_name(self, param1: type1, param2: type1) -> return_type = {
     ... implementation where "self" is a reference to the struct itself
```

(continues on next page)

(continued from previous page)

```
  }
}
```

In the example above

```
impl Counter {
    fn to_string(self) -> string = {
        "Counter(" + self.name + ", " + self.count.to_string() + ")"
    }

    fn inc(self) -> Unit = {
        self.count = self.count + 1
    }
}
```

then the following can be done:

```
let my_counter = Counter("My counter")

CONSOLE.print_line(my_counter.to_string())   // Will be "Counter(My counter, 0)"

my_counter.inc()

CONSOLE.print_line(my_counter.to_string())   // Will be "Counter(My counter, 1)"
```

2.11 Examples

2.11.1 Hello world

A wee bit more than just that.

Code

```
use { Vessel } from ksp::vessel
use { CONSOLE } from ksp::console

pub fn main_flight(vessel: Vessel) -> Unit = {
    CONSOLE.clear()
    CONSOLE.print_line("Hello world: " + vessel.name)
}
```

Annotated

```
use { Vessel } from ksp::vessel
```

- Import the *Vessel* type from the builtin `ksp::vessel` module.
- The `Vessel` type contains many properties and methods to interact with a vessel in the game
- Please to not interact with Kerbals this way
  - … I mean … you potentially could, but it is probably not very nice

```
use { CONSOLE } from ksp::console
```

- Import the *CONSOLE* type from the builtin `ksp::console` module.

```
pub fn main_flight(vessel: Vessel) -> Unit = {
```

- The entrypoint for flight-mode, i.e. the script will appear on the main window if the game is in flight-mode
- `vessel` will be a reference to the currently active vessel

```
    CONSOLE.clear()
```

- Clear the console

```
    CONSOLE.print_line("Hello world:" + vessel.name)
```

- Will print “Hello world:” followed by the name of the active vessel

CHAPTER

THREE

INTERACT WITH THE GAME

3.1 Asynchronous execution

By default all scripts are running asynchronously in the game, so in many cases triggering an action will not produce an immediate change.

Instead you should use one of the following functions in *ksp::game* to give the game a little time to catch up.

3.1.1 Yield

The `yield()` function is the most basic form ot "interruption", it will pause the script for the shortest amount possible and wait for the next update loop of the game.

Essentially it is a `sleep(0)` (see below).

3.1.2 Sleep

The `sleep(seconds)` function will pause the script for a given amount of seconds.

Due to the nature of the game loop this is just a rough estimate. If you need a precise amount of seconds passed for some calculation you should recheck the `current_time()`.

3.1.3 Wait while/until

`wait_while(condition)` pauses the script while a given condition is true and correspondingly `wait_until(condition)` pauses while the condition is false (until it becomes true).

In both cases `condition` is a function that returns a boolean (`fn() -> bool`). So a `sleep(12.34)` could be also written as:

```
const ut = current_time()

wait_while(fn() -> current_time() - ut < 12.34)
```

or

```
const ut = current_time()

wait_until(fn() -> current_time() - ut >= 12.34)
```

Note that the `condition` is checked on every game loop, so it should be as simple as possible. Doing a complex calculation here can have a serious impact on the game.

3.2 Controlling the vessel

3.2.1 SAS

The simplest way to control the orientation of the vessel is by using the SAS system.

SAS can be enabled/disable via

```
vessel.autopilot.enabled = true/false
```

or

```
vessel.actions.sas = true/false
```

Note: The SAS action group is somewhat wonky, so under the hood both ways do exactly the same thing.

The SAS system can be then set to different modes:

```
vessel.autopilot.mode = AutopilotMode.$(Mode)
```

with $(Mode) as

- `StabilityAssist`: Default mode that does not any steering, just kills rotation
- `Prograde`, `Retrograde`, `Normal`, `Antinormal`, `RadialIn`, `RadialOut`: Steer to orbital directions
- `Target`: Steer towards target (if vessel has a target)
- `AntiTarget`: Steer away from taret (if vessel has a target)
- `Maneuver`: Steer in direction of next maneuvering node (if there is one)
- `Navigation`, `MODE_AUTOPILOT`: Not sure what the difference is between those two, but you can provide a target orientation.

So

```
vessel.autopilot.enabled = true
vessel.autopilot.mode = AutopilotMode.Autopilot
vessel.autopilot.target_orientation = direction
```

tells the SAS to steer the vessel into `direction`.

3.2.2 Use input override methods

If direct control is required is is possible to just override the controls of a vessel by using the following methods:

- *override_input_pitch*
- *override_input_yaw*
- *override_input_roll*
- *override_input_translate_x*
- *override_input_translate_y*

- *override_input_translate_z*

3.2.3 Control manager

All control managers in the *ksp::control* module follow the same principle:

```
let manager = vessel.manage_the_thing_to_control(value_provider)  // Script takes over␣
↪the control (overriding the pilot)

// ... wait/sleep for whatever condition

manager.release()  // Script releases control, i.e. give control back to the pilot


// Optionally wait from something else

manager.resume()  // Take back control again
```

The `value_provider` is a function that will call on very game update, it gets a `deltaT: float` of the time since the last update and return the steering value.

ThrottleManager

```
let manager = vessel.manage_throttle(fn(deltaT) -> throttle_value)
```

manages the throttle of a rocket or airplane. The value is between `0.0` (no throttle) and `1.0` (full throttle).

```
let manager = vessel.set_throttle(throttle_value)
```

is just a shorthand to set the throttle to a constant value. This can be changed at any time via `manager.throttle = new_value`.

SteeringManager

```
let manager = vessel.manage_steering(fn(deltaT) -> vec3(pitch_value, yaw_value, roll_
↪value))
```

manages the pitch, yaw and roll of the flight stick. All values are between `-1.0` and `1.0` (full rudder in either direction)

```
let manager = vessel.set_sterring(vec3(pitch_value, yaw_value, roll_value))
```

is just a shorthand to set pitch, yaw, roll to a constant value. This can be changed at any time via `manager.pitch_yaw_roll = new_value`.

RCSTranslateManager

```
let manager = vessel.manage_rcs_translate(fn(deltaT) -> vec3(x, y, z))
```

manages the RCS controls. All values are between `-1.0` and `1.0` (full thrust in either direction)

```
let manager = vessel.set_rcs_translate(vec3(x, y, z))
```

is just a shorthand RCS thrust to a constant value. This can be changed at any time via `manager.translate = new_value`.

Note: This will do nothing is RCS is disabled.

RCS can be enabled/disabled via

```
vessel.actions.rcs = true/false
```

WheelSteeringManager / WheelThrottleManager

... the corresponding control managers for rovers. Completely untested so far.

3.3 Coordinate system independent vectors

The following new types have been introduces to `ksp::math`

- `GlobalPosition` a position in space
- `GlobalVector` a vector in space (i.e. a direction with a magnitude)
- `GlobalDirection` a direction/orientation (like the orientation of the vessel)
- `GlobalVelocity` a velocity
- `GlobalAngularVelocity` an angular velocity

All of these are coordinate system independent, i.e. to get the actual `x`, `y`, `z` values one has to provide a frame of reference/coordinate system. The `Body`, `Vessel` and `Orbit` types now have corresponding `global_...` methods to obtain these value.

The following frames of reference are available:

- `body.celestial_frame` the celestial (non-rotating) frame of a planet or moon
- `body.body_frame` the rotating frame of a planet or moon
- `vessel.celestial_frame` the celestial (non-rotating) frame of a vessel
- `vessel.body_frame` the rotating frame of a vessel
- `vessel.control_frame` the rotating frame of a vessel from the current control point (e.g this changes when controlling a vessel from a docking port)

As this sound pretty abstract here are some examples:

```
const p : GlobalPosition = vessel.global_position

const p_body : Vec3 = p.to_local(vessel.main_body.celestial_frame)

const p_vessel : Vec3 = p.to_local(vessel.celestial_frame)
```

in this case

- `p` represents the current position of the vessel
- `p_body` is the coordinates of the vessel in the coordinate system of the main body in the current SOI
- `p_vessel` is the coordinates of the vessel in the coordinate system of the vessel itself, which happens to be `[0, 0, 0]`

To print a `GlobalPosition` to the console, it is also necessary to provide a coordinate system

```
const frame = vessel.main_body.celestial_frame  // we will do our calculations in this␣
↪frame of reference

CONSOLE.print_line(p.to_string(frame))
CONSOLE.print_line(p.to_fixed(frame, 3))
```

A `GlobalVector` is very similar to a `GlobalPosition`, just that it represents something that goes from A to B, which is represented in the calculations:

```
// Just to positions of something
const p1 : GlobalPosition = ...
const p2 : GlobalPosition = ...

p2 - p1     // this a GlobalVector from p1 to p2
p1 - p2     // this a GlobalVector from p2 to p1
p1 + p2     // ERROR: This is not allowed

// A vector
const v1 : GlobalVector = ...

p1 + v1         // New position when moving from p1 in direction v1
p2 + 2 * v2     // New position when moving from p2 twice the distrance in direction v1
```

Things become more tricky when dealing with velocities, because it now depends if the frame of reference itself is in motion or not. In general: The `celestial_frame` are considered to be non-rotation (fixed to celestial stars) while the `body_frame` is taking the rotation of the current body into account.

```
const v = vessel.global_velocity

const v_celestial = v.to_local(vessel.main_body.celestial_frame)

const v_body = v.to_local(vessel.main_body.body_frame)
```

in this case

- `v` represents the current velocity of the vessel
- `v_celestial` is the velocity vector in the celestial frame of the main body, this is also known as the `orbital_velocity`
- `v_body` is the velocity vector in the rotating frame of the main body, this is also known as the `surface_velocity`

3.3.1 Supported operations

GlobalPosition

- Two `GlobalPosition` can be subtracted from each other resulting in a `GlobalVector`

```
let p1 : GlobalPosition = ...
let p2 : GlobalPosition = ...
let v : GlobalVector = p2 - p1 // Global vector pointing from p1 to p2
```

- A `GlobalVector` can be added to a `GlobalPosition` resulting in a new `GlobalPosition`

```
// p1, p2, v from above
let p3 : GlobalPosition = p1 + v // This will be the same as p2
```

- `GlobalPosition` has a `distance` helper to calculate the distance between two points in space:

```
let p1 : GlobalPosition = ...
let p2 : GlobalPosition = ...
let d : float = p1.distance(p2)
let d2 : float = p1.distance_sqr(p2) // Squared distance, i.e. d2 = d * d
```

- `GlobalPosition` has a `lerp_to` helper to get midpoints between two positions

```
let p1 : GlobalPosition = ...
let p2 : GlobalPosition = ...
let p3 : GlobalPosition = p1.lerp_to(p2, 0.5)  // Midpoint between p1, p2
let v : GlobalVector = p2 - p1
let p4 : GlobalPosition = p1 + 0.5 * v         // Same as p3
```

GlobalVector

`GlobalVector` can be used almost the same as a regular vector.

- A `GlobalVector` can be multiplied with a `float` resulting in a new `GlobalVector`

```
let v : GlobalVector = ...
let f : float = ...
let v1 : GlobalVector = f * v
let v2 : GlobalVector = v * f
```

- Two `GlobalVector` can be added to and subtracted from each other resulting in a new `GlobalVector`

```
let v1 : GlobalVector = ...
let v2 : GlobalVector = ...
let v3 : GlobalVector = v1 + v2
let v4 : GlobalVector = v1 + v2
```

- `GlobalVector` supports the dot-product:

```
let v1 : GlobalVector = ...
let v2 : GlobalVector = ...
let f1 : float = v1 * v2
let f2 : float = v1.dot(v2)  // Same as above
```

- `GlobalVector` supports the cross-product:

```
let v1 : GlobalVector = ...
let v2 : GlobalVector = ...
let v3 : GlobalVector = v1.cross(v2)
```

3.4 Debug vectors and ground markers

3.4.1 Debug vectors

Debug vectors are back, i.e. point lines you can draw in the game world to figure out what is wrong with the autopilot you are working on.

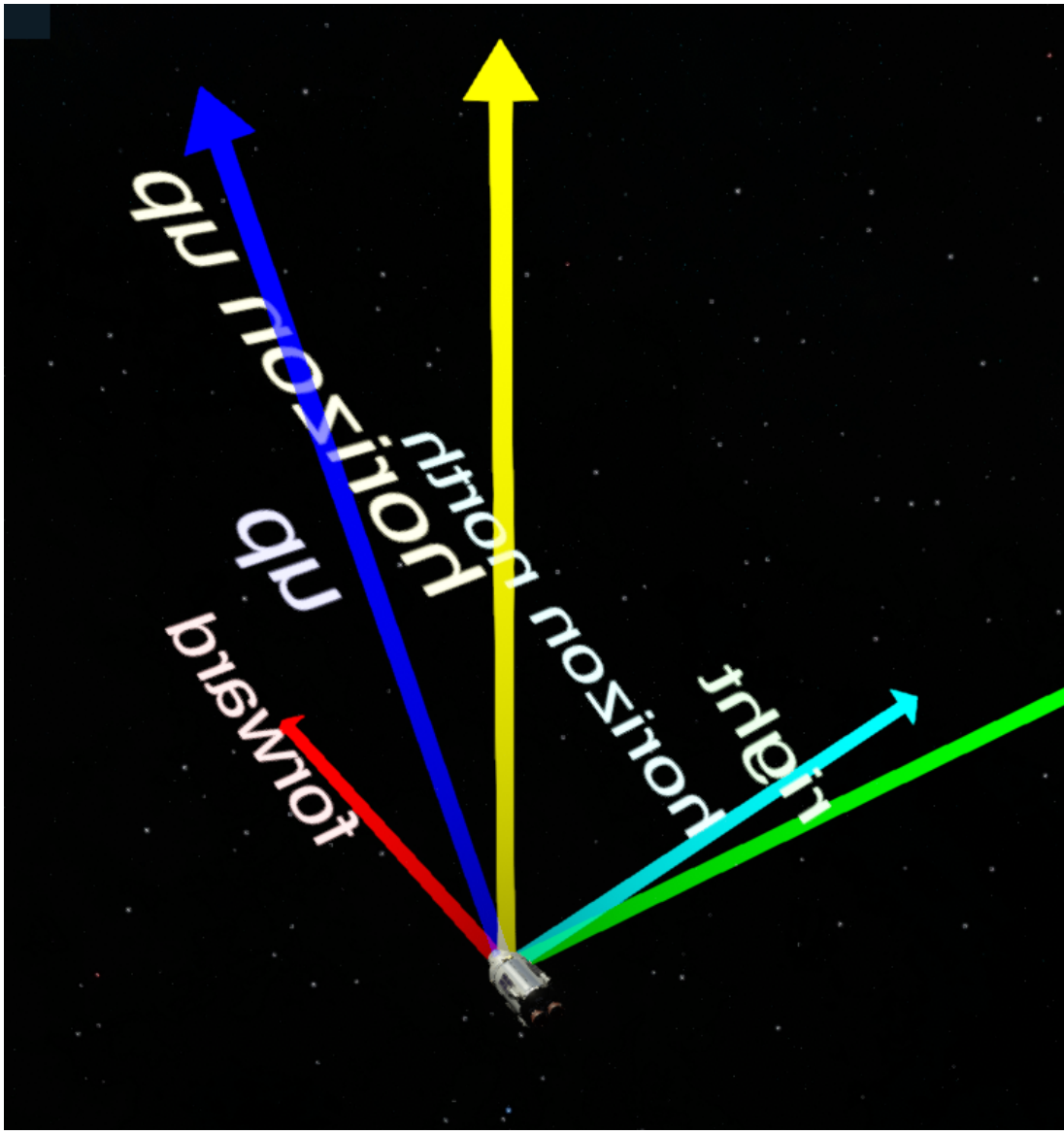
Here is a little example script:

```
use { Vessel } from ksp::vessel
use { CONSOLE, RED, BLUE, YELLOW, GREEN, CYAN } from ksp::console
use { DEBUG } from ksp::debug
use { sleep } from ksp::game

pub fn main_flight(vessel: Vessel) -> Result<Unit, string> = {
    let forward = DEBUG.add_vector(fn() -> vessel.global_position, fn() -> vessel.global_
↪facing.vector * 50, RED, "forward", 1)
    let up = DEBUG.add_vector(fn() -> vessel.global_position, fn() -> vessel.global_
↪facing.up_vector * 50, BLUE, "up", 1)
    let right = DEBUG.add_vector(fn() -> vessel.global_position, fn() -> vessel.global_
↪facing.right_vector * 50, GREEN, "right", 1)
    let horizon_up = DEBUG.add_vector(fn() -> vessel.global_position, fn() -> vessel.
↪global_up * 50, YELLOW, "horizon up", 1)
    let horizon_north = DEBUG.add_vector(fn() -> vessel.global_position, fn() -> vessel.
↪global_north * 50, CYAN, "horizon north", 1)

    sleep(30)
}
```

will produce something like this:



If you make them long enough (e.g. multiply with `5000` instead of `50`) they will be visible on the map view as well.

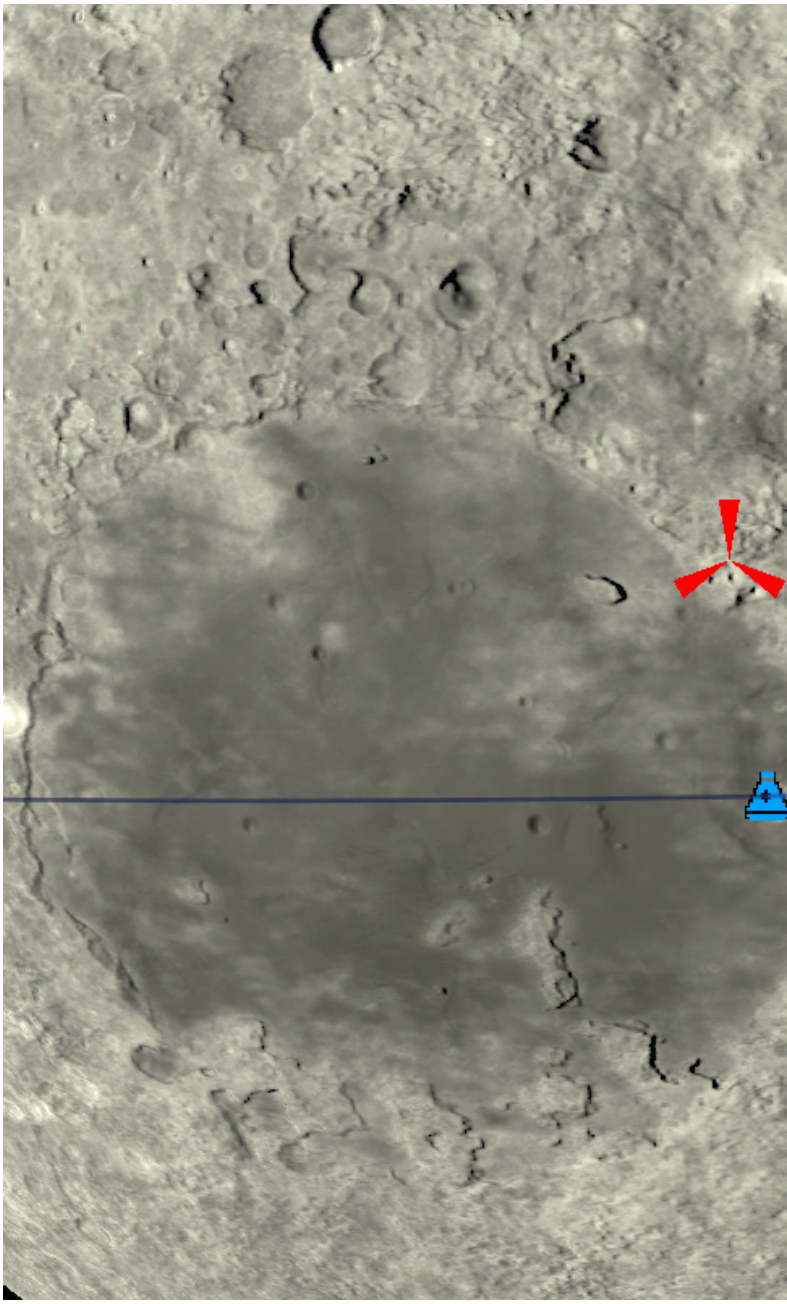
3.4.2 Ground markers

Ground markers are nice to visualize a `GeoPosition`:

```
use { Vessel } from ksp::vessel
use { CONSOLE, RED, BLUE, YELLOW, GREEN, CYAN } from ksp::console
use { DEBUG } from ksp::debug
use { sleep } from ksp::game

pub fn main_flight(vessel: Vessel) -> Result<Unit, string> = {
    let ground = DEBUG.add_ground_marker(vessel.geo_coordinates, RED, 0)

    sleep(30)
}
```



will mark the point on the ground where the vessel currently is:
This is visible in the map view and also in the flight view if you are close to the ground

3.5 PIDLoop

3.5.1 Example

Here is a simple example how to create and use a *PIDLoop*:

```
use { Vessel } from ksp::vessel
use { CONSOLE } from ksp::console
use { sleep, current_time } from ksp::game
```

(continues on next page)

(continued from previous page)

```
use { pid_loop } from ksp::control

pub fn main_flight( vessel: Vessel) -> Result<Unit, string> = {
    let tKp = 0.1                //throttle hold PID control variables
    let tKi = 0.0002
    let tKd = 0.01
    let tmin = 0.0
    let tmax = 1.0
    let throttlePID = pid_loop(tKp, tKi, tKd, tmin, tmax)

    let fake_speed = 0.0

    throttlePID.setpoint = 100   // desired speed

    CONSOLE.clear()
    for(i in 0..1000) {
        sleep(0.2)
        const throttle = throttlePID.update(current_time(), fake_speed)

        fake_speed += throttle * 10    // super-simple acceleration
        fake_speed -= 2                // super-simple drag

        CONSOLE.print_line("S: " + fake_speed.to_fixed(3) + " T: " + throttle.to_
↪fixed(3))
    }
}
```

3.6 Collecting telemetry data

The *ksp::telemetry* module offers the ability to collect arbitrary telemetry data in form of *TimeSeries*.

A `TimeSeries` can be used to record the changes of a values over a span of time. To limit the amount of memory a time series as a minimum time resolution, i.e. the interval of `start_ut` to `end_ut` is split into a limited amount of buckets (at the moment the number of buckets can not exceed 5000). Additionally overall number of time series that can be held in memory is limited to 20.

3.6.1 Simple example

The following example demonstrates how a time series can be created in a script:

```
use { Vessel } from ksp::vessel
use { CONSOLE } from ksp::console
use { add_time_series } from ksp::telemetry
use { cos_deg } from core::math

pub fn main_flight( vessel: Vessel) -> Result<Unit, string> = {
    let cos = add_time_series("Cos", 0, 0.1)
    for(i in 0..20000) {
        cos.add_data(i * 0.2, cos_deg(i / 20.0))
```

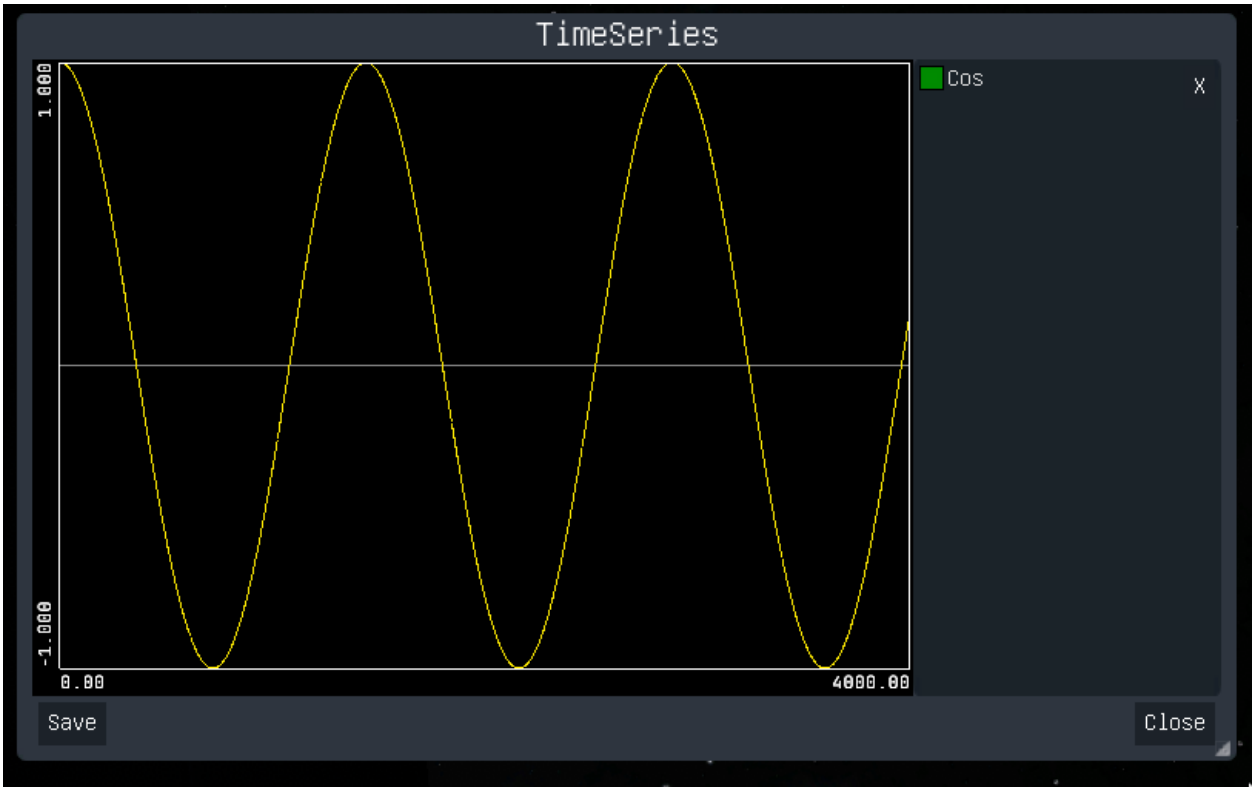
(continues on next page)

(continued from previous page)

```
    }
}
```

- With the `add_time_series` function a new time series "Cos" is created with a start time of `0` and an initial time resolution of `0.1`.
- Then values are added using the `add_data` method on the time series.

Which will then crate something like this.



3.6.2 More useful example

The following example is a script records the altitude, mass, pitch, horizontal surface speed and orbit apoapsis of the current vessel until it is terminated:

```
use { Vessel } from ksp::vessel
use { CONSOLE } from ksp::console
use { current_time, sleep } from ksp::game
use { add_time_series } from ksp::telemetry

pub fn main_flight( vessel: Vessel) -> Result<Unit, string> = {
    let ut = current_time()
    let altitude = add_time_series("Altitude", ut, 0.1)
    let mass = add_time_series("Mass", ut, 0.1)
    let pitch = add_time_series("Pitch", ut, 0.1)
    let surface_velocity = add_time_series("Surface velcoty", ut, 0.1)
    let horizonal_surface_speed = add_time_series("Horizonal Srf Speed", ut, 0.1)
```

(continues on next page)

(continued from previous page)

```
    let apoapsis = add_time_series("Apoapsis", ut, 0.1)

    while(true) {
        ut = current_time()
        altitude.add_data(ut, vessel.altitude_sealevel)
        mass.add_data(ut, vessel.mass)
        pitch.add_data(ut, vessel.pitch_yaw_roll.x)
        surface_velocity.add_data(ut, vessel.surface_velocity.magnitude)
        horizonal_surface_speed.add_data(ut, vessel.horizontal_surface_speed)
        apoapsis.add_data(ut, vessel.orbit.apoapsis | 0)

        sleep(0.2)
    }
}
```

As it does not interfered with the steering at all, it can run in parallel to any start or landing script.

3.6.3 Storing the data

The data of all time series can be saved to a json file either via the UI or by using the *save_time_series* function.

The content of this file will look as follows:

```
[
  {
    "name": "Altitude",
    "startUT": 32089.195872716373,
    "endUT": 32403.095872716374,
    "resolution": 0.1,
    "values": [
      {
        "ut": 32089.195872716373,
        "count": 1,
        "min": 157.59264053509105,
        "avg": 157.59264053509105,
        "max": 157.59264053509105
      }, {
          "ut": 32089.395872716374,
          "count": 1,
          "min": 158.46842386422213,
          "avg": 158.46842386422213,
          "max": 158.46842386422213
      },

...
    ]
  },
  {
    "name": "Mass",
    "startUT": 32089.195872716373,
    "endUT": 32403.095872716374,
    "resolution": 0.1,
```

(continues on next page)

(continued from previous page)


```
    "values": [
      {
        "ut": 32089.195872716373,
        "count": 1,
        "min": 242.71394239419527,
        "avg": 242.71394239419527,
        "max": 242.71394239419527
      },
      {
        "ut": 32089.395872716374,
        "count": 1,
        "min": 242.36318659964232,
        "avg": 242.36318659964232,
        "max": 242.36318659964232
      },
...
```


- `name` is the name of the time series
- `startUT` is the start time (seconds)
- `endUT` is the end time (seconds)
- `resolution` is the time resolution (seconds)
- `values` is an array of all the non-empty buckets, whereas each bucket contains
  - `ut` start time of the bucket
  - `count` the number of values that have been recorded in the timeframe of the bucket (i.e. the resolution)
  - `min`, `avg`, `max` the minimum, average and maximum fo all values that have recorded been recorded in the timeframe of the bucket.

CHAPTER

FOUR

REFERENCE

4.1 core::background

Provides means to run functions as asynchronous background task.

4.1.1 Types

Task

Represents a background task

Fields

Name	Type	Read-only	Description
is_canceled	bool	R/O	Check if the task has been canceled
is_completed	bool	R/O	Check if the task is completed
is_success	bool	R/O	Check if the task is completed and has a value
result	T	R/O	Get the result of the task once completed

Methods

cancel

```
task.cancel ( ) -> Unit
```

Cancel/abort the task

wait_complete

```
task.wait_complete ( ) -> T
```

Asynchronously wait for background task to complete

4.1.2 Functions

is_background

```
pub sync fn is_background ( ) -> bool
```

Check if current thread is a background thread

run

```
pub sync fn run ( function : sync fn() -> T ) -> core::background::Task<T>
```

Run a function as background task.

Parameters

Name	Type	Optional	Description
function	sync fn() -> T		

4.2 core::error

Error reporting and error handling.

4.2.1 Types

Error

Error information of a failed Result.

Fields

Name	Type	Read-only	Description
message	string	R/O	
stack_trace	*core::error::StackEntry*[]	R/O	

Methods

to_string

```
error.to_string ( ) -> string
```

StackEntry

Stacktrace entry.

Fields

Name	Type	Read-only	Description
arguments	string[]	R/O	
line	int	R/O	
name	string	R/O	
source_name	string	R/O	

Methods

to_string

```
stackentry.to_string ( ) -> string
```

4.2.2 Functions

current_stack

```
pub sync fn current_stack ( ) -> core::error::StackEntry[]
```

Get current stack.

4.3 core::logging

Provides basic low-level logging. In KSP all log messages will appear in the debug console as well as the `KSP.log` file.

4.3.1 Functions

debug

```
pub sync fn debug ( message : string ) -> Unit
```

Write a debug-level `message`.

Parameters

Name	Type	Optional	Description
message	string		

error

```
pub sync fn error ( message : string ) -> Unit
```

Write an error-level `message`.

Parameters

Name	Type	Optional	Description
message	string		

info

```
pub sync fn info ( message : string ) -> Unit
```

Write an info-level `message`.

Parameters

Name	Type	Optional	Description
message	string		

warning

```
pub sync fn warning ( message : string ) -> Unit
```

Write a warning-level `message`.

Parameters

Name	Type	Optional	Description
message	string		

4.4 core::math

Collection of basic mathematical functions.

4.4.1 Types

Random

Random number generator

Methods

next_float

```
random.next_float ( ) -> float
```

Get next random number between 0.0 and 1.0

next_gaussian

```
random.next_gaussian ( mu : float,
                       sigma : float ) -> float
```

Get next gaussian distributed random number

Parameters

Name	Type	Optional	Description
mu	float	x	Mean value
sigma	float	x	Standard deviation

next_int

```
random.next_int ( min : int,
                  max : int ) -> int
```

Get next random number between `min` and `max`

Parameters

Name	Type	Optional	Description
min	int		Minimum value (inclusive)
max	int		Maximum value (inclusive)

4.4.2 Constants

Name	Type	Description
DEG_TO_RAD	float	Multiplicator to convert an angle of degree to radian.
E	float	Represents the natural logarithmic base, specified by the e constant,
EPSILON	float	Machine epsilon, i.e lowest possible resolution of a floating point number.
MAX_FLOAT	float	Maximum possible floating point number.
MAX_INT	int	Maximum possible integer number.
MIN_FLOAT	float	Minimum possible floating point number.
MIN_INT	int	Minimum possible integer number.
PI	float	Represents the ratio of the circumference of a circle to its diameter, specified by the constant, .
RAD_TO_DEG	float	Multiplicator to convert an angle of radian to degree.

4.4.3 Functions

abs

```
pub sync fn abs ( value : float ) -> float
```

Returns the absolute value of a number.

Parameters

Name	Type	Optional	Description
value	float		

acos

```
pub sync fn acos ( d : float ) -> float
```

Returns the angle in radian whose cosine is the specified number.

Parameters

Name	Type	Optional	Description
d	float		

acos_deg

```
pub sync fn acos_deg ( x : float ) -> float
```

Returns the angle in degree whose cosine is the specified number.

Parameters

Name	Type	Optional	Description
x	float		

acosh

```
pub sync fn acosh ( x : float ) -> float
```

Returns the angle whose hyperbolic cosine is the specified number.

Parameters

Name	Type	Optional	Description
x	float		

asin

```
pub sync fn asin ( d : float ) -> float
```

Returns the angle in radian whose sine is the specified number.

Parameters

Name	Type	Optional	Description
d	float		

asin_deg

```
pub sync fn asin_deg ( x : float ) -> float
```

Returns the angle in degree whose sine is the specified number.

Parameters

Name	Type	Optional	Description
x	float		

asinh

```
pub sync fn asinh ( x : float ) -> float
```

Returns the angle whose hyperbolic sine is the specified number.

Parameters

Name	Type	Optional	Description
x	float		

atan

```
pub sync fn atan ( d : float ) -> float
```

Returns the angle in radian whose tanget is the specified number.

Parameters

Name	Type	Optional	Description
d	float		

atan2

```
pub sync fn atan2 ( y : float,
                    x : float ) -> float
```

Returns the angle in redian whose tangent is the quotient of two specified numbers.

Parameters

Name	Type	Optional	Description
y	float		
x	float		

atan2_deg

```
pub sync fn atan2_deg ( y : float,
                        x : float ) -> float
```

Returns the angle in degree whose tangent is the quotient of two specified numbers.

Parameters

Name	Type	Optional	Description
y	float		
x	float		

atan_deg

```
pub sync fn atan_deg ( x : float ) -> float
```

Returns the angle in degree whose tangent is the specified number.

Parameters

Name	Type	Optional	Description
x	float		

atanh

```
pub sync fn atanh ( x : float ) -> float
```

Returns the angle whose hyperbolic tanget is the specified number.

Parameters

Name	Type	Optional	Description
x	float		

ceiling

```
pub sync fn ceiling ( a : float ) -> float
```

Returns the smallest integral value that is greater than or equal to the specified number.

Parameters

Name	Type	Optional	Description
a	float		

clamp

```
pub sync fn clamp ( x : float,
                    min : float,
                    max : float ) -> float
```

Clamp a number between a given minimum and maximum

Parameters

Name	Type	Optional	Description
x	float		
min	float		
max	float		

clamp_degrees180

```
pub sync fn clamp_degrees180 ( angle : float ) -> float
```

Clamp an angle between -180 and 180 degree

Parameters

Name	Type	Optional	Description
angle	float		

clamp_degrees360

```
pub sync fn clamp_degrees360 ( angle : float ) -> float
```

Clamp an angle between 0 and 360 degree

Parameters

Name	Type	Optional	Description
angle	float		

clamp_radians2_pi

```
pub sync fn clamp_radians2_pi ( angle : float ) -> float
```

Clamp an angle between 0 and 2

Parameters

Name	Type	Optional	Description
angle	float		

clamp_radians_pi

```
pub sync fn clamp_radians_pi ( angle : float ) -> float
```

Clamp an angle between - and

Parameters

Name	Type	Optional	Description
angle	float		

cos

```
pub sync fn cos ( d : float ) -> float
```

Returns the cosine of the specified angle in redian.

Parameters

Name	Type	Optional	Description
d	float		

cos_deg

```
pub sync fn cos_deg ( x : float ) -> float
```

Returns the cosine of the specified angle in degree.

Parameters

Name	Type	Optional	Description
x	float		

cosh

```
pub sync fn cosh ( value : float ) -> float
```

Returns the hyperbolic cosine of the specified angle.

Parameters

Name	Type	Optional	Description
value	float		

exp

```
pub sync fn exp ( d : float ) -> float
```

Returns e raised to the specified power.

Parameters

Name	Type	Optional	Description
d	float		

floor

```
pub sync fn floor ( d : float ) -> float
```

Returns the largest integral value less than or equal to the specified number.

Parameters

Name	Type	Optional	Description
d	float		

log

```
pub sync fn log ( d : float ) -> float
```

Returns the natural (base e) logarithm of a specified number.

Parameters

Name	Type	Optional	Description
d	float		

log10

```
pub sync fn log10 ( d : float ) -> float
```

Returns the base 10 logarithm of a specified number.

Parameters

Name	Type	Optional	Description
d	float		

max

```
pub sync fn max ( val1 : float,
                  val2 : float ) -> float
```

Returns the larger of two decimal numbers.

Parameters

Name	Type	Optional	Description
val1	float		
val2	float		

min

```
pub sync fn min ( val1 : float,
                  val2 : float ) -> float
```

Returns the smaller of two decimal numbers.

Parameters

Name	Type	Optional	Description
val1	float		
val2	float		

pow

```
pub sync fn pow ( x : float,
                  y : float ) -> float
```

Returns a specified number raised to the specified power.

Parameters

Name	Type	Optional	Description
x	float		
y	float		

random

```
pub sync fn random ( ) -> core::math::Random
```

New random number generator

random_from_seed

```
pub sync fn random_from_seed ( seed : int ) -> core::math::Random
```

New random number generator from given seed

Parameters

Name	Type	Optional	Description
seed	int		

round

```
pub sync fn round ( a : float ) -> float
```

Rounds a decimal value to the nearest integral value, and rounds midpoint values to the nearest even number.

Parameters

Name	Type	Optional	Description
a	float		

sin

```
pub sync fn sin ( a : float ) -> float
```

Returns the sine of the specified angle in redian.

Parameters

Name	Type	Optional	Description
a	float		

sin_deg

```
pub sync fn sin_deg ( x : float ) -> float
```

Returns the sine of the specified angle in degree.

Parameters

Name	Type	Optional	Description
x	float		

sinh

```
pub sync fn sinh ( value : float ) -> float
```

Returns the hyperbolic sine of the specified angle.

Parameters

Name	Type	Optional	Description
value	float		

sqrt

```
pub sync fn sqrt ( d : float ) -> float
```

Returns the square root of a specified number.

Parameters

Name	Type	Optional	Description
d	float		

tan

```
pub sync fn tan ( a : float ) -> float
```

Returns the sine of the specified angle in redian.

Parameters

Name	Type	Optional	Description
a	float		

tan_deg

```
pub sync fn tan_deg ( x : float ) -> float
```

Returns the sine of the specified angle in degree.

Parameters

Name	Type	Optional	Description
x	float		

tanh

```
pub sync fn tanh ( value : float ) -> float
```

Returns the hyperbolic tangent of the specified angle.

Parameters

Name	Type	Optional	Description
value	float		

truncate

```
pub sync fn truncate ( d : float ) -> float
```

Calculates the integral part of a specified number.

Parameters

Name	Type	Optional	Description
d	float		

4.5 core::str

Provided basic string manipulation and formatting.

4.5.1 Functions

format

```
pub sync fn format ( format : string,
                     items : T ) -> string
```

Format items using C# format strings (https://learn.microsoft.com/en-us/dotnet/api/system.string.format). Items can be either a single value, an array or a tuple.

Parameters

Name	Type	Optional	Description
format	string		
items	T		

join

```
pub sync fn join ( separator : string,
                   items : string[] ) -> string
```

Join an array of string with a separator.

Parameters

Name	Type	Optional	Description
separator	string		
items	string[]		

4.6 core::testing

Provides basic assertions for testing. All functions provided by this module should be only used by `test` function.

4.6.1 Functions

assert_false

```
pub sync fn assert_false ( actual : bool ) -> Unit
```

Assert that `actual` is false (Test only)

Parameters

Name	Type	Optional	Description
actual	bool		

assert_float

```
pub sync fn assert_float ( expected : float,
                           actual : float,
                           delta : float ) -> Unit
```

Assert that `actual` float is almost equal to `expected` with an absolute tolerance of `delta` (Test only)

Parameters

Name	Type	Optional	Description
expected	float		
actual	float		
delta	float	x	

assert_int

```
pub sync fn assert_int ( expected : int,
                         actual : int ) -> Unit
```

Assert that `actual` integer is equal to `expected` (Test only)

Parameters

Name	Type	Optional	Description
expected	int		
actual	int		

assert_none

```
pub sync fn assert_none ( actual : Option<T> ) -> Unit
```

Parameters

Name	Type	Optional	Description
actual	Option		

assert_some

```
pub sync fn assert_some ( expected : T,
                          actual : Option<T> ) -> Unit
```

Parameters

Name	Type	Optional	Description
expected	T		
actual	Option		

assert_string

```
pub sync fn assert_string ( expected : string,
                            actual : string ) -> Unit
```

Assert that `actual` string is equal to `expected` (Test only)

Parameters

Name	Type	Optional	Description
expected	string		
actual	string		

assert_true

```
pub sync fn assert_true ( actual : bool ) -> Unit
```

Assert that `actual` is true (Test only)

Parameters

Name	Type	Optional	Description
actual	bool		

assert_yield

```
pub sync fn assert_yield ( expected : int ) -> Unit
```

Assert that test case has yielded `expected` number of times already (Async test only)

Parameters

Name	Type	Optional	Description
expected	int		

fail_test

```
pub sync fn fail_test ( message : string ) -> Unit
```

Fail the test case with a `message` (Test only).

Parameters

Name	Type	Optional	Description
message	string		

test_sleep

```
pub sync fn test_sleep ( millis : int ) -> Unit
```

Suspend execution for `millis`

Parameters

Name	Type	Optional	Description
millis	int		

yield

```
pub fn yield ( ) -> Unit
```

Yield the test case (Async test only)

4.7 ksp::addons

Provides access to optional addons.

4.7.1 Types

FlightPlan

Fields

Name	Type	Read-only	Description
version	string	R/O	

Methods

circularize

```
flightplan.circularize ( burnUt : float,
                         burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

course_correction

```
flightplan.course_correction ( burnUt : float,
                               burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

ellipticize

```
flightplan.ellipticize ( burnUt : float,
                         newAp : float,
                         newPe : float,
                         burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
newAp	float		
newPe	float		
burnOffsetFactor	float	x	

hohmann_transfer

```
flightplan.hohmann_transfer ( burnUt : float,
                              burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

intercept_tgt

```
flightplan.intercept_tgt ( burnUt : float,
                           tgtUt : float,
                           burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
tgtUt	float		
burnOffsetFactor	float	x	

match_planes

```
flightplan.match_planes ( burnUt : float,
                          burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

match_velocity

```
flightplan.match_velocity ( burnUt : float,
                            burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

moon_return

```
flightplan.moon_return ( burnUt : float,
                         burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

planetary_xfer

```
flightplan.planetary_xfer ( burnUt : float,
                            burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
burnOffsetFactor	float	x	

set_inclination

```
flightplan.set_inclination ( burnUt : float,
                             inclination : float,
                             burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
inclination	float		
burnOffsetFactor	float	x	

set_new_ap

```
flightplan.set_new_ap ( burnUt : float,
                        newAp : float,
                        burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
newAp	float		
burnOffsetFactor	float	x	

set_new_lan

```
flightplan.set_new_lan ( burnUt : float,
                         newLanValue : float,
                         burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
newLanValue	float		
burnOffsetFactor	float	x	

set_new_pe

```
flightplan.set_new_pe ( burnUt : float,
                        newPe : float,
                        burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
newPe	float		
burnOffsetFactor	float	x	

set_new_sma

```
flightplan.set_new_sma ( burnUt : float,
                         newSma : float,
                         burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
newSma	float		
burnOffsetFactor	float	x	

set_node_longitude

```
flightplan.set_node_longitude ( burnUt : float,
                                newNodeLongValue : float,
                                burnOffsetFactor : float ) -> bool
```

Parameters

Name	Type	Optional	Description
burnUt	float		
newNodeLongValue	float		
burnOffsetFactor	float	x	

4.7.2 Functions

flight_plan

```
pub sync fn flight_plan ( ) -> Option<ksp::addons::FlightPlan>
```

Access the "Flight Plan" API (https://github.com/schlosrat/FlightPlan) Will be undefined if FlightPlan is not installed.

4.8 ksp::console

Provides functions to interact with the in-game KontrolSystem Console. As of now the console is output- and monochrome-only, this might change in the future.

Additionally there is support for displaying popup messages on the HUD.

4.8.1 Types

Console

Representation of a console

Fields

Name	Type	Read-only	Description
cursor_col	int	R/O	
cursor_row	int	R/O	

Methods

clear

```
console.clear ( ) -> Unit
```

Clear the console of all its content and move cursor to (0, 0).

clear_line

```
console.clear_line ( row : int ) -> Unit
```

Clear a line

Parameters

Name	Type	Optional	Description
row	int		

move_cursor

```
console.move_cursor ( row : int,
                      column : int ) -> Unit
```

Move the cursor to a give `row` and `column`.

Parameters

Name	Type	Optional	Description
row	int		
column	int		

print

```
console.print ( message : string ) -> Unit
```

Print a message at the current cursor position (and move cursor forward)

Parameters

Name	Type	Optional	Description
message	string		

print_at

```
console.print_at ( row : int,
                   column : int,
                   message : string ) -> Unit
```

Moves the cursor to the specified position, prints the message and restores the previous cursor position

Parameters

Name	Type	Optional	Description
row	int		
column	int		
message	string		

print_line

```
console.print_line ( message : string ) -> Unit
```

Print a message at the current cursor position and move cursor to the beginning of the next line.

Parameters

Name	Type	Optional	Description
message	string		

RgbaColor

Interface color with alpha channel.

Fields

Name	Type	Read-only	Description
alpha	float	R/O	
blue	float	R/O	
green	float	R/O	
red	float	R/O	

4.8.2 Constants

Name	Type	Description
BLACK	ksp::console::RgbaColor	Color black
BLUE	ksp::console::RgbaColor	Color blue
CONSOLE	ksp::console::Console	Main console
CYAN	ksp::console::RgbaColor	Color cyan
GREEN	ksp::console::RgbaColor	Color green
RED	ksp::console::RgbaColor	Color red
WHITE	ksp::console::RgbaColor	Color red
YELLOW	ksp::console::RgbaColor	Color yellow

4.8.3 Functions

color

```
pub sync fn color ( red : float,
                    green : float,
                    blue : float,
                    alpha : float ) -> ksp::console::RgbaColor
```

Create a new color from `red`, `green`, `blue` and `alpha` (0.0 - 1.0).

Parameters

Name	Type	Optional	Description
red	float		
green	float		
blue	float		
alpha	float	x	

4.9 ksp::control

4.9.1 Types

MovingAverage

Fields

Name	Type	Read-only	Description
last_sample_time	float	R/W	
mean	float	R/W	
mean_diff	float	R/W	
sample_limit	int	R/W	
value_count	int	R/O	

Methods

reset

```
movingaverage.reset ( ) -> Unit
```

update

```
movingaverage.update ( sampleTime : float,
                       value : float ) -> float
```

Parameters

Name	Type	Optional	Description
sampleTime	float		
value	float		

update_delta

```
movingaverage.update_delta ( deltaT : float,
                             input : float ) -> float
```

Parameters

Name	Type	Optional	Description
deltaT	float		
input	float		

PIDLoop

Fields

Name	Type	Read-only	Description
change_rate	float	R/W	
d_term	float	R/W	
error	float	R/W	
error_sum	float	R/W	
extra_unwind	bool	R/W	
i_term	float	R/W	
input	float	R/W	
kd	float	R/W	
ki	float	R/W	
kp	float	R/W	
last_sample_time	float	R/W	
max_output	float	R/W	
min_output	float	R/W	
output	float	R/W	
p_term	float	R/W	
setpoint	float	R/W	

Methods

reset_i

```
pidloop.reset_i ( ) -> Unit
```

Reset the integral part of the PID loop

update

```
pidloop.update ( sampleTime : float,
                 input : float ) -> float
```

Parameters

Name	Type	Optional	Description
sampleTime	float		
input	float		

update_delta

```
pidloop.update_delta ( deltaT : float,
                       input : float ) -> float
```

Parameters

Name	Type	Optional	Description
deltaT	float		
input	float		

RCSTranslateManager

Fields

Name	Type	Read-only	Description
translate	*ksp::math::Vec3*	R/W	

Methods

release

```
rcstranslatemanager.release ( ) -> Unit
```

resume

```
rcstranslatemanager.resume ( ) -> Unit
```

set_translate_provider

```
rcstranslatemanager.set_translate_provider ( newTranslateProvider : sync fn(float) ->␣
↪ksp::math::Vec3 ) -> Unit
```

Parameters

Name	Type	Optional	Description
newTranslateProvider	sync fn(float) -> ksp::math::Vec3		

SteeringManager

Fields

Name	Type	Read-only	Description
pitch_yaw_roll	*ksp::math::Vec3*	R/W	

Methods

release

```
steeringmanager.release ( ) -> Unit
```

resume

```
steeringmanager.resume ( ) -> Unit
```

set_pitch_yaw_roll_provider

```
steeringmanager.set_pitch_yaw_roll_provider ( newPitchYawRollProvider : sync fn(float) ->
↪ ksp::math::Vec3 ) -> Unit
```

Parameters

Name	Type	Optional	Description
newPitchYawRollProvider	sync fn(float) -> ksp::math::Vec3		

ThrottleManager

Fields

Name	Type	Read-only	Description
throttle	float	R/W	

Methods

release

```
throttlemanager.release ( ) -> Unit
```

resume

```
throttlemanager.resume ( ) -> Unit
```

set_throttle_provider

```
throttlemanager.set_throttle_provider ( newThrottleProvider : sync fn(float) -> float ) -
↪> Unit
```

Parameters

Name	Type	Optional	Description
newThrottleProvider	sync fn(float) -> float		

TorquePI

Fields

Name	Type	Read-only	Description
I	float	R/W	
loop	*ksp::control::PIDLoop*	R/W	
tr	float	R/W	
ts	float	R/W	

Methods

reset_i

```
torquepi.reset_i ( ) -> Unit
```

Reset the integral part of the PID loop

update

```
torquepi.update ( sampleTime : float,
                  input : float,
                  setpoint : float,
                  momentOfInertia : float,
                  maxOutput : float ) -> float
```

Parameters

Name	Type	Optional	Description
sampleTime	float		
input	float		
setpoint	float		
momentOfInertia	float		
maxOutput	float		

update_delta

```
torquepi.update_delta ( deltaT : float,
                        input : float,
                        setpoint : float,
                        momentOfInertia : float,
                        maxOutput : float ) -> float
```

Parameters

Name	Type	Optional	Description
deltaT	float		
input	float		
setpoint	float		
momentOfInertia	float		
maxOutput	float		

WheelSteeringManager

Fields

Name	Type	Read-only	Description
wheel_steer	float	R/W	

Methods

release

```
wheelsteeringmanager.release ( ) -> Unit
```

resume

```
wheelsteeringmanager.resume ( ) -> Unit
```

set_wheel_steer_provider

```
wheelsteeringmanager.set_wheel_steer_provider ( newWheelSteerProvider : sync fn(float) ->
↪ float ) -> Unit
```

Parameters

Name	Type	Optional	Description
newWheelSteerProvider	sync fn(float) -> float		

WheelThrottleManager

Fields

Name	Type	Read-only	Description
wheel_throttle	float	R/W	

Methods

release

```
wheelthrottlemanager.release ( ) -> Unit
```

resume

```
wheelthrottlemanager.resume ( ) -> Unit
```

set_wheel_throttle_provider

```
wheelthrottlemanager.set_wheel_throttle_provider ( newWheelThrottleProvider : sync␣
↪fn(float) -> float ) -> Unit
```

Parameters

Name	Type	Optional	Description
newWheelThrottleProvider	sync fn(float) -> float		

4.9.2 Functions

moving_average

```
pub sync fn moving_average ( sampleLimit : int ) -> ksp::control::MovingAverage
```

Create a new MovingAverage with given sample limit.

Parameters

Name	Type	Optional	Description
sampleLimit	int		

pid_loop

```
pub sync fn pid_loop ( kp : float,
                       ki : float,
                       kd : float,
                       minOutput : float,
                       maxOutput : float,
                       extraUnwind : bool ) -> ksp::control::PIDLoop
```

Create a new PIDLoop with given parameters.

Parameters

Name	Type	Optional	Description
kp	float		
ki	float		
kd	float		
minOutput	float		
maxOutput	float		
extraUnwind	bool	x	

torque_pi

```
pub sync fn torque_pi ( ts : float ) -> ksp::control::TorquePI
```

Create a new TorquePI with given parameters.

Parameters

Name	Type	Optional	Description
ts	float		

4.10 ksp::debug

Provides utility functions to draw in-game markers that can be helpful to visualize why an algorithm went haywire.

4.10.1 Types

Debug

Collection of debug helper

Methods

add_billboard

```
debug.add_billboard ( positionProvider : sync fn() -> ksp::math::GlobalPosition,
                      textProvider : sync fn() -> string,
                      color : ksp::console::RgbaColor,
                      fontSize : int ) -> ksp::debug::DebugBillboard
```

Parameters

Name	Type	Optional	Description
positionProvider	sync fn() -> ksp::math::GlobalPosition		
textProvider	sync fn() -> string		
color	ksp::console::RgbaColor		
fontSize	int		

add_ground_marker

```
debug.add_ground_marker ( geoCoordinates : ksp::orbit::GeoCoordinates,
                          color : ksp::console::RgbaColor,
                          rotation : float ) -> ksp::debug::GroundMarker
```

Parameters

Name	Type	Optional	Description
geoCoordinates	ksp::orbit::GeoCoordinates		
color	ksp::console::RgbaColor		
rotation	float		

add_line

```
debug.add_line ( startProvider : sync fn() -> ksp::math::GlobalPosition,
                 endProvider : sync fn() -> ksp::math::GlobalPosition,
                 color : ksp::console::RgbaColor,
                 label : string,
                 width : float ) -> ksp::debug::DebugVector
```

Draws a line from `start` to `end` with a specified `color` and `width` in the current game scene. The line may have a `label` at its mid-point.

Parameters

Name	Type	Optional	Description
startProvider	sync fn() -> ksp::math::GlobalPosition		
endProvider	sync fn() -> ksp::math::GlobalPosition		
color	ksp::console::RgbaColor		
label	string		
width	float		

add_path

```
debug.add_path ( path : ksp::math::GlobalPosition[],
                 color : ksp::console::RgbaColor,
                 width : float ) -> ksp::debug::DebugPath
```

Parameters

Name	Type	Optional	Description
path	ksp::math::GlobalPosition[]		
color	ksp::console::RgbaColor		
width	float		

add_vector

```
debug.add_vector ( startProvider : sync fn() -> ksp::math::GlobalPosition,
                   vectorProvider : sync fn() -> ksp::math::GlobalVector,
                   color : ksp::console::RgbaColor,
                   label : string,
                   width : float ) -> ksp::debug::DebugVector
```

Draws a `vector` positioned at `start` with a specified `color` and `width` in the current game scene. The vector may have a `label` at its mid-point.

Parameters

Name	Type	Optional	Description
startProvider	sync fn() -> ksp::math::GlobalPosition		
vectorProvider	sync fn() -> ksp::math::GlobalVector		
color	ksp::console::RgbaColor		
label	string		
width	float		

clear_markers

```
debug.clear_markers ( ) -> Unit
```

Remove all markers from the game-scene.

DebugBillboard

Represents a ground marker on a given celestial body.

Fields

Name	Type	Read-only	Description
color	*ksp::console::RgbaColor*	R/W	The color of the billboard text
font_size	int	R/W	
visible	bool	R/W	Controls if the billboard is currently visible (initially `true`)

Methods

remove

```
debugbillboard.remove ( ) -> Unit
```

DebugPath

Represents a debugging path in the current scene.

Fields

Name	Type	Read-only	Description
color	*ksp::console::RgbaColor*	R/W	The color of the debugging path
path	*ksp::math::GlobalPosition*[]	R/W	
visible	bool	R/W	Controls if the debug path is currently visible (initially `true`)
width	float	R/W	The width of the debugging path

Methods

remove

```
debugpath.remove ( ) -> Unit
```

DebugVector

Represents a debugging vector in the current scene.

Fields

Name	Type	Read-only	Description
color	*ksp::console::RgbaColor*	R/W	The color of the debugging vector
end	*ksp::math::GlobalPosition*	R/W	The current end position of the debugging vector.
pointy	bool	R/W	Controls if an arrow should be drawn at the end.
scale	float	R/W	
start	*ksp::math::GlobalPosition*	R/W	The current starting position of the debugging vector.
visible	bool	R/W	Controls if the debug-vector is currently visible (initially `true`)
width	float	R/W	The width of the debugging vector

Methods

remove

```
debugvector.remove ( ) -> Unit
```

set_end_provider

```
debugvector.set_end_provider ( endProvider : sync fn() -> ksp::math::GlobalPosition ) ->␣
↪Unit
```

Change the function providing the end position of the debug vector.

Parameters

Name	Type	Optional	Description
endProvider	sync fn() -> ksp::math::GlobalPosition		

set_start_provider

```
debugvector.set_start_provider ( startProvider : sync fn() -> ksp::math::GlobalPosition␣
↪) -> Unit
```

Change the function providing the start position of the debug vector.

Parameters

Name	Type	Optional	Description
startProvider	sync fn() -> ksp::math::GlobalPosition		

set_vector_provider

```
debugvector.set_vector_provider ( vectorProvider : sync fn() -> ksp::math::GlobalVector␣
↪) -> Unit
```

Change the function providing the vector/direction of the debug vector.

Parameters

Name	Type	Optional	Description
vectorProvider	sync fn() -> ksp::math::GlobalVector		

GroundMarker

Represents a ground marker on a given celestial body.

Fields

Name	Type	Read-only	Description
color	*ksp::console::RgbaColor*	R/W	The color of the ground marker vector
geo_coordinates	*ksp::orbit::GeoCoordinat*	R/W	
rotation	float	R/W	
visible	bool	R/W	Controls if the ground marker is currently visible (initially `true`)

Methods

remove

```
groundmarker.remove ( ) -> Unit
```

LogFile

Represents a log file.

Methods

log

```
logfile.log ( message : string ) -> Unit
```

Write a log message to the file.

Parameters

Name	Type	Optional	Description
message	string		

read_lines

```
logfile.read_lines ( ) -> string[]
```

Read all previous log entries

truncate

```
logfile.truncate ( ) -> Unit
```

Truncate/clear the log file.

SaveLoadControl

Control load/save of game

Fields

Name	Type	Read-only	Description
is_loaded	bool	R/O	
is_loading	bool	R/O	
is_saving	bool	R/O	
quick_load_allowed	bool	R/O	Check if quick load is allowed by the game settings

Methods

quick_load

```
saveloadcontrol.quick_load ( ) -> Unit
```

Trigger a quick load. Note: This will implicitly terminate all running scripts.

quick_save

```
saveloadcontrol.quick_save ( ) -> Unit
```

Trigger a quick save

try_recover_vessel

```
saveloadcontrol.try_recover_vessel ( ) -> bool
```

Try to recover the current vessel. Currently a vessel is only recoverable if splashed or landed on Kerbin. This will implicitly terminate the running script if successful.

4.10.2 Constants

Name	Type	Description
DEBUG	ksp::debug::Debug	Collection of debug helper
MAIN_LOG	ksp::debug::LogFile	Main script specific log file
SAVE_LOAD_CONTROL	ksp::debug::SaveLoadControl	Control save/load of game

4.10.3 Functions

open_log_file

```
pub sync fn open_log_file ( name : string ) -> ksp::debug::LogFile
```

Parameters

Name	Type	Optional	Description
name	string		

4.11 ksp::game

Collection to game and runtime related functions.

4.11.1 Types

EventProcessStarted

Process started event will be published to message bus when a process is started

Fields

Name	Type	Read-only	Description
arguments	string[]	R/O	Process start arguments
name	string	R/O	Process name

EventProcessStopped

Process stop event will be published to message bus when a process is stopped

Fields

Name	Type	Read-only	Description
error	Option<string>	R/O	Error message in case of abnormal termination
name	string	R/O	Process name

Importance

Importance of a notification

Methods

to_string

```
importance.to_string ( ) -> string
```

String representation of the number

ImportanceConstants

Fields

Name	Type	Read-only	Description
High	*ksp::game::Importance*	R/O	High
Low	*ksp::game::Importance*	R/O	Low
Medium	*ksp::game::Importance*	R/O	Medium
None	*ksp::game::Importance*	R/O	

Methods

from_string

```
importanceconstants.from_string ( value : string ) -> Option<ksp::game::Importance>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

Mainframe

Fields

Name	Type	Read-only	Description
available_processes	*ksp::game::Process*[]	R/O	
version	string	R/O	Version number of the KontrolSystem

Methods

find_process

```
mainframe.find_process ( name : string ) -> Option<ksp::game::Process>
```

Parameters

Name	Type	Optional	Description
name	string		

MessageBus

Shared message bus

Methods

publish

```
messagebus.publish ( message : T ) -> Unit
```

Publish a message to anyone interested (or the void)

Parameters

Name	Type	Optional	Description
message	T		

subscribe

```
messagebus.subscribe ( ) -> ksp::game::Subscription<T>
```

Create a subscription to a specific message type

Process

Fields

Name	Type	Read-only	Description
arguments	*ksp::game::ProcessArgument*[]	R/O	
is_running	bool	R/O	
name	string	R/O	

Methods

start

```
process.start ( forVessel : Option<ksp::vessel::Vessel>,
                arguments : string[] ) -> bool
```

Parameters

Name	Type	Optional	Description
forVessel	Optionksp::vessel::Vessel	x	
arguments	string[]	x	

stop

```
process.stop ( ) -> bool
```

ProcessArgument

Fields

Name	Type	Read-only	Description
default_value	string	R/O	
name	string	R/O	
type	string	R/O	

Subscription

Central message bus

Fields

Name	Type	Read-only	Description
has_messages	bool	R/O	Check if subscription has pending messages

Methods

peek

```
subscription.peek ( ) -> Option<T>
```

Peek for next message without consuming it

recv

```
subscription.recv ( ) -> Option<T>
```

Receive next message

unsubscribe

```
subscription.unsubscribe ( ) -> Unit
```

Unsubscribe from the message bus. No further messages will be received

4.11.2 Constants

Name	Type	Description
Importance	ksp::game::ImportanceConstants	Importance of a notification
MAINFRAME	ksp::game::Mainframe	KontrolSystem mainframe
MESSAGE_BUS	ksp::game::MessageBus	Shared message bus

4.11.3 Functions

current_time

```
pub sync fn current_time ( ) -> float
```

Get the current universal time (UT) in seconds from start.

notification_alert

```
pub sync fn notification_alert ( title : string,
                                 message : string,
                                 importance : ksp::game::Importance,
                                 duration : float ) -> Unit
```

Show an alert notification

Parameters

Name	Type	Optional	Description
title	string		
message	string		
importance	ksp::game::Importance	x	
duration	float	x	

notification_passive

```
pub sync fn notification_passive ( message : string,
                                   duration : float ) -> Unit
```

Show a passive notification

Parameters

Name	Type	Optional	Description
message	string		
duration	float	x	

sleep

```
pub fn sleep ( seconds : float ) -> Unit
```

Stop execution of given number of seconds (factions of a seconds are supported as well).

Parameters

Name	Type	Optional	Description
seconds	float		

wait_until

```
pub fn wait_until ( predicate : sync fn() -> bool ) -> Unit
```

Stop execution until a given condition is met.

Parameters

Name	Type	Optional	Description
predicate	sync fn() -> bool		

wait_while

```
pub fn wait_while ( predicate : sync fn() -> bool ) -> Unit
```

Stop execution as long as a given condition is met.

Parameters

Name	Type	Optional	Description
predicate	sync fn() -> bool		

yield

```
pub fn yield ( ) -> Unit
```

Yield execution to allow Unity to do some other stuff inbetween.

4.12 ksp::game::warp

Collection of functions to control time warp.

4.12.1 Functions

cancel

```
pub fn cancel ( ) -> Unit
```

Deprecated: use cancel_warp()

cancel_warp

```
pub sync fn cancel_warp ( ) -> Unit
```

Cancel time warp

current_index

```
pub sync fn current_index ( ) -> int
```

Deprecated: Use current_warp_index()

current_rate

```
pub sync fn current_rate ( ) -> float
```

Deprecated: Use current_warp_rate()

current_warp_index

```
pub sync fn current_warp_index ( ) -> int
```

Get the current warp index. Actual factor depends on warp mode.

current_warp_rate

```
pub sync fn current_warp_rate ( ) -> float
```

Get the current warp rate (i.e. actual time multiplier).

get_warp_rates

```
pub sync fn get_warp_rates ( ) -> float[]
```

Get all available warp rates

is_physics_time_warp

```
pub sync fn is_physics_time_warp ( ) -> bool
```

Check if time warp is still in physics mode

is_warping

```
pub sync fn is_warping ( ) -> bool
```

Check if time warp is currently active

max_warp_index

```
pub sync fn max_warp_index ( ) -> int
```

Get current maximum allowed time warp index.

set_warp_index

```
pub sync fn set_warp_index ( index : int ) -> Unit
```

Set the current time warp index.

Parameters

Name	Type	Optional	Description
index	int		

warp_to

```
pub sync fn warp_to ( ut : float ) -> Unit
```

Synchronized version of `warp_to`. Use with care.

Parameters

Name	Type	Optional	Description
ut	float		

4.13 ksp::math

Collection of KSP/Unity related mathematical functions.

4.13.1 Types

Direction

Represents the rotation from an initial coordinate system when looking down the z-axis and “up” being the y-axis

Fields

Name	Type	Read-only	Description
angle	float	R/O	The rotation angle around the axis in degrees
axis	*ksp::math::Vec3*	R/O	The rotation axis
euler	*ksp::math::Vec3*	R/W	Euler angles in degree of the rotation
inverse	*ksp::math::Direction*	R/O	Inverse direction
pitch	float	R/O	Pitch in degree
right_vector	*ksp::math::Vec3*	R/O	Right vector of the rotation
roll	float	R/O	Roll in degree
up_vector	*ksp::math::Vec3*	R/O	Up vector of the rotation
vector	*ksp::math::Vec3*	R/W	Fore vector of the rotation (i.e. looking/facing direction
yaw	float	R/O	Yaw in degree

Methods

to_global

```
direction.to_global ( frame : ksp::math::TransformFrame ) -> ksp::math::GlobalDirection
```

Associate this direction with a coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
direction.to_string ( ) -> string
```

Convert the direction to string

GlobalAngularVelocity

An angular velocity in space, that can be projected to a 3-dimensional vector in a specific frame of reference

Methods

relative_to

```
globalangularvelocity.relative_to ( frame : ksp::math::TransformFrame ) ->␣
↪ksp::math::GlobalVector
```

Get relative angular velocity to a frame of reference

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_fixed

```
globalangularvelocity.to_fixed ( frame : ksp::math::TransformFrame,
                                 decimals : int ) -> string
```

Convert angular velocity to string with fixed number of `decimals` in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference
decimals	int		Number of decimals

to_local

```
globalangularvelocity.to_local ( frame : ksp::math::TransformFrame ) -> ksp::math::Vec3
```

Get local angular velocity in a frame of reference

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
globalangularvelocity.to_string ( frame : ksp::math::TransformFrame ) -> string
```

Convert angular velocity to string in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

GlobalDirection

Represents the rotation from an initial coordinate system when looking down the z-axis and "up" being the y-axis

Fields

Name	Type	Read-only	Description
right_vector	*ksp::math::GlobalVector*	R/O	Right vector of the rotation
up_vector	*ksp::math::GlobalVector*	R/O	Up vector of the rotation
vector	*ksp::math::GlobalVector*	R/W	Fore vector of the rotation (i.e. looking/facing direction

Methods

euler

```
globaldirection.euler ( frame : ksp::math::TransformFrame ) -> ksp::math::Vec3
```

Get euler angles in a specific coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

pitch

```
globaldirection.pitch ( frame : ksp::math::TransformFrame ) -> float
```

Get pitch angle in a specific coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

roll

```
globaldirection.roll ( frame : ksp::math::TransformFrame ) -> float
```

Get roll angle in a specific coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_local

```
globaldirection.to_local ( frame : ksp::math::TransformFrame ) -> ksp::math::Direction
```

Get local direction in a coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
globaldirection.to_string ( frame : ksp::math::TransformFrame ) -> string
```

Convert the direction to string

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

yaw

```
globaldirection.yaw ( frame : ksp::math::TransformFrame ) -> float
```

Get yaw angle in a specific coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

GlobalPosition

A position in space that can be projected to a 3-dimensional vector in a specific coordinate system

Methods

distance

```
globalposition.distance ( other : ksp::math::GlobalPosition ) -> float
```

Calculate the distance of `other` position.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalPosition		Other position

distance_sqr

```
globalposition.distance_sqr ( other : ksp::math::GlobalPosition ) -> float
```

Calculate the squared distance of `other` position.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalPosition		Other position

lerp_to

```
globalposition.lerp_to ( other : ksp::math::GlobalPosition,
                         t : float ) -> ksp::math::GlobalPosition
```

Linear interpolate position between this and `other` position, where `t = 0.0` is this and `t = 1.0` is `other`.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalPosition		Other position
t	float		Relative position of mid-point (0.0 - 1.0)

to_fixed

```
globalposition.to_fixed ( frame : ksp::math::TransformFrame,
                          decimals : int ) -> string
```

Convert the vector to string with fixed number of `decimals` in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference
decimals	int		Number of decimals

to_local

```
globalposition.to_local ( frame : ksp::math::TransformFrame ) -> ksp::math::Vec3
```

Get local vector in a coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
globalposition.to_string ( frame : ksp::math::TransformFrame ) -> string
```

Convert vector to string in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

GlobalVector

Abstract vector in space that can be projected to a concrete 3-dimensional vector in a specific coordinate system

Fields

Name	Type	Read-only	Description
magnitude	float	R/O	Magnitude/length of the vector
normalized	*ksp::math::GlobalVector*	R/O	Normalized vector (i.e. scaled to length 1)
sqr_magnitude	float	R/O	Squared magnitude of the vector

Methods

cross

```
globalvector.cross ( other : ksp::math::GlobalVector ) -> ksp::math::GlobalVector
```

Calculate the cross/other product with `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalVector		Other vector

dot

```
globalvector.dot ( other : ksp::math::GlobalVector ) -> float
```

Calculate the dot/inner product with `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalVector		Other vector

exclude_from

```
globalvector.exclude_from ( other : ksp::math::GlobalVector ) -> ksp::math::GlobalVector
```

Exclude this from `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalVector		Other vector

lerp_to

```
globalvector.lerp_to ( other : ksp::math::GlobalVector,
                       t : float ) -> ksp::math::GlobalVector
```

Linear interpolate position between this and `other` vector, where `t = 0.0` is this and `t = 1.0` is `other`.

Parameters

Name	Type	Optional	Description
other	ksp::math::GlobalVector		Other vector
t	float		Relative position of mid-point (0.0 - 1.0)

to_direction

```
globalvector.to_direction ( ) -> ksp::math::GlobalDirection
```

Convert the vector to a rotation/direction in space.

to_fixed

```
globalvector.to_fixed ( frame : ksp::math::TransformFrame,
                        decimals : int ) -> string
```

Convert the vector to string with fixed number of `decimals` in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference
decimals	int		Number of decimals

to_local

```
globalvector.to_local ( frame : ksp::math::TransformFrame ) -> ksp::math::Vec3
```

Get local vector in a coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
globalvector.to_string ( frame : ksp::math::TransformFrame ) -> string
```

Convert vector to string in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

GlobalVelocity

A velocity in space, that can be projected to a 3-dimensional vector in a specific frame of reference

Fields

Name	Type	Read-only	Description
position	*ksp::math::GlobalPosition*	R/W	Position the velocity was measured at
vector	*ksp::math::GlobalVector*	R/W	Relative velocity vector

Methods

to_fixed

```
globalvelocity.to_fixed ( frame : ksp::math::TransformFrame,
                          decimals : int ) -> string
```

Convert the vector to string with fixed number of `decimals` in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference
decimals	int		Number of decimals

to_local

```
globalvelocity.to_local ( frame : ksp::math::TransformFrame ) -> ksp::math::Vec3
```

Get local velocity in a frame of reference

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_relative

```
globalvelocity.to_relative ( frame : ksp::math::TransformFrame ) ->␣
↪ksp::math::GlobalVector
```

Get relative velocity to frame of reference

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
globalvelocity.to_string ( frame : ksp::math::TransformFrame ) -> string
```

Convert vector to string in a given coordinate system.

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

Matrix2x2

A 2-dimensional matrix.

Fields

Name	Type	Read-only	Description
a	float	R/O	a
b	float	R/O	b
c	float	R/O	c
d	float	R/O	d
determinant	float	R/O	Get determinant of matrix
inverse	*ksp::math::Matrix2x2*	R/O	Invert matrix

TransformFrame

Representation of a coordinate frame of reference

Fields

Name	Type	Read-only	Description
back	*ksp::math::GlobalVector*	R/O	backward vector of the coordinate system
down	*ksp::math::GlobalVector*	R/O	down vector of the coordinate system
forward	*ksp::math::GlobalVector*	R/O	forward vector of the coordinate system
left	*ksp::math::GlobalVector*	R/O	left vector of the coordinate system
right	*ksp::math::GlobalVector*	R/O	right vector of the coordinate system
up	*ksp::math::GlobalVector*	R/O	up vector of the coordinate system

Methods

to_local_position

```
transformframe.to_local_position ( position : ksp::math::GlobalPosition ) ->␣
↪ksp::math::Vec3
```

Get local coordinates of a position

Parameters

Name	Type	Optional	Description
position	ksp::math::GlobalPosition		Position to transform

to_local_vector

```
transformframe.to_local_vector ( vector : ksp::math::GlobalVector ) -> ksp::math::Vec3
```

Get local coordinates of a vector

Parameters

Name	Type	Optional	Description
vector	ksp::math::GlobalVector		Vector to transform

to_local_velocity

```
transformframe.to_local_velocity ( velocity : ksp::math::GlobalVelocity ) ->␣
↪ksp::math::Vec3
```

Get local coordinates of a velocity

Parameters

Name	Type	Optional	Description
velocity	ksp::math::GlobalVelocity		Velocity to transform

Vec2

A 2-dimensional vector.

Fields

Name	Type	Read-only	Description
magnitude	float	R/O	Magnitude/length of the vector
normalized	*ksp::math::Vec2*	R/O	Normalized vector (i.e. scaled to length 1)
sqr_magnitude	float	R/O	Squared magnitude of the vector
x	float	R/W	x-coordinate
y	float	R/W	y-coordinate

Methods

angle_to

```
vec2.angle_to ( other : ksp::math::Vec2 ) -> float
```

Calculate the angle in degree to `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec2		Other vector

to_fixed

```
vec2.to_fixed ( decimals : int ) -> string
```

Convert the vector to string with fixed number of `decimals`.

Parameters

Name	Type	Optional	Description
decimals	int		Number of decimals

to_string

```
vec2.to_string ( ) -> string
```

Convert the vector to string

Vec3

A 3-dimensional vector.

Fields

Name	Type	Read-only	Description
magnitude	float	R/O	Magnitude/length of the vector
normalized	*ksp::math::Vec3*	R/O	Normalized vector (i.e. scaled to length 1)
sqr_magnitude	float	R/O	Squared magnitude of the vector
x	float	R/W	x-coordinate
xzy	*ksp::math::Vec3*	R/O	Swapped y- and z-coordinate
y	float	R/W	y-coordinate
z	float	R/W	z-coordinate

Methods

angle_to

```
vec3.angle_to ( other : ksp::math::Vec3 ) -> float
```

Calculate the angle in degree to `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector

cross

```
vec3.cross ( other : ksp::math::Vec3 ) -> ksp::math::Vec3
```

Calculate the cross/other product with `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector

distance_to

```
vec3.distance_to ( other : ksp::math::Vec3 ) -> float
```

Calculate the distance between this and `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector

dot

```
vec3.dot ( other : ksp::math::Vec3 ) -> float
```

Calculate the dot/inner product with `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector

exclude_from

```
vec3.exclude_from ( other : ksp::math::Vec3 ) -> ksp::math::Vec3
```

Exclude this from `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector

lerp_to

```
vec3.lerp_to ( other : ksp::math::Vec3,
               t : float ) -> ksp::math::Vec3
```

Linear interpolate position between this and `other` vector, where `t = 0.0` is this and `t = 1.0` is `other`.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector
t	float		Relative position of mid-point (0.0 - 1.0)

project_to

```
vec3.project_to ( other : ksp::math::Vec3 ) -> ksp::math::Vec3
```

Project this on `other` vector.

Parameters

Name	Type	Optional	Description
other	ksp::math::Vec3		Other vector

to_direction

```
vec3.to_direction ( ) -> ksp::math::Direction
```

Point in direction of this vector.

to_fixed

```
vec3.to_fixed ( decimals : int ) -> string
```

Convert the vector to string with fixed number of `decimals`.

Parameters

Name	Type	Optional	Description
decimals	int		Number of decimals

to_global

```
vec3.to_global ( frame : ksp::math::TransformFrame ) -> ksp::math::GlobalVector
```

Associate this vector with a coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_position

```
vec3.to_position ( frame : ksp::math::TransformFrame ) -> ksp::math::GlobalPosition
```

Consider this vector as position in a coordinate system

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		Frame of reference

to_string

```
vec3.to_string ( ) -> string
```

Convert vector to string.

4.13.2 Functions

angle_axis

```
pub sync fn angle_axis ( angle : float,
                         axis : ksp::math::Vec3 ) -> ksp::math::Direction
```

Create a Direction from a given axis with rotation angle in degree

Parameters

Name	Type	Optional	Description
angle	float		
axis	ksp::math::Vec3		

angle_delta

```
pub sync fn angle_delta ( a : float,
                          b : float ) -> float
```

Calculate the difference between two angles in degree (-180 .. 180)

Parameters

Name	Type	Optional	Description
a	float		
b	float		

euler

```
pub sync fn euler ( x : float,
                    y : float,
                    z : float ) -> ksp::math::Direction
```

Create a Direction from euler angles in degree

Parameters

Name	Type	Optional	Description
x	float		
y	float		
z	float		

from_vector_to_vector

```
pub sync fn from_vector_to_vector ( v1 : ksp::math::Vec3,
                                    v2 : ksp::math::Vec3 ) -> ksp::math::Direction
```

Create a Direction to rotate from one vector to another

Parameters

Name	Type	Optional	Description
v1	ksp::math::Vec3		
v2	ksp::math::Vec3		

global_angle_axis

```
pub sync fn global_angle_axis ( angle : float,
                                axis : ksp::math::GlobalVector ) ->␣
↪ksp::math::GlobalDirection
```

Create a Direction from a given axis with rotation angle in degree

Parameters

Name	Type	Optional	Description
angle	float		
axis	ksp::math::GlobalVector		

global_euler

```
pub sync fn global_euler ( frame : ksp::math::TransformFrame,
                           x : float,
                           y : float,
                           z : float ) -> ksp::math::GlobalDirection
```

Create a Direction from euler angles in degree

Parameters

Name	Type	Optional	Description
frame	ksp::math::TransformFrame		
x	float		
y	float		
z	float		

global_from_vector_to_vector

```
pub sync fn global_from_vector_to_vector ( v1 : ksp::math::GlobalVector,
                                           v2 : ksp::math::GlobalVector ) ->␣
↪ksp::math::GlobalDirection
```

Create a Direction to rotate from one vector to another

Parameters

Name	Type	Optional	Description
v1	ksp::math::GlobalVector		
v2	ksp::math::GlobalVector		

global_look_dir_up

```
pub sync fn global_look_dir_up ( lookDirection : ksp::math::GlobalVector,
                                 upDirection : ksp::math::GlobalVector ) ->␣
↪ksp::math::GlobalDirection
```

Create a Direction from a fore-vector and an up-vector

Parameters

Name	Type	Optional	Description
lookDirection	ksp::math::GlobalVector		
upDirection	ksp::math::GlobalVector		

look_dir_up

```
pub sync fn look_dir_up ( lookDirection : ksp::math::Vec3,
                          upDirection : ksp::math::Vec3 ) -> ksp::math::Direction
```

Create a Direction from a fore-vector and an up-vector

Parameters

Name	Type	Optional	Description
lookDirection	ksp::math::Vec3		
upDirection	ksp::math::Vec3		

matrix2x2

```
pub sync fn matrix2x2 ( a : float,
                        b : float,
                        c : float,
                        d : float ) -> ksp::math::Matrix2x2
```

Create a new 2-dimensional matrix

Parameters

Name	Type	Optional	Description
a	float		
b	float		
c	float		
d	float		

matrix4x4

```
pub sync fn matrix4x4 ( ) -> ksp::math::Matrix4x4
```

Create a new 4-dimensional matrix

vec2

```
pub sync fn vec2 ( x : float,
                   y : float ) -> ksp::math::Vec2
```

Create a new 2-dimensional vector

Parameters

Name	Type	Optional	Description
x	float		
y	float		

vec3

```
pub sync fn vec3 ( x : float,
                   y : float,
                   z : float ) -> ksp::math::Vec3
```

Create a new 3-dimensional vector

Parameters

Name	Type	Optional	Description
x	float		
y	float		
z	float		

4.14 ksp::oab

Collection of types and functions to get information about the current object/vessel assembly.

4.14.1 Types

ObjectAssembly

Represents an object assembly, i.e. a potential vessel.

Fields

Name	Type	Read-only	Description
delta_v	*ksp::oab::ObjectAssemblyDel*	R/O	Collection of methods to obtain delta-v information of the assembly.
dry_mass	float	R/O	Total dry mass of assembly.
parts	*ksp::oab::ObjectAssemblyPar*	R/O	Get a list of all parts of assembly.
total_mass	float	R/O	Total mass of assembly.
wet_mass	float	R/O	Total wet mass of assembly.

ObjectAssemblyAirIntake

Fields

Name	Type	Read-only	Description
enabled	bool	R/O	Enable/disable module
flow_rate	float	R/O	Resource flow rate
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
resource	*ksp::resource::ResourceDefinition*	R/O	
resource_units	float	R/O	
toogle_intake	bool	R/W	Toggle air intake.

ObjectAssemblyBuilder

Represents the current object assembly builder.

Fields

Name	Type	Read-only	Description
assemblies	*ksp::oab::ObjectAssembly*[]	R/O	Get all object assemblies (i.e. all parts that are not fully connected)
main_assembl	Option<*ksp::oab::ObjectAssembl*	R/O	Get the current main assembly if there is one.

ObjectAssemblyCommand

Fields

Name	Type	Read-only	Description
control_state	*ksp::vessel::CommandControlState*	R/O	
has_hibernation	bool	R/O	
hibernation_multiplier	float	R/O	
is_hibernating	bool	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	

ObjectAssemblyControlSurface

Fields

Name	Type	Read-only	Description
angle_of_attack	float	R/O	
authority_limiter	float	R/W	
drag	float	R/O	
enable_pitch	bool	R/W	
enable_roll	bool	R/W	
enable_yaw	bool	R/W	
invert_control	bool	R/W	
lift	float	R/O	
lift_drag_ratio	float	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	

ObjectAssemblyDecoupler

Fields

Name	Type	Read-only	Description
ejection_impulse	float	R/W	
is_decoupled	bool	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	

ObjectAssemblyDeltaV

Delta V information of an object assembly

Fields

Name	Type	Read-only	Description
stages	*ksp::oab::ObjectAssemblyStageDeltaV*[]	R/O	

Methods

stage

```
objectassemblydeltav.stage ( stage : int ) -> Option<ksp::oab::ObjectAssemblyStageDeltaV>
```

Get delta-v information for a specific `stage` of the object assembly, if existent.

Parameters

Name	Type	Optional	Description
stage	int		

ObjectAssemblyDeployable

Fields

Name	Type	Read-only	Description
deploy_limit	float	R/W	
deploy_state	*ksp::vessel::DeployableDeployState*	R/O	
extendable	bool	R/O	
extended	bool	R/W	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
retractable	bool	R/O	

Methods

set_extended

```
objectassemblydeployable.set_extended ( extend : bool ) -> Unit
```

Parameters

Name	Type	Optional	Description
extend	bool		

ObjectAssemblyDockingNode

Fields

Name	Type	Read-only	Description
docking_state	*ksp::vessel::DockingState*	R/O	
is_deployable_docking_port	bool	R/O	
node_types	string[]	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	

ObjectAssemblyDrag

Fields

Name	Type	Read-only	Description
exposed_area	float	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
reference_area	float	R/O	
total_area	float	R/O	

ObjectAssemblyEngine

Fields

Name	Type	Read-only	Description
auto_switch_mode	bool	R/W	
current_engine_mode	*ksp::vessel::EngineMode*	R/O	Get the current engine mode
current_propellant	*ksp::resource::ResourceDefiniti*	R/O	Get the propellant of the current engine mode
current_throttle	float	R/O	
current_thrust	float	R/O	
engine_modes	*ksp::vessel::EngineMode*[]	R/O	Get all engine modes
has_ignited	bool	R/O	Check if engine has ignited
independent_throttle	float	R/W	Current independent throttle between 0.0 - 1.0
independent_throttle_enabled	bool	R/W	Toggle independent throttle
is_flameout	bool	R/O	Check if engine had a flame-out
is_operational	bool	R/O	Check if engine is operational
is_propellant_starved	bool	R/O	
is_shutdown	bool	R/O	Check if engine is shutdown
is_staged	bool	R/O	
max_fuel_flow	float	R/O	
max_thrust_output_atm	float	R/O	
max_thrust_output_vac	float	R/O	
min_fuel_flow	float	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
propellants	*ksp::resource::ResourceDefiniti*	R/O	Get the propellants of the different engine modes
real_isp	float	R/O	
throttle_min	float	R/O	
thrust_direction	*ksp::math::Vec3*	R/O	Direction of thrust
thrust_limiter	float	R/W	Current thrust limit value between 0.0 - 1.0

Methods

calc_max_thrust_output_atm

```
objectassemblyengine.calc_max_thrust_output_atm ( atmPressurekPa : float,
                                                  atmTemp : float,
                                                  atmDensity : float,
                                                  machNumber : float ) -> float
```

Calculate maximum thrust in atmosphere given atmospheric parameters

Parameters

Name	Type	Optional	Description
atmPressurekPa	float	x	
atmTemp	float	x	
atmDensity	float	x	
machNumber	float	x	

ObjectAssemblyEngineDeltaV

Fields

Name	Type	Read-only	Description
engine	*ksp::oab::ObjectAssemblyEngin*	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
start_burn_stage	int	R/O	Number of the stage when engine is supposed to start

Methods

get_ISP

```
objectassemblyenginedeltav.get_ISP ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated ISP of the engine in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_thrust

```
objectassemblyenginedeltav.get_thrust ( situation : ksp::vessel::DeltaVSituation ) ->␣
↪float
```

Estimated thrust of the engine in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

ObjectAssemblyExperiment

Fields

Name	Type	Read-only	Description
crew_required	int	R/O	
definition	*ksp::science::ExperimentDefinition*	R/O	Get the definition of the experiment.
experiment_id	string	R/O	
experiment_uses_resources	bool	R/O	
resources_cost	*ksp::resource::ResourceSetting*[]	R/O	
time_to_complete	float	R/O	

ObjectAssemblyGenerator

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
generator_output	float	R/O	
is_always_active	bool	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
resource_setting	*ksp::resource::ResourceSetting*	R/O	

ObjectAssemblyLiftingSurface

Fields

Name	Type	Read-only	Description
angle_of_attack	float	R/O	
drag_scalar	float	R/O	
lift_drag_ratio	float	R/O	
lift_scalar	float	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	

ObjectAssemblyLight

Fields

Name	Type	Read-only	Description
blink_enabled	bool	R/W	
blink_rate	float	R/W	
has_resources_to_operate	bool	R/O	
light_color	*ksp::math::Vec3*	R/W	
light_enabled	bool	R/W	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
pitch	float	R/W	
required_resource	*ksp::resource::ResourceSetting*	R/O	
rotation	float	R/W	

ObjectAssemblyPart

Represents are part in an object assembly.

Fields

Name	Type	Read-only	Description
activation_stage	int	R/O	
air_intake	Option<*ksp::oab::ObjectAssemblyAirIntake*>	R/O	
command_module	Option<*ksp::oab::ObjectAssemblyCommand*>	R/O	
control_surface	Option<*ksp::oab::ObjectAssemblyControlSurface*>	R/O	
decouple_stage	int	R/O	
decoupler	Option<*ksp::oab::ObjectAssemblyDecoupler*>	R/O	
deployable	Option<*ksp::oab::ObjectAssemblyDeployable*>	R/O	
docking_node	Option<*ksp::oab::ObjectAssemblyDockingNode*>	R/O	
drag	Option<*ksp::oab::ObjectAssemblyDrag*>	R/O	
dry_mass	float	R/O	Dry mass of the part
engine	Option<*ksp::oab::ObjectAssemblyEngine*>	R/O	
fuel_cross_feed	bool	R/O	
generator	Option<*ksp::oab::ObjectAssemblyGenerator*>	R/O	
green_mass	float	R/O	Green mass (Kerbals) of the part
is_decoupler	bool	R/O	

continues on next page

Table 1 – continued from previous page

Name	Type	Read-only	Description
is_deployable	bool	R/O	
is_drag	bool	R/O	
is_engine	bool	R/O	
is_generator	bool	R/O	
is_lifting_surface	bool	R/O	
is_light	bool	R/O	
is_rcs	bool	R/O	
is_reaction_wheel	bool	R/O	
is_s_science_experiment	bool	R/O	
is_solar_panel	bool	R/O	
is_transmitter	bool	R/O	
lifting_surface	Option<*ksp::oab::ObjectAssemblyLiftingSurface*>	R/O	
light	Option<*ksp::oab::ObjectAssemblyLight*>	R/O	
part_category	*ksp::vessel::PartCategory*	R/O	
part_description	string	R/O	
part_name	string	R/O	
part_title	string	R/O	
rcs	Option<*ksp::oab::ObjectAssemblyRCS*>	R/O	
reaction_wheel	Option<*ksp::oab::ObjectAssemblyReactionWheel*>	R/O	
relative_position	*ksp::math::Vec3*	R/O	
resources	*ksp::oab::ObjectAssemblyResource*[]	R/O	
science_experiment	Option<*ksp::oab::ObjectAssemblyScienceExperiment*>	R/O	
solar_panel	Option<*ksp::oab::ObjectAssemblySolarPanel*>	R/O	
total_mass	float	R/O	Total mass of the part
transmitter	Option<*ksp::oab::ObjectAssemblyTransmitter*>	R/O	
wet_mass	float	R/O	

ObjectAssemblyRCS

Fields

Name	Type	Read-only	Description
enable_pitch	bool	R/W	
enable_roll	bool	R/W	
enable_x	bool	R/W	
enable_y	bool	R/W	
enable_yaw	bool	R/W	
enable_z	bool	R/W	
enabled	bool	R/W	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
propellant	*ksp::resource::ResourceDefinition*	R/O	
thrust_limiter	float	R/W	

ObjectAssemblyReactionWheel

Fields

Name	Type	Read-only	Description
has_resources_to_operate	bool	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
potential_torque	*ksp::math::Vec3*	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	
toggle_torque	bool	R/W	
wheel_actuator_mode	*ksp::vessel::ActuatorMode*	R/W	
wheel_authority	float	R/W	
wheel_state	*ksp::vessel::ReactionWheelState*	R/O	

ObjectAssemblyResource

Fields

Name	Type	Read-only	Description
capacity_units	float	R/O	
resource	*ksp::resource::ResourceDefinition*	R/O	
stored_units	float	R/O	
total_mass	float	R/O	

ObjectAssemblyScienceExperiment

Fields

Name	Type	Read-only	Description
experiments	*ksp::oab::ObjectAssemblyExperiment*[]	R/O	
is_deployed	bool	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	

ObjectAssemblySolarPanel

Fields

Name	Type	Read-only	Description
base_flow_rate	float	R/O	Base flow rate
efficiency_multiplier	float	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
resource_setting	*ksp::resource::ResourceSetting*	R/O	

ObjectAssemblyStageDeltaV

Fields

Name	Type	Read-only	Description
active_engines	*ksp::oab::ObjectAssemblyEngineDeltaV*[]	R/O	
burn_time	float	R/O	Estimated burn time of the stage.
dry_mass	float	R/O	Dry mass of the stage.
end_mass	float	R/O	End mass of the stage.
engines	*ksp::oab::ObjectAssemblyEngineDeltaV*[]	R/O	
fuel_mass	float	R/O	Mass of the fuel in the stage.
parts	*ksp::oab::ObjectAssemblyPart*[]	R/O	
stage	int	R/O	The stage number.
start_mass	float	R/O	Start mass of the stage.

Methods

get_ISP

```
objectassemblystagedeltav.get_ISP ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated ISP of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_TWR

```
objectassemblystagedeltav.get_TWR ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated TWR of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_deltav

```
objectassemblystagedeltav.get_deltav ( situation : ksp::vessel::DeltaVSituation ) ->␣
↪float
```

Estimated delta-v of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_thrust

```
objectassemblystagedeltav.get_thrust ( situation : ksp::vessel::DeltaVSituation ) ->␣
↪float
```

Estimated thrust of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

ObjectAssemblyTransmitter

Fields

Name	Type	Read-only	Description
active_transmission_completed	float	R/O	
active_transmission_size	float	R/O	
communication_range	float	R/O	
data_packet_size	float	R/O	
data_transmission_interval	float	R/O	
has_resources_to_operate	bool	R/O	
is_transmitting	bool	R/O	
part	*ksp::oab::ObjectAssemblyPart*	R/O	
part_name	string	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	

4.14.2 Functions

active_object_assembly_builder

```
pub sync fn active_object_assembly_builder ( ) -> Result<ksp::oab::ObjectAssemblyBuilder>
```

Try to get the currently active vessel. Will result in an error if there is none.

4.15 ksp::orbit

4.15.1 Types

Body

Represents an in-game celestial body.

Fields

Name	Type	Read-only	Description
SOI_radius	float	R/O	Radius of the sphere of influence of the body
angular_velocity	*ksp::math::Vec3*	R/O	Angular velocity vector of the body
atmosphere_depth	float	R/O	Depth/height of the atmosphere if present.
body_frame	*ksp::math::TransformFra*	R/O	The body/rotating reference frame of the body.
celestial_frame	*ksp::math::TransformFra*	R/O	The celestial/non-rotating reference frame of the body.
global_angular_ve	*ksp::math::GlobalAngula*	R/O	Angular velocity vector of the body (coordinate system independent)
global_position	*ksp::math::GlobalPositio*	R/O	The current position of the body (coordinate system independent)
global_right	*ksp::math::GlobalVector*	R/O	Right vector of the body (coordinate system independent)
global_up	*ksp::math::GlobalVector*	R/O	Up vector of the body (coordinate system independent)
grav_parameter	float	R/O	Standard gravitation parameter of the body.
has_atmosphere	bool	R/O	`true` if the celestial body has an atmosphere to deal with.
name	string	R/O	Name of the celestial body.
orbit	*ksp::orbit::Orbit*	R/O	The orbit of the celestial body itself (around the parent body)
orbiting_bodies	*ksp::orbit::Body*[]	R/O	Get all celestial bodies in orbit around this body (aka child bodies).
parent_body	Option<*ksp::orbit::Body*>	R/O	Get the celestial body this celestial body orbits if it exists (aka the parent body).
position	*ksp::math::Vec3*	R/O	The current position of the body
radius	float	R/O	Radius of the body at sea level
right	*ksp::math::Vec3*	R/O	Right vector of the body in its celestial frame
rotation_period	float	R/O	Rotation period of the planet.
up	*ksp::math::Vec3*	R/O	Up vector of the body in its celestial frame
waypoints	*ksp::orbit::Waypoint*[]	R/O	List of all waypoints defined on the body

Methods

atmosphere_density

```
body.atmosphere_density ( altitude : float ) -> float
```

Get atmospheric density at a given `altitude`

Parameters

Name	Type	Optional	Description
altitude	float		

atmosphere_pressure_kpa

```
body.atmosphere_pressure_kpa ( altitude : float ) -> float
```

Get atmospheric pressure in kPa at a given `altitude`

Parameters

Name	Type	Optional	Description
altitude	float		

atmosphere_temperature

```
body.atmosphere_temperature ( altitude : float ) -> float
```

Get temperature of atmosphere at a given `altitude`

Parameters

Name	Type	Optional	Description
altitude	float		

create_orbit

```
body.create_orbit ( position : ksp::math::Vec3,
                    velocity : ksp::math::Vec3,
                    ut : float ) -> ksp::orbit::Orbit
```

Create a new orbit around this body starting at a given relative `position` and `velocity` at universal time `ut`

Parameters

Name	Type	Optional	Description
position	ksp::math::Vec3		
velocity	ksp::math::Vec3		
ut	float		

geo_coordinates

```
body.geo_coordinates ( latitude : float,
                       longitude : float ) -> ksp::orbit::GeoCoordinates
```

Get `GeoCoordinates` struct representing a `latitude` and `longitude` of the body

Parameters

Name	Type	Optional	Description
latitude	float		Latitude in degrees
longitude	float		Longitude in degrees

global_create_orbit

```
body.global_create_orbit ( velocity : ksp::math::GlobalVelocity,
                           ut : float ) -> ksp::orbit::Orbit
```

Create a new orbit around this body starting at a given a coordinate independent `velocity` at universal time `ut`

Parameters

Name	Type	Optional	Description
velocity	ksp::math::GlobalVelocity		
ut	float		

global_surface_normal

```
body.global_surface_normal ( latitude : float,
                             longitude : float ) -> ksp::math::GlobalVector
```

Get the surface normal at a `latitude` and `longitude` (i.e. the vector pointing up at this geo coordinate, coordinate system independent)

Parameters

Name	Type	Optional	Description
latitude	float		Latitude in degrees
longitude	float		Longitude in degrees

global_surface_position

```
body.global_surface_position ( latitude : float,
                               longitude : float,
                               altitude : float ) -> ksp::math::GlobalPosition
```

Position of a `latitude` and `longitude` at an altitude relative to sea-level (coordinate system independent)

Parameters

Name	Type	Optional	Description
latitude	float		Latitude in degrees
longitude	float		Longitude in degrees
altitude	float		Altitude relative to sea-level

surface_normal

```
body.surface_normal ( latitude : float,
                      longitude : float ) -> ksp::math::Vec3
```

Get the surface normal at a `latitude` and `longitude` (i.e. the vector pointing up at this geo coordinate)

Parameters

Name	Type	Optional	Description
latitude	float		Latitude in degrees
longitude	float		Longitude in degrees

surface_position

```
body.surface_position ( latitude : float,
                        longitude : float,
                        altitude : float ) -> ksp::math::Vec3
```

Position of a `latitude` and `longitude` at an altitude relative to sea-level in the celestial frame of the body

Parameters

Name	Type	Optional	Description
latitude	float		Latitude in degrees
longitude	float		Longitude in degrees
altitude	float		Altitude relative to sea-level

terrain_height

```
body.terrain_height ( latitude : float,
                      longitude : float ) -> float
```

Height of the terrain relative to sea-level at `latitude` and `longitude`

Parameters

Name	Type	Optional	Description
latitude	float		Latitude in degrees
longitude	float		Longitude in degrees

GeoCoordinates

Represents a geo coordinate (longitude, latitude) of a specific celestial body.

Fields

Name	Type	Read-only	Description
body	*ksp::orbit::Body*	R/O	The celestial body the geo coordinate is based on.
global_surface_norn	*ksp::math::GlobalVec*	R/O	Coordinate system independent surface normal (i.e. up vector)
latitude	float	R/W	Latitude in degrees
longitude	float	R/W	Longitude in degrees
surface_normal	*ksp::math::Vec3*	R/O	The surface normal (i.e. up vector) in the celestial frame of the body
terrain_height	float	R/O	Height of the terrain relative to sea-level

Methods

altitude_position

```
geocoordinates.altitude_position ( altitude : float ) -> ksp::math::Vec3
```

Position of the geo coordinate in the celestial frame of the body

Parameters

Name	Type	Optional	Description
altitude	float		Altitude relative to sea-level

global_altitude_position

```
geocoordinates.global_altitude_position ( altitude : float ) -> ksp::math::GlobalPosition
```

Coordinate system independent position of the geo coordinate

Parameters

Name	Type	Optional	Description
altitude	float		Altitude relative to sea-level

Orbit

Represents an in-game orbit.

Fields

Name	Type	Read-only	Description
LAN	float	R/O	Longitude of ascending node of the orbit in degree
apoapsis	Option<float>	R/O	Apoapsis of the orbit above sealevel of the `reference_body`. Is not defined for a hyperbolic orbit
apoapsis_radius	Option<float>	R/O	Radius of apoapsis of the orbit (i.e. from the center of the `reference_body'). Is not defined for a hyperbolic orbit
argument_of_peri	float	R/O	Argument of periapsis of the orbit.
eccentricity	float	R/O	Eccentricity of the orbit.
epoch	float	R/O	Orbit epoch.
global_orbit_	*ksp::math::G*	R/O	Get the coordinate independent normal vector of the orbit
inclination	float	R/O	Inclination of the orbit in degree.
mean_anoma	float	R/O	Mean anomaly of the orbit at `epoch`
mean_motior	float	R/O	Mean motion of the orbit.
orbit_normal	*ksp::math::Ve*	R/O	Normal vector perpendicular to orbital plane.
periapsis	float	R/O	Periapsis of the orbit above sealevel of the `reference_body`
periapsis_radius	float	R/O	Radius of periapsis of the orbit (i.e. from the center of the `reference_body')
period	float	R/O	Orbital period.
reference_body	*ksp::orbit::Bo*	R/O	The celestial body the orbit is referenced on.
reference_frame	*ksp::math::Tr*	R/O	Internal reference frame of the orbit. This might be useful to compare numbers. Note: All relative vectors are in the celestial frame of the `reference_body` which might be different!
relative_ascendir	*ksp::math::Ve*	R/O	Get the relative position of the ascending node.
relative_eccentri	*ksp::math::Ve*	R/O	Get the relative eccentricity vector.
semi_major_	float	R/O	Semi major axis of the orbit.

Methods

ascending_node_true_anomaly

```
orbit.ascending_node_true_anomaly ( b : ksp::orbit::Orbit ) -> float
```

Gives the true anomaly (in a's orbit) at which a crosses its ascending node with b's orbit. The returned value is always between 0 and 360.

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		

descending_node_true_anomaly

```
orbit.descending_node_true_anomaly ( b : ksp::orbit::Orbit ) -> float
```

Gives the true anomaly (in a's orbit) at which a crosses its descending node with b's orbit. The returned value is always between 0 and 360.

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		

get_eccentric_anomaly_at_true_anomaly

```
orbit.get_eccentric_anomaly_at_true_anomaly ( trueAnomaly : float ) -> float
```

Converts a true anomaly into an eccentric anomaly. For elliptical orbits this returns a value between 0 and 2pi. For hyperbolic orbits the returned value can be any number.

Parameters

Name	Type	Optional	Description
trueAnomaly	float		

get_mean_anomaly_at_eccentric_anomaly

```
orbit.get_mean_anomaly_at_eccentric_anomaly ( ecc : float ) -> float
```

Converts an eccentric anomaly into a mean anomaly. For an elliptical orbit, the returned value is between 0 and 2pi. For a hyperbolic orbit, the returned value is any number.

Parameters

Name	Type	Optional	Description
ecc	float		

global_position

```
orbit.global_position ( ut : float ) -> ksp::math::GlobalPosition
```

Get the coordinate independent position at a given universal time `ut`. Note: This takes the motion of the parent body into account.

Parameters

Name	Type	Optional	Description
ut	float		

global_position_for_true_anomaly

```
orbit.global_position_for_true_anomaly ( trueAnomaly : float ) ->␣
↪ksp::math::GlobalPosition
```

Get the coordinate independent position for a given `trueAnomaly`

Parameters

Name	Type	Optional	Description
trueAnomaly	float		

global_relative_position

```
orbit.global_relative_position ( ut : float ) -> ksp::math::GlobalVector
```

Shorthand for `orbit.global_position(ut) - orbit.reference_body.orbit.global_position(ut)`

Parameters

Name	Type	Optional	Description
ut	float		

global_velocity

```
orbit.global_velocity ( ut : float ) -> ksp::math::GlobalVelocity
```

Get the coordinate independent velocity at a given universal time `ut`. Note: This takes the motion of the parent body into account.

Parameters

Name	Type	Optional	Description
ut	float		

horizontal

```
orbit.horizontal ( ut : float ) -> ksp::math::Vec3
```

Relative horizontal vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

mean_anomaly_at_ut

```
orbit.mean_anomaly_at_ut ( ut : float ) -> float
```

The mean anomaly of the orbit. For elliptical orbits, the value return is always between 0 and 2pi. For hyperbolic orbits, the value can be any number.

Parameters

Name	Type	Optional	Description
ut	float		

next_apoapsis_time

```
orbit.next_apoapsis_time ( ut : Option<float> ) -> Option<float>
```

Returns the next time at which the orbiting object will be at apoapsis after a given universal time `ut`. If `ut` is omitted the current time will be used. For elliptical orbits, this will be between `ut` and `ut` + Period. For hyperbolic orbits, this is undefined.

Parameters

Name	Type	Optional	Description
ut	Option	x	

next_periapsis_time

```
orbit.next_periapsis_time ( ut : Option<float> ) -> float
```

The next time at which the orbiting object will be at periapsis after a given universal time `ut`. If `ut` is omitted the current time will be used. For elliptical orbits, this will be between `ut` and `ut` + Period. For hyperbolic orbits, this can be any time, including a time in the past, if the periapsis is in the past.

Parameters

Name	Type	Optional	Description
ut	Option	x	

next_time_of_radius

```
orbit.next_time_of_radius ( ut : float,
                            radius : float ) -> Option<float>
```

Finds the next time at which the orbiting object will achieve a given `radius` from center of the body after a given universal time `ut`. This will be undefined if the specified `radius` is impossible for this orbit, otherwise: For elliptical orbits this will be a time between `ut` and `ut` + period. For hyperbolic orbits this can be any time. If the given radius will be achieved in the future then the next time at which that radius will be achieved will be returned. If the given radius was only achieved in the past, then there are no guarantees about which of the two times in the past will be returned.

Parameters

Name	Type	Optional	Description
ut	float		
radius	float		

normal_plus

```
orbit.normal_plus ( ut : float ) -> ksp::math::Vec3
```

The relative normal-plus vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

orbital_velocity

```
orbit.orbital_velocity ( ut : float ) -> ksp::math::Vec3
```

Get the relative orbital velocity at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

perturbed_orbit

```
orbit.perturbed_orbit ( ut : float,
                        dV : ksp::math::Vec3 ) -> ksp::orbit::Orbit
```

Returns a new Orbit object that represents the result of applying a given relative `deltaV` to o at `ut`. Note: The resulting orbit is calculated as if the velocity change happens instantaneously, which might lead to unrealistic results for larger `deltaV`. The main use-case of this method is to be used as part of an orbit-optimization algorithm as it is quiet fast.

Parameters

Name	Type	Optional	Description
ut	float		
dV	ksp::math::Vec3		

prograde

```
orbit.prograde ( ut : float ) -> ksp::math::Vec3
```

The relative prograde vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

radial_plus

```
orbit.radial_plus ( ut : float ) -> ksp::math::Vec3
```

The relative radial-plus vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

radius

```
orbit.radius ( ut : float ) -> float
```

Get the orbital radius (distance from center of body) at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

relative_position

```
orbit.relative_position ( ut : float ) -> ksp::math::Vec3
```

Get relative position at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

relative_position_for_true_anomaly

```
orbit.relative_position_for_true_anomaly ( trueAnomaly : float ) -> ksp::math::Vec3
```

Get relative position for a given `trueAnomaly`

Parameters

Name	Type	Optional	Description
trueAnomaly	float		

synodic_period

```
orbit.synodic_period ( other : ksp::orbit::Orbit ) -> float
```

Computes the period of the phase angle between orbiting objects of this orbit and and `other` orbit. For noncircular orbits the time variation of the phase angle is only quasiperiodic and for high eccentricities and/or large relative inclinations, the relative motion is not really periodic at all.

Parameters

Name	Type	Optional	Description
other	ksp::orbit::Orbit		

time_of_ascending_node

```
orbit.time_of_ascending_node ( b : ksp::orbit::Orbit,
                               maybeUt : Option<float> ) -> float
```

Returns the next time at which a will cross its ascending node with b. For elliptical orbits this is a time between UT and UT + a.period. For hyperbolic orbits this can be any time, including a time in the past if the ascending node is in the past. NOTE: this function will throw an ArgumentException if a is a hyperbolic orbit and the ascending node occurs at a true anomaly that a does not actually ever attain. If `ut` is omitted the current time will be used.

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		
maybeUt	Option	x	

time_of_descending_node

```
orbit.time_of_descending_node ( b : ksp::orbit::Orbit,
                                maybeUt : Option<float> ) -> float
```

Returns the next time at which a will cross its descending node with b. For elliptical orbits this is a time between UT and UT + a.period. For hyperbolic orbits this can be any time, including a time in the past if the descending node is in the past. NOTE: this function will throw an ArgumentException if a is a hyperbolic orbit and the descending node occurs at a true anomaly that a does not actually ever attain. If `ut` is omitted the current time will be used

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		
maybeUt	Option	x	

time_of_true_anomaly

```
orbit.time_of_true_anomaly ( trueAnomaly : float,
                             maybeUt : Option<float> ) -> float
```

Next time of a certain true anomaly after a given universal time `ut`. If `ut` is omitted the current time will be used

Parameters

Name	Type	Optional	Description
trueAnomaly	float		
maybeUt	Option	x	

to_fixed

```
orbit.to_fixed ( decimals : int ) -> string
```

Convert orbital parameter to string using specified number of `decimals`

Parameters

Name	Type	Optional	Description
decimals	int		

to_string

```
orbit.to_string ( ) -> string
```

Convert orbital parameters to string.

true_anomaly_at_radius

```
orbit.true_anomaly_at_radius ( radius : float ) -> float
```

Get the true anomaly of a radius. If the radius is below the periapsis the true anomaly of the periapsis will be returned. If it is above the apoapsis the true anomaly of the apoapsis is returned. The returned value is always between 0 and 2pi.

Parameters

Name	Type	Optional	Description
radius	float		

true_anomaly_at_ut

```
orbit.true_anomaly_at_ut ( ut : float ) -> float
```

The true anomaly of the orbit at a given universal type `ut`. The vector is projected into the orbital plane and then the true anomaly is computed as the angle this vector makes with the vector pointing to the periapsis. The returned value is always between 0 and 2pi.

Parameters

Name	Type	Optional	Description
ut	float		

true_anomaly_from_vector

```
orbit.true_anomaly_from_vector ( vec : ksp::math::Vec3 ) -> float
```

Converts a relative direction, into a true anomaly. The vector is projected into the orbital plane and then the true anomaly is computed as the angle this vector makes with the vector pointing to the periapsis. The returned value is always between 0 and 2pi.

Parameters

Name	Type	Optional	Description
vec	ksp::math::Vec3		

up

```
orbit.up ( ut : float ) -> ksp::math::Vec3
```

Relative up vector of the orbit at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

ut_at_mean_anomaly

```
orbit.ut_at_mean_anomaly ( meanAnomaly : float,
                           ut : float ) -> float
```

The next time at which the orbiting object will reach the given mean anomaly. For elliptical orbits, this will be a time between UT and UT + o.period. For hyperbolic orbits, this can be any time, including a time in the past, if the given mean anomaly occurred in the past

Parameters

Name	Type	Optional	Description
meanAnomaly	float		
ut	float		

OrbitPatch

Represents a orbit patch of a trajectory

Fields

Name	Type	Read only	Description
LAN	float	R/O	Longitude of ascending node of the orbit in degree
apoapsis	Option<float>	R/O	Apoapsis of the orbit above sealevel of the `reference_body`. Is not defined for a hyperbolic orbit
apoapsis_radius	Option<float>	R/O	Radius of apoapsis of the orbit (i.e. from the center of the `reference_body'). Is not defined for a hyperbolic orbit
argument_of_per	float	R/O	Argument of periapsis of the orbit.
eccentricity	float	R/O	Eccentricity of the orbit.
end_transitio	*ksp::orbit::Pat*	R/O	Get transition type at the end of the orbit patch
end_ut	float	R/O	Universal time of the start of the orbit patch
epoch	float	R/O	Orbit epoch.
global_orbit_	*ksp::math::Glo*	R/O	Get the coordinate independent normal vector of the orbit
inclination	float	R/O	Inclination of the orbit in degree.
mean_anoma	float	R/O	Mean anomaly of the orbit at `epoch`
mean_motio	float	R/O	Mean motion of the orbit.
next_patch	Option<*ksp::orbi*	R/O	Get the next orbit patch of the trajectory (if available)
orbit_normal	*ksp::math::Vec*	R/O	Normal vector perpendicular to orbital plane.
periapsis	float	R/O	Periapsis of the orbit above sealevel of the `reference_body`
periapsis_radius	float	R/O	Radius of periapsis of the orbit (i.e. from the center of the `reference_body')
period	float	R/O	Orbital period.
previous_patch	Option<*ksp::orbi*	R/O	Get the previous orbit patch of the trajectory (if available)
reference_body	*ksp::orbit::Bod*	R/O	The celestial body the orbit is referenced on.
reference_frame	*ksp::math::Tra*	R/O	Internal reference frame of the orbit. This might be useful to compare numbers. Note: All relative vectors are in the celestial frame of the `reference_body` which might be different!
relative_ascendi	*ksp::math::Vec*	R/O	Get the relative position of the ascending node.
relative_eccentri	*ksp::math::Vec*	R/O	Get the relative eccentricity vector.
semi_major_	float	R/O	Semi major axis of the orbit.
start_transitio	*ksp::orbit::Pat*	R/O	Get transition type at the beginning of the orbit patch
start_ut	float	R/O	Universal time of the start of the orbit patch
trajectory	*ksp::orbit::Tra*	R/O	The trajectory this orbit patch belongs to

Methods

ascending_node_true_anomaly

```
orbitpatch.ascending_node_true_anomaly ( b : ksp::orbit::Orbit ) -> float
```

Gives the true anomaly (in a's orbit) at which a crosses its ascending node with b's orbit. The returned value is always between 0 and 360.

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		

descending_node_true_anomaly

```
orbitpatch.descending_node_true_anomaly ( b : ksp::orbit::Orbit ) -> float
```

Gives the true anomaly (in a's orbit) at which a crosses its descending node with b's orbit. The returned value is always between 0 and 360.

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		

get_eccentric_anomaly_at_true_anomaly

```
orbitpatch.get_eccentric_anomaly_at_true_anomaly ( trueAnomaly : float ) -> float
```

Converts a true anomaly into an eccentric anomaly. For elliptical orbits this returns a value between 0 and 2pi. For hyperbolic orbits the returned value can be any number.

Parameters

Name	Type	Optional	Description
trueAnomaly	float		

get_mean_anomaly_at_eccentric_anomaly

```
orbitpatch.get_mean_anomaly_at_eccentric_anomaly ( ecc : float ) -> float
```

Converts an eccentric anomaly into a mean anomaly. For an elliptical orbit, the returned value is between 0 and 2pi. For a hyperbolic orbit, the returned value is any number.

Parameters

Name	Type	Optional	Description
ecc	float		

global_position

```
orbitpatch.global_position ( ut : float ) -> ksp::math::GlobalPosition
```

Get the coordinate independent position at a given universal time `ut`. Note: This takes the motion of the parent body into account.

Parameters

Name	Type	Optional	Description
ut	float		

global_position_for_true_anomaly

```
orbitpatch.global_position_for_true_anomaly ( trueAnomaly : float ) ->␣
↪ksp::math::GlobalPosition
```

Get the coordinate independent position for a given `trueAnomaly`

Parameters

Name	Type	Optional	Description
trueAnomaly	float		

global_relative_position

```
orbitpatch.global_relative_position ( ut : float ) -> ksp::math::GlobalVector
```

Shorthand for `orbit.global_position(ut) - orbit.reference_body.orbit.global_position(ut)`

Parameters

Name	Type	Optional	Description
ut	float		

global_velocity

```
orbitpatch.global_velocity ( ut : float ) -> ksp::math::GlobalVelocity
```

Get the coordinate independent velocity at a given universal time `ut`. Note: This takes the motion of the parent body into account.

Parameters

Name	Type	Optional	Description
ut	float		

horizontal

```
orbitpatch.horizontal ( ut : float ) -> ksp::math::Vec3
```

Relative horizontal vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

mean_anomaly_at_ut

```
orbitpatch.mean_anomaly_at_ut ( ut : float ) -> float
```

The mean anomaly of the orbit. For elliptical orbits, the value return is always between 0 and 2pi. For hyperbolic orbits, the value can be any number.

Parameters

Name	Type	Optional	Description
ut	float		

next_apoapsis_time

```
orbitpatch.next_apoapsis_time ( maybeUt : Option<float> ) -> Option<float>
```

Returns the next time at which the orbiting object will be at apoapsis after a given universal time `ut`. If `ut` is omitted the current time will be used. For elliptical orbits, this will be between `ut` and `ut` + Period. For hyperbolic orbits, this is undefined.

Parameters

Name	Type	Optional	Description
maybeUt	Option	x	

next_periapsis_time

```
orbitpatch.next_periapsis_time ( maybeUt : Option<float> ) -> float
```

The next time at which the orbiting object will be at periapsis after a given universal time `ut`. If `ut` is omitted the current time will be used. For elliptical orbits, this will be between `ut` and `ut` + Period. For hyperbolic orbits, this can be any time, including a time in the past, if the periapsis is in the past.

Parameters

Name	Type	Optional	Description
maybeUt	Option	x	

next_time_of_radius

```
orbitpatch.next_time_of_radius ( ut : float,
                                 radius : float ) -> Option<float>
```

Finds the next time at which the orbiting object will achieve a given `radius` from center of the body after a given universal time `ut`. This will be undefined if the specified `radius` is impossible for this orbit, otherwise: For elliptical orbits this will be a time between `ut` and `ut` + period. For hyperbolic orbits this can be any time. If the given radius will be achieved in the future then the next time at which that radius will be achieved will be returned. If the given radius was only achieved in the past, then there are no guarantees about which of the two times in the past will be returned.

Parameters

Name	Type	Optional	Description
ut	float		
radius	float		

normal_plus

```
orbitpatch.normal_plus ( ut : float ) -> ksp::math::Vec3
```

The relative normal-plus vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

orbital_velocity

```
orbitpatch.orbital_velocity ( ut : float ) -> ksp::math::Vec3
```

Get the relative orbital velocity at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

perturbed_orbit

```
orbitpatch.perturbed_orbit ( ut : float,
                             dV : ksp::math::Vec3 ) -> ksp::orbit::Orbit
```

Returns a new Orbit object that represents the result of applying a given relative `deltaV` to o at `ut`. Note: The resulting orbit is calculated as if the velocity change happens instantaneously, which might lead to unrealistic results for larger `deltaV`. The main use-case of this method is to be used as part of an orbit-optimization algorithm as it is quiet fast.

Parameters

Name	Type	Optional	Description
ut	float		
dV	ksp::math::Vec3		

prograde

```
orbitpatch.prograde ( ut : float ) -> ksp::math::Vec3
```

The relative prograde vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

radial_plus

```
orbitpatch.radial_plus ( ut : float ) -> ksp::math::Vec3
```

The relative radial-plus vector at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

radius

```
orbitpatch.radius ( ut : float ) -> float
```

Get the orbital radius (distance from center of body) at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

relative_position

```
orbitpatch.relative_position ( ut : float ) -> ksp::math::Vec3
```

Get relative position at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

relative_position_for_true_anomaly

```
orbitpatch.relative_position_for_true_anomaly ( trueAnomaly : float ) -> ksp::math::Vec3
```

Get relative position for a given `trueAnomaly`

Parameters

Name	Type	Optional	Description
trueAnomaly	float		

synodic_period

```
orbitpatch.synodic_period ( other : ksp::orbit::Orbit ) -> float
```

Computes the period of the phase angle between orbiting objects of this orbit and and `other` orbit. For noncircular orbits the time variation of the phase angle is only quasiperiodic and for high eccentricities and/or large relative inclinations, the relative motion is not really periodic at all.

Parameters

Name	Type	Optional	Description
other	ksp::orbit::Orbit		

time_of_ascending_node

```
orbitpatch.time_of_ascending_node ( b : ksp::orbit::Orbit,
                                    maybeUt : Option<float> ) -> float
```

Returns the next time at which a will cross its ascending node with b. For elliptical orbits this is a time between UT and UT + a.period. For hyperbolic orbits this can be any time, including a time in the past if the ascending node is in the past. NOTE: this function will throw an ArgumentException if a is a hyperbolic orbit and the ascending node occurs at a true anomaly that a does not actually ever attain. If `ut` is omitted the current time will be used.

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		
maybeUt	Option	x	

time_of_descending_node

```
orbitpatch.time_of_descending_node ( b : ksp::orbit::Orbit,
                                     maybeUt : Option<float> ) -> float
```

Returns the next time at which a will cross its descending node with b. For elliptical orbits this is a time between UT and UT + a.period. For hyperbolic orbits this can be any time, including a time in the past if the descending node is in the past. NOTE: this function will throw an ArgumentException if a is a hyperbolic orbit and the descending node occurs at a true anomaly that a does not actually ever attain. If `ut` is omitted the current time will be used

Parameters

Name	Type	Optional	Description
b	ksp::orbit::Orbit		
maybeUt	Option	x	

time_of_true_anomaly

```
orbitpatch.time_of_true_anomaly ( trueAnomaly : float,
                                  maybeUt : Option<float> ) -> float
```

Next time of a certain true anomaly after a given universal time `ut`. If `ut` is omitted the current time will be used

Parameters

Name	Type	Optional	Description
trueAnomaly	float		
maybeUt	Option	x	

to_fixed

```
orbitpatch.to_fixed ( decimals : int ) -> string
```

Convert orbital parameter to string using specified number of `decimals`

Parameters

Name	Type	Optional	Description
decimals	int		

to_string

```
orbitpatch.to_string ( ) -> string
```

Convert orbital parameters to string.

true_anomaly_at_radius

```
orbitpatch.true_anomaly_at_radius ( radius : float ) -> float
```

Get the true anomaly of a radius. If the radius is below the periapsis the true anomaly of the periapsis will be returned. If it is above the apoapsis the true anomaly of the apoapsis is returned. The returned value is always between 0 and 2pi.

Parameters

Name	Type	Optional	Description
radius	float		

true_anomaly_at_ut

```
orbitpatch.true_anomaly_at_ut ( ut : float ) -> float
```

The true anomaly of the orbit at a given universal type `ut`. The vector is projected into the orbital plane and then the true anomaly is computed as the angle this vector makes with the vector pointing to the periapsis. The returned value is always between 0 and 2pi.

Parameters

Name	Type	Optional	Description
ut	float		

true_anomaly_from_vector

```
orbitpatch.true_anomaly_from_vector ( vec : ksp::math::Vec3 ) -> float
```

Converts a relative direction, into a true anomaly. The vector is projected into the orbital plane and then the true anomaly is computed as the angle this vector makes with the vector pointing to the periapsis. The returned value is always between 0 and 2pi.

Parameters

Name	Type	Optional	Description
vec	ksp::math::Vec3		

up

```
orbitpatch.up ( ut : float ) -> ksp::math::Vec3
```

Relative up vector of the orbit at a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

ut_at_mean_anomaly

```
orbitpatch.ut_at_mean_anomaly ( meanAnomaly : float,
                                ut : float ) -> float
```

The next time at which the orbiting object will reach the given mean anomaly. For elliptical orbits, this will be a time between UT and UT + o.period. For hyperbolic orbits, this can be any time, including a time in the past, if the given mean anomaly occurred in the past

Parameters

Name	Type	Optional	Description
meanAnomaly	float		
ut	float		

PatchTransitionType

Transition type at the beginning or end of an orbit patch

Methods

to_string

```
patchtransitiontype.to_string ( ) -> string
```

String representation of the number

PatchTransitionTypeConstants

Fields

Name	Type	Read-only	Description
Collision	*ksp::orbit::PatchTransitionType*	R/O	Orbits collides with a (celestial) object
CompletelyOutOf-Fuel	*ksp::orbit::PatchTransitionType*	R/O	Planed maneuver will run out of fuel
Encounter	*ksp::orbit::PatchTransitionType*	R/O	Orbit enters a sphere of influence (SOI)
EndThrust	*ksp::orbit::PatchTransitionType*	R/O	End of thrust of a planed maneuver
Escape	*ksp::orbit::PatchTransitionType*	R/O	Orbit leaves a sphere of influence (SOI)
Final	*ksp::orbit::PatchTransitionType*	R/O	Final transition (orbit ends here)
Initial	*ksp::orbit::PatchTransitionType*	R/O	Initial transition (orbit starts here)
Maneuver	*ksp::orbit::PatchTransitionType*	R/O	Orbit changes due to a planed maneuver
PartialOutOfFuel	*ksp::orbit::PatchTransitionType*	R/O	Planed maneuver will partially run out of fuel

Methods

from_string

```
patchtransitiontypeconstants.from_string ( value : string ) -> Option
↪<ksp::orbit::PatchTransitionType>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

Trajectory

Representation of a trajectory of a vessel that might has multiple orbit patches

Can be used in `for(... in ...)` loop to iterate over *ksp::orbit::OrbitPatch*

Can be used like an array *ksp::orbit::OrbitPatch*[]

Fields

Name	Type	Read-only	Description
end_ut	float	R/O	Universal time of the end of the trajectory
length	int	R/O	length
start_ut	float	R/O	Universal time of the start of the trajectory

Methods

exists

```
trajectory.exists ( predicate : sync fn(ksp::orbit::OrbitPatch) -> bool ) -> bool
```

Check if an item of the array matches `predicate`

Parameters

Name	Type	Optional	Description
predicate	sync fn(ksp::orbit::OrbitPatch) -> bool		Predicate function/check to be applied on each element of the array until element is found.

filter

```
trajectory.filter ( predicate : sync fn(ksp::orbit::OrbitPatch) -> bool ) -> T[]
```

Filter the content of the array by a `predicate

Parameters

Name	Type	Optional	Description
predicate	sync fn(ksp::orbit::OrbitPatch) -> bool		Predicate function/check to be applied on each element of the array

filter_map

```
trajectory.filter_map ( mapper : sync fn(ksp::orbit::OrbitPatch) -> Option<U> ) -> U[]
```

Map the content of the array

Parameters

Name	Type	Optional	Description
mapper	sync fn(ksp::orbit::OrbitPatch) -> Option		Function to be applied on each element of the array

find

```
trajectory.find ( predicate : sync fn(ksp::orbit::OrbitPatch) -> bool ) -> Option<T>
```

Find first item of the array matching `predicate`

Parameters

Name	Type	Optional	Description
predicate	sync fn(ksp::orbit::OrbitPatch) -> bool		Predicate function/check to be applied on each element of the array until element is found.

find_patch

```
trajectory.find_patch ( ut : float ) -> Option<ksp::orbit::OrbitPatch>
```

Find orbit patch for a given universal time `ut`

Parameters

Name	Type	Optional	Description
ut	float		

flat_map

```
trajectory.flat_map ( mapper : sync fn(ksp::orbit::OrbitPatch) -> U[] ) -> U[]
```

Map the content of the array

Parameters

Name	Type	Optional	Description
mapper	sync fn(ksp::orbit::OrbitPatch) -> U[]		Function to be applied on each element of the array

map

```
trajectory.map ( mapper : sync fn(ksp::orbit::OrbitPatch) -> U ) -> U[]
```

Map the content of the array

Parameters

Name	Type	Optional	Description
mapper	sync fn(ksp::orbit::OrbitPatch) -> U		Function to be applied on each element of the array

map_with_index

```
trajectory.map_with_index ( mapper : sync fn(ksp::orbit::OrbitPatch, int) -> U ) -> U[]
```

Map the content of the array

Parameters

Name	Type	Optional	Description
mapper	sync fn(ksp::orbit::OrbitPatch, int) -> U		Function to be applied on each element of the array including index of the element

reduce

```
trajectory.reduce ( initial : U,
                    reducer : sync fn(U, ksp::orbit::OrbitPatch) -> U ) -> U
```

Reduce array by an operation

Parameters

Name	Type	Optional	Description
initial	U		Initial value of the accumulator
reducer	sync fn(U, ksp::orbit::OrbitPatch) -> U		Combines accumulator with each element of the array and returns new accumulator value

reverse

```
trajectory.reverse ( ) -> ksp::orbit::OrbitPatch[]
```

Reverse the order of the array

Waypoint

Fields

Name	Type	Read-only	Description
altitude	float	R/O	Get altitude above sea-level of waypoint
body	*ksp::orbit::Body*	R/O	Celestial body the waypoint is based on
geo_coordinates	*ksp::orbit::GeoCoordinates*	R/O	Get `GeoCoordinates` of the waypoint
global_position	*ksp::math::GlobalPosition*	R/O	Coordinate system independent position of the waypoint
name	string	R/O	Name/label of the waypoint

4.15.2 Constants

Name	Type	Description
PatchTransitionType	ksp::orbit::PatchTransitionTypeConstan	Transition type at the beginning or end of an orbit patch

4.15.3 Functions

find_body

```
pub sync fn find_body ( name : string ) -> Result<ksp::orbit::Body>
```

Find a body by name.

Parameters

Name	Type	Optional	Description
name	string		

find_waypoint

```
pub sync fn find_waypoint ( name : string ) -> Result<ksp::orbit::Waypoint>
```

Find waypoint by name/label.

Parameters

Name	Type	Optional	Description
name	string		

galactic_origin

```
pub sync fn galactic_origin ( ) -> ksp::math::TransformFrame
```

Get the galactic celestial frame.

4.16 ksp::resource

Collection of types and functions to get information and manipulate in-game resources.

4.16.1 Types

FlowDirection

Resource flow direction

Methods

to_string

```
flowdirection.to_string ( ) -> string
```

String representation of the number

FlowDirectionConstants

Fields

Name	Type	Read-only	Description
FLOW_INBOUND	*ksp::resource::FlowDirec*	R/O	Inbound resource request (i.e demand resource from other parts)
FLOW_OUTBOUN	*ksp::resource::FlowDirec*	R/O	Outbound resource request (i.e. provide resource to other parts)

Methods

from_string

```
flowdirectionconstants.from_string ( value : string ) -> Option
↪<ksp::resource::FlowDirection>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ResourceContainer

Fields

Name	Type	Read-only	Description
list	*ksp::resource::ResourceData*[]	R/O	
stored_total_mass	float	R/O	
stored_total_thermal_mass	float	R/O	

Methods

dump_all

```
resourcecontainer.dump_all ( ) -> Unit
```

ResourceData

Fields

Name	Type	Read-only	Description
capacity_units	float	R/O	
resource	*ksp::resource::ResourceDefinition*	R/O	
stored_units	float	R/O	

ResourceDefinition

Represents an in-game resource.

Fields

Name	Type	Read-only	Description
display_abbreviatio	string	R/O	Resource abbreviation as displayed in UI
display_name	string	R/O	Name of the resource as displayed in UI
id	int	R/O	Resource identifier
is_recipe	bool	R/O	Check if resource is a recipe, i.e. a combination of resource
mass_per_unit	float	R/O	Mass per resource unit
mass_per_volum	float	R/O	Mass per volume aka. density
name	string	R/O	Name of the resource
recipe_ingredien	*Option<ksp::resource::ResourceReceip*	R/O	Get ingredients if resource is a recipe.
uses_air	bool	R/O	Check if resource requires air to be used.
volume_per_unit	float	R/O	Volume per resource unit

ResourceReceipeIngredient

Fields

Name	Type	Read-only	Description
resource	*ksp::resource::ResourceDefinition*	R/O	
units	float	R/O	

ResourceSetting

Fields

Name	Type	Read-only	Description
acceptance_threshold	float	R/O	
rate	float	R/O	
resource	*ksp::resource::ResourceDefinition*	R/O	

ResourceTransfer

Represents a resource transfer

Fields

Name	Type	Read-only	Description
entries	*ksp::resource::ResourceTransferEntry*[]	R/O	Get currently registers resource transfer entries.
is_running	bool	R/O	Check if a resource transfer is in progress.

Methods

add_container

```
resourcetransfer.add_container ( flowDirection : ksp::resource::FlowDirection,
                                 resourceContainer : ksp::resource::ResourceContainer,
                                 relativeAmount : float ) -> bool
```

Parameters

Name	Type	Optional	Description
flowDirection	ksp::resource::FlowDirection		
resourceContainer	ksp::resource::ResourceContainer		
relativeAmount	float	x	

add_resource

```
resourcetransfer.add_resource ( flowDirection : ksp::resource::FlowDirection,
                                resource : ksp::resource::ResourceData,
                                maxUnits : float ) -> bool
```

Parameters

Name	Type	Optional	Description
flowDirection	ksp::resource::FlowDirection		
resource	ksp::resource::ResourceData		
maxUnits	float		

clear

```
resourcetransfer.clear ( ) -> Unit
```

Cleanup all registered resource transfer entries. This will implicitly stop the resource transfer if it is still running.

start

```
resourcetransfer.start ( ) -> bool
```

Start the resource transfer.

stop

```
resourcetransfer.stop ( ) -> bool
```

Stop the resource transfer.

ResourceTransferEntry

Fields

Name	Type	Read-only	Description
flow_direction	*ksp::resource::FlowDirection*	R/O	
resource_container	*ksp::resource::ResourceContainer*	R/O	

4.16.2 Constants

Name	Type	Description
FlowDirection	ksp::resource::FlowDirectionConstants	Resource flow direction

4.16.3 Functions

create_resource_transfer

```
pub sync fn create_resource_transfer ( ) -> ksp::resource::ResourceTransfer
```

4.17 ksp::science

Collection of types and functions to get information and manipulate in-game science experiments.

4.17.1 Types

CompletedResearchReport

Represents a completed research report

Fields

Name	Type	Read-only	Description
definition	*ksp::science::ExperimentDefinition*	R/O	Get the definition of the experiment.
experiment_id	string	R/O	
research_location_id	string	R/O	
science_value	float	R/O	

Experiment

Represents an in-game science experiment.

Fields

Name	Type	Read-only	Description
crew_required	int	R/O	
current_experiment_state	*ksp::science::ExperimentState*	R/O	
current_running_time	float	R/O	
current_situation_is_vali	bool	R/O	
definition	*ksp::science::ExperimentDefiniti*	R/O	Get the definition of the experiment.
experiment_id	string	R/O	
experiment_location	Option<*ksp::science::ResearchLoc*	R/O	Get the research location the experiment was last performed.
experiment_uses_resources	bool	R/O	
has_enough_resource	bool	R/O	
previous_experiment_state	*ksp::science::ExperimentState*	R/O	
region_required	bool	R/O	
resources_cost	*ksp::resource::ResourceSetting*[]	R/O	
time_to_complete	float	R/O	
valid_locations	*ksp::science::ResearchLocation*[	R/O	

Methods

cancel_experiment

```
experiment.cancel_experiment ( ) -> bool
```

pause_experiment

```
experiment.pause_experiment ( ) -> bool
```

potential_science_value

```
experiment.potential_science_value ( ) -> float
```

run_experiment

```
experiment.run_experiment ( ) -> bool
```

ExperimentDefinition

Represents definition of an in-game science experiment.

Fields

Name	Type	Read-only	Description
data_value	float	R/O	
display_name	string	R/O	
id	string	R/O	
requires_eva	bool	R/O	
sample_value	float	R/O	
transmission_size	float	R/O	

ExperimentState

Science experiment state

Methods

to_string

```
experimentstate.to_string ( ) -> string
```

String representation of the number

ExperimentStateConstants

Fields

Name	Type	Read-only	Description
ALREADYSTORED	*ksp::science::ExperimentSt*	R/O	Experiment has already stored results
BLOCKED	*ksp::science::ExperimentSt*	R/O	Experiment is blocked
INSUFFICIENTCREW	*ksp::science::ExperimentSt*	R/O	Experiment requires more available crew members
INSUFFICIENTSTOR-AGE	*ksp::science::ExperimentSt*	R/O	Not enough storage capacity for experiment
INVALIDLOCATION	*ksp::science::ExperimentSt*	R/O	Location not valid
LOCATION-CHANGED	*ksp::science::ExperimentSt*	R/O	Experiment location changed
NOCONTROL	*ksp::science::ExperimentSt*	R/O	Experiment requires control of the vessel
NONE	*ksp::science::ExperimentSt*	R/O	Unknown state
OUTOFRESOURCE	*ksp::science::ExperimentSt*	R/O	Experiment ran out of resources
PAUSED	*ksp::science::ExperimentSt*	R/O	Experiment is paused
READY	*ksp::science::ExperimentSt*	R/O	Experiment is ready to run
RUNNING	*ksp::science::ExperimentSt*	R/O	Experiment is running

Methods

from_string

```
experimentstateconstants.from_string ( value : string ) -> Option
↪<ksp::science::ExperimentState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ResearchLocation

Represents a research location of a science experiment.

Fields

Name	Type	Read-only	Description
body_name	string	R/O	
id	string	R/O	
requires_region	bool	R/O	
science_region	string	R/O	
science_situation	*ksp::science::ScienceSituation*	R/O	

ResearchReport

Represents the stored report of a science experiment

Fields

Name	Type	Read-only	Description
definition	*ksp::science::ExperimentDefi*	R/O	Get the definition of the experiment.
ec_required	float	R/O	
experiment_id	string	R/O	
report_type	*ksp::science::ScienceReportTy*	R/O	
research_location	*ksp::science::ResearchLocatic*	R/O	Get the research location the experiment was performed at.
research_location_id	string	R/O	
time_required	float	R/O	
transmission_percentage	float	R/O	
transmission_size	float	R/O	
transmission_status	bool	R/O	

Methods

start_transmit

```
researchreport.start_transmit ( ) -> bool
```

ScienceExperimentType

Science experiment type

Methods

to_string

```
scienceexperimenttype.to_string ( ) -> string
```

String representation of the number

ScienceExperimentTypeConstants

Fields

Name	Type	Read-only	Description
Both	*ksp::science::ScienceExperimentTyp*	R/O	Science experiment producing both sample and data
DataType	*ksp::science::ScienceExperimentTyp*	R/O	Science experiment producing data
Sample-Type	*ksp::science::ScienceExperimentTyp*	R/O	Science experiment producing sample

Methods

from_string

```
scienceexperimenttypeconstants.from_string ( value : string ) -> Option
↪<ksp::science::ScienceExperimentType>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ScienceReportType

Type of science report

Methods

to_string

```
sciencereporttype.to_string ( ) -> string
```

String representation of the number

ScienceReportTypeConstants

Fields

Name	Type	Read-only	Description
DataType	*ksp::science::ScienceReportType*	R/O	Science data
SampleType	*ksp::science::ScienceReportType*	R/O	Science sample for experiments

Methods

from_string

```
sciencereporttypeconstants.from_string ( value : string ) -> Option
↪<ksp::science::ScienceReportType>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ScienceSituation

Situation of a science experiment

Methods

to_string

```
sciencesituation.to_string ( ) -> string
```

String representation of the number

ScienceSituationConstants

Fields

Name	Type	Read-only	Description
Atmosphere	*ksp::science::ScienceSituation*	R/O	Experiment inside an atmosphere
HighOrbit	*ksp::science::ScienceSituation*	R/O	Experiment in high orbit
Landed	*ksp::science::ScienceSituation*	R/O	Experiment while landed
LowOrbit	*ksp::science::ScienceSituation*	R/O	Experiment in low orbit
None	*ksp::science::ScienceSituation*	R/O	No specific situation required
Splashed	*ksp::science::ScienceSituation*	R/O	Experiment while splashed

Methods

from_string

```
sciencesituationconstants.from_string ( value : string ) -> Option
↪<ksp::science::ScienceSituation>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

4.17.2 Constants

Name	Type	Description
ExperimentState	ksp::science::ExperimentStateConstants	Science experiment state
ScienceExperimentType	ksp::science::ScienceExperimentTypeConstants	Science experiment type
ScienceReportType	ksp::science::ScienceReportTypeConstants	Type of science report
ScienceSituation	ksp::science::ScienceSituationConstants	Situation of a science experiment

4.17.3 Functions

get_completed_research_reports

```
pub sync fn get_completed_research_reports ( ) -> ksp::science::CompletedResearchReport[]
```

Get all completed research reports.

4.18 ksp::telemetry

4.18.1 Types

TimeSeries

Fields

Name	Type	Read-only	Description
end_ut	float	R/O	End time of the time series. This will increase when data is added.
name	string	R/O	Name of the time series. Has to be unique.
resolution	float	R/O	Current time resolution of the time series. This will increase the longer `end_ut - start_ut` gets to prevent accumulation of too much data.
start_ut	float	R/O	Start time of the time series.

Methods

add_data

```
timeseries.add_data ( ut : float,
                      value : float ) -> bool
```

Add a data `value` at time `ut`.

Parameters

Name	Type	Optional	Description
ut	float		
value	float		

4.18.2 Functions

add_time_series

```
pub sync fn add_time_series ( name : string,
                              startUt : float,
                              initialResolution : float ) -> ksp::telemetry::TimeSeries
```

Create a new time series starting at `startUt` with an initial resultion `initialResolution`. If a time series of the `name` already exists it will be replace by the new one.

Parameters

Name	Type	Optional	Description
name	string		
startUt	float		
initialResolution	float		

get_time_series

```
pub sync fn get_time_series ( name : string ) -> Option<ksp::telemetry::TimeSeries>
```

Get a time series by name. Will be undefined if there it does not exists

Parameters

Name	Type	Optional	Description
name	string		

remove_all_time_series

```
pub sync fn remove_all_time_series ( ) -> Unit
```

Remove all time series.

remove_time_series

```
pub sync fn remove_time_series ( name : string ) -> bool
```

Remove a time series by name.

Parameters

Name	Type	Optional	Description
name	string		

save_time_series

```
pub sync fn save_time_series ( filename : string ) -> Unit
```

Store the data of all time series to a file.

Parameters

Name	Type	Optional	Description
filename	string		

4.19 ksp::testing

4.19.1 Constants

Name	Type	Description
IDENTITY_COORDINATE_SYSTEM	ksp::math::TransformFrame	

4.19.2 Functions

assert_false

```
pub sync fn assert_false ( actual : bool ) -> Unit
```

Assert that `actual` is false (Test only)

Parameters

Name	Type	Optional	Description
actual	bool		

assert_float

```
pub sync fn assert_float ( expected : float,
                           actual : float,
                           delta : float ) -> Unit
```

Assert that `actual` float is almost equal to `expected` with an absolute tolerance of `delta` (Test only)

Parameters

Name	Type	Optional	Description
expected	float		
actual	float		
delta	float	x	

assert_int

```
pub sync fn assert_int ( expected : int,
                         actual : int ) -> Unit
```

Assert that `actual` integer is equal to `expected` (Test only)

Parameters

Name	Type	Optional	Description
expected	int		
actual	int		

assert_none

```
pub sync fn assert_none ( actual : Option<T> ) -> Unit
```

Parameters

Name	Type	Optional	Description
actual	Option		

assert_some

```
pub sync fn assert_some ( expected : T,
                         actual : Option<T> ) -> Unit
```

Parameters

Name	Type	Optional	Description
expected	T		
actual	Option		

assert_string

```
pub sync fn assert_string ( expected : string,
                            actual : string ) -> Unit
```

Assert that `actual` string is equal to `expected` (Test only)

Parameters

Name	Type	Optional	Description
expected	string		
actual	string		

assert_true

```
pub sync fn assert_true ( actual : bool ) -> Unit
```

Assert that `actual` is true (Test only)

Parameters

Name	Type	Optional	Description
actual	bool		

assert_vec2

```
pub sync fn assert_vec2 ( expected : ksp::math::Vec2,
                          actual : ksp::math::Vec2,
                          delta : float ) -> Unit
```

Parameters

Name	Type	Optional	Description
expected	ksp::math::Vec2		
actual	ksp::math::Vec2		
delta	float	x	

assert_vec3

```
pub sync fn assert_vec3 ( expected : ksp::math::Vec3,
                          actual : ksp::math::Vec3,
                          delta : float ) -> Unit
```

Parameters

Name	Type	Optional	Description
expected	ksp::math::Vec3		
actual	ksp::math::Vec3		
delta	float	x	

assert_yield

```
pub sync fn assert_yield ( expected : int ) -> Unit
```

Assert that test case has yielded `expected` number of times already (Async test only)

Parameters

Name	Type	Optional	Description
expected	int		

fail_test

```
pub sync fn fail_test ( message : string ) -> Unit
```

Fail the test case with a `message` (Test only).

Parameters

Name	Type	Optional	Description
message	string		

test_sleep

```
pub sync fn test_sleep ( millis : int ) -> Unit
```

Suspend execution for `millis`

Parameters

Name	Type	Optional	Description
millis	int		

yield

```
pub fn yield ( ) -> Unit
```

Yield the test case (Async test only)

4.20 ksp::ui

4.20.1 Types

Align

Alignment of the element in off direction (horizontal for vertical container and vice versa)

Methods

to_string

```
align.to_string ( ) -> string
```

String representation of the number

AlignConstants

Fields

Name	Type	Read-only	Description
Center	*ksp::ui::Align*	R/O	Align the element to the center of the container.
End	*ksp::ui::Align*	R/O	Align the element to end of container (right/bottom).
Start	*ksp::ui::Align*	R/O	Align the element to start of container (left/top).
Stretch	*ksp::ui::Align*	R/O	Stretch the element to full size of container

Methods

from_string

```
alignconstants.from_string ( value : string ) -> Option<ksp::ui::Align>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

Button

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	Enable/disable the button
font_size	float	R/W	Font size of the button label
label	string	R/W	Button label

Methods

on_click

```
button.on_click ( onClick : sync fn() -> T ) -> Unit
```

Function to be called if button is clicked

Parameters

Name	Type	Optional	Description
onClick	sync fn() -> T		

remove

```
button.remove ( ) -> Unit
```

Canvas

Fields

Name	Type	Read-only	Description
height	float	R/O	Current height of the canvas (determined by the surrounding container)
min_size	*ksp::math::Vec2*	R/W	Minimum size of the canvas.
width	float	R/O	Current width of the canvas (determined by the surrounding container)

Methods

add_line

```
canvas.add_line ( points : ksp::math::Vec2[],
                  strokeColor : ksp::console::RgbaColor,
                  closed : bool,
                  thickness : float ) -> ksp::ui::Line2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	
thickness	float	x	

add_pixel_line

```
canvas.add_pixel_line ( points : ksp::math::Vec2[],
                        strokeColor : ksp::console::RgbaColor,
                        closed : bool ) -> ksp::ui::PixelLine2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	

add_polygon

```
canvas.add_polygon ( points : ksp::math::Vec2[],
                     fillColor : ksp::console::RgbaColor ) -> ksp::ui::Polygon2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
fillColor	ksp::console::RgbaColor		

add_rect

```
canvas.add_rect ( point1 : ksp::math::Vec2,
                  point2 : ksp::math::Vec2,
                  fillColor : ksp::console::RgbaColor ) -> ksp::ui::Rect2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
fillColor	ksp::console::RgbaColor		

add_rotate

```
canvas.add_rotate ( degrees : float ) -> ksp::ui::Rotate2D
```

Parameters

Name	Type	Optional	Description
degrees	float		

add_scale

```
canvas.add_scale ( scale : ksp::math::Vec2 ) -> ksp::ui::Scale2D
```

Parameters

Name	Type	Optional	Description
scale	ksp::math::Vec2		

add_text

```
canvas.add_text ( position : ksp::math::Vec2,
                  text : string,
                  fontSize : float,
                  color : ksp::console::RgbaColor,
                  degrees : float,
                  pivot : ksp::math::Vec2 ) -> ksp::ui::Text2D
```

Parameters

Name	Type	Optional	Description
position	ksp::math::Vec2		
text	string		
fontSize	float		
color	ksp::console::RgbaColor		
degrees	float	x	
pivot	ksp::math::Vec2	x	

add_translate

```
canvas.add_translate ( translate : ksp::math::Vec2 ) -> ksp::ui::Translate2D
```

Parameters

Name	Type	Optional	Description
translate	ksp::math::Vec2		

add_value_raster

```
canvas.add_value_raster ( point1 : ksp::math::Vec2,
                          point2 : ksp::math::Vec2,
                          values : float[],
                          width : int,
                          height : int,
                          gradientWrapper : ksp::ui::Gradient ) -> ksp::ui::ValueRaster2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
values	float[]		
width	int		
height	int		
gradientWrapper	ksp::ui::Gradient		

clear

```
canvas.clear ( ) -> Unit
```

remove

```
canvas.remove ( ) -> Unit
```

ConsoleWindow

Represents the console window

Fields

Name	Type	Read-only	Description
is_closed	bool	R/O	Check if the console window is closed
min_size	*ksp::math::Vec2*	R/O	Get minimum size of window
position	*ksp::math::Vec2*	R/W	Get or change position of window
size	*ksp::math::Vec2*	R/W	Get or change size of window

Methods

center

```
consolewindow.center ( ) -> Unit
```

Center window on the screen.

close

```
consolewindow.close ( ) -> Unit
```

Close the console window

open

```
consolewindow.open ( ) -> Unit
```

Open the console window

Container

Methods

add_button

```
container.add_button ( label : string,
                       align : ksp::ui::Align,
                       stretch : float ) -> ksp::ui::Button
```

Add button to the container

Parameters

Name	Type	Optional	Description
label	string		
align	ksp::ui::Align	x	Alignment of the button in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_canvas

```
container.add_canvas ( minWidth : float,
                       minHeight : float,
                       align : ksp::ui::Align,
                       stretch : float ) -> ksp::ui::Canvas
```

Add canvas to the container

Parameters

Name	Type	Optional	Description
minWidth	float		Minimum width of the canvas
minHeight	float		Minimum height of the canvas
align	ksp::ui::Align	x	Alignment of the canvas in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_dropdown

```
container.add_dropdown ( options : string[],
                         align : ksp::ui::Align,
                         stretch : float ) -> ksp::ui::Dropdown
```

Add dropdown field to the container

Parameters

Name	Type	Optional	Description
options	string[]		Selectable options
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_float_input

```
container.add_float_input ( align : ksp::ui::Align,
                            stretch : float ) -> ksp::ui::FloatInputField
```

Add float input field to the container

Parameters

Name	Type	Optional	Description
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_horizontal

```
container.add_horizontal ( gap : float,
                           align : ksp::ui::Align,
                           stretch : float ) -> ksp::ui::Container
```

Add sub container with horizontal layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the container
align	ksp::ui::Align	x	Alignment of the sub container in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_horizontal_panel

```
container.add_horizontal_panel ( gap : float,
                                 align : ksp::ui::Align,
                                 stretch : float ) -> ksp::ui::Container
```

Add sub panel with horizontal layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the panel
align	ksp::ui::Align	x	Alignment of the panel in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_horizontal_slider

```
container.add_horizontal_slider ( min : float,
                                  max : float,
                                  align : ksp::ui::Align,
                                  stretch : float ) -> ksp::ui::Slider
```

Add horizontal slider to the container

Parameters

Name	Type	Optional	Description
min	float		Minimum value of the slider
max	float		Maximum value of the slider
align	ksp::ui::Align	x	Alignment of the slider in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_int_input

```
container.add_int_input ( align : ksp::ui::Align,
                          stretch : float ) -> ksp::ui::IntInputField
```

Add integer input field to the container

Parameters

Name	Type	Optional	Description
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_label

```
container.add_label ( label : string,
                      align : ksp::ui::Align,
                      stretch : float ) -> ksp::ui::Label
```

Add label to the container

Parameters

Name	Type	Optional	Description
label	string		
align	ksp::ui::Align	x	Alignment of the label in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_spacer

```
container.add_spacer ( size : float,
                       stretch : float ) -> Unit
```

Add empty space between elements

Parameters

Name	Type	Optional	Description
size	float		Minimum amount of space between elements
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_string_input

```
container.add_string_input ( align : ksp::ui::Align,
                             stretch : float ) -> ksp::ui::StringInputField
```

Add string input field to the container

Parameters

Name	Type	Optional	Description
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_toggle

```
container.add_toggle ( label : string,
                       align : ksp::ui::Align,
                       stretch : float ) -> ksp::ui::Toggle
```

Add toggle to the container

Parameters

Name	Type	Optional	Description
label	string		
align	ksp::ui::Align	x	Alignment of the toggle in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_vertical

```
container.add_vertical ( gap : float,
                         align : ksp::ui::Align,
                         stretch : float ) -> ksp::ui::Container
```

Add sub container with vertical layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the container
align	ksp::ui::Align	x	Alignment of the sub container in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_vertical_panel

```
container.add_vertical_panel ( gap : float,
                               align : ksp::ui::Align,
                               stretch : float ) -> ksp::ui::Container
```

Add sub panel with vertical layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the panel
align	ksp::ui::Align	x	Alignment of the panel in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_vertical_scroll

```
container.add_vertical_scroll ( minWidth : float,
                                minHeight : float,
                                gap : float,
                                align : ksp::ui::Align,
                                stretch : float ) -> ksp::ui::Container
```

Add vertical scroll view to the container

Parameters

Name	Type	Optional	Description
minWidth	float		Minimum width of the scroll view
minHeight	float		Minimum height of the scroll view
gap	float	x	Gap between each element of the panel
align	ksp::ui::Align	x	Alignment of the panel in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

remove

```
container.remove ( ) -> Unit
```

Dropdown

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
options	string[]	R/W	
value	int	R/W	

Methods

bind

```
dropdown.bind ( boundValue : Cell<T> ) -> ksp::ui::Dropdown
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		

on_change

```
dropdown.on_change ( onChange : sync fn(int) -> T ) -> Unit
```

Parameters

Name	Type	Optional	Description
onChange	sync fn(int) -> T		

remove

```
dropdown.remove ( ) -> Unit
```

FloatInputField

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
font_size	float	R/W	
value	float	R/W	

Methods

bind

```
floatinputfield.bind ( boundValue : Cell<T> ) -> ksp::ui::FloatInputField
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		

on_change

```
floatinputfield.on_change ( onChange : sync fn(float) -> T ) -> Unit
```

Parameters

Name	Type	Optional	Description
onChange	sync fn(float) -> T		

remove

```
floatinputfield.remove ( ) -> Unit
```

Gradient

Methods

add_color

```
gradient.add_color ( value : float,
                     color : ksp::console::RgbaColor ) -> bool
```

Parameters

Name	Type	Optional	Description
value	float		
color	ksp::console::RgbaColor		

IntInputField

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
font_size	float	R/W	
value	int	R/W	

Methods

bind

```
intinputfield.bind ( boundValue : Cell<T> ) -> ksp::ui::IntInputField
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		

on_change

```
intinputfield.on_change ( onChange : sync fn(float) -> T ) -> Unit
```

Parameters

Name	Type	Optional	Description
onChange	sync fn(float) -> T		

remove

```
intinputfield.remove ( ) -> Unit
```

Label

Fields

Name	Type	Read-only	Description
font_size	float	R/W	
text	string	R/W	

Methods

bind

```
label.bind ( boundValue : Cell<T>,
             format : string ) -> ksp::ui::Label
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		
format	string	x	

remove

```
label.remove ( ) -> Unit
```

Line2D

Fields

Name	Type	Read-only	Description
closed	bool	R/W	
points	*ksp::math::Vec2*[]	R/W	
stroke_color	*ksp::console::RgbaColor*	R/W	
thickness	float	R/W	

PixelLine2D

Fields

Name	Type	Read-only	Description
closed	bool	R/W	
points	*ksp::math::Vec2*[]	R/W	
stroke_color	*ksp::console::RgbaColor*	R/W	

Polygon2D

Fields

Name	Type	Read-only	Description
fill_color	*ksp::console::RgbaColor*	R/W	
points	*ksp::math::Vec2*[]	R/W	

Rect2D

Fields

Name	Type	Read-only	Description
fill_color	*ksp::console::RgbaColor*	R/W	
point1	*ksp::math::Vec2*	R/W	
point2	*ksp::math::Vec2*	R/W	

Rotate2D

Fields

Name	Type	Read-only	Description
degrees	float	R/W	
pivot	*ksp::math::Vec2*	R/W	

Methods

add_line

```
rotate2d.add_line ( points : ksp::math::Vec2[],
                    strokeColor : ksp::console::RgbaColor,
                    closed : bool,
                    thickness : float ) -> ksp::ui::Line2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	
thickness	float	x	

add_pixel_line

```
rotate2d.add_pixel_line ( points : ksp::math::Vec2[],
                          strokeColor : ksp::console::RgbaColor,
                          closed : bool ) -> ksp::ui::PixelLine2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	

add_polygon

```
rotate2d.add_polygon ( points : ksp::math::Vec2[],
                       fillColor : ksp::console::RgbaColor ) -> ksp::ui::Polygon2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
fillColor	ksp::console::RgbaColor		

add_rect

```
rotate2d.add_rect ( point1 : ksp::math::Vec2,
                    point2 : ksp::math::Vec2,
                    fillColor : ksp::console::RgbaColor ) -> ksp::ui::Rect2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
fillColor	ksp::console::RgbaColor		

add_rotate

```
rotate2d.add_rotate ( degrees : float ) -> ksp::ui::Rotate2D
```

Parameters

Name	Type	Optional	Description
degrees	float		

add_scale

```
rotate2d.add_scale ( scale : ksp::math::Vec2 ) -> ksp::ui::Scale2D
```

Parameters

Name	Type	Optional	Description
scale	ksp::math::Vec2		

add_text

```
rotate2d.add_text ( position : ksp::math::Vec2,
                    text : string,
                    fontSize : float,
                    color : ksp::console::RgbaColor,
                    degrees : float,
                    pivot : ksp::math::Vec2 ) -> ksp::ui::Text2D
```

Parameters

Name	Type	Optional	Description
position	ksp::math::Vec2		
text	string		
fontSize	float		
color	ksp::console::RgbaColor		
degrees	float	x	
pivot	ksp::math::Vec2	x	

add_translate

```
rotate2d.add_translate ( translate : ksp::math::Vec2 ) -> ksp::ui::Translate2D
```

Parameters

Name	Type	Optional	Description
translate	ksp::math::Vec2		

add_value_raster

```
rotate2d.add_value_raster ( point1 : ksp::math::Vec2,
                            point2 : ksp::math::Vec2,
                            values : float[],
                            width : int,
                            height : int,
                            gradientWrapper : ksp::ui::Gradient ) ->␣
↪ksp::ui::ValueRaster2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
values	float[]		
width	int		
height	int		
gradientWrapper	ksp::ui::Gradient		

clear

```
rotate2d.clear ( ) -> Unit
```

Scale2D

Fields

Name	Type	Read-only	Description
pivot	*ksp::math::Vec2*	R/W	
scale	*ksp::math::Vec2*	R/W	

Methods

add_line

```
scale2d.add_line ( points : ksp::math::Vec2[],
                   strokeColor : ksp::console::RgbaColor,
                   closed : bool,
                   thickness : float ) -> ksp::ui::Line2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	
thickness	float	x	

add_pixel_line

```
scale2d.add_pixel_line ( points : ksp::math::Vec2[],
                         strokeColor : ksp::console::RgbaColor,
                         closed : bool ) -> ksp::ui::PixelLine2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	

add_polygon

```
scale2d.add_polygon ( points : ksp::math::Vec2[],
                      fillColor : ksp::console::RgbaColor ) -> ksp::ui::Polygon2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
fillColor	ksp::console::RgbaColor		

add_rect

```
scale2d.add_rect ( point1 : ksp::math::Vec2,
                   point2 : ksp::math::Vec2,
                   fillColor : ksp::console::RgbaColor ) -> ksp::ui::Rect2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
fillColor	ksp::console::RgbaColor		

add_rotate

```
scale2d.add_rotate ( degrees : float ) -> ksp::ui::Rotate2D
```

Parameters

Name	Type	Optional	Description
degrees	float		

add_scale

```
scale2d.add_scale ( scale : ksp::math::Vec2 ) -> ksp::ui::Scale2D
```

Parameters

Name	Type	Optional	Description
scale	ksp::math::Vec2		

add_text

```
scale2d.add_text ( position : ksp::math::Vec2,
                   text : string,
                   fontSize : float,
                   color : ksp::console::RgbaColor,
                   degrees : float,
                   pivot : ksp::math::Vec2 ) -> ksp::ui::Text2D
```

Parameters

Name	Type	Optional	Description
position	ksp::math::Vec2		
text	string		
fontSize	float		
color	ksp::console::RgbaColor		
degrees	float	x	
pivot	ksp::math::Vec2	x	

add_translate

```
scale2d.add_translate ( translate : ksp::math::Vec2 ) -> ksp::ui::Translate2D
```

Parameters

Name	Type	Optional	Description
translate	ksp::math::Vec2		

add_value_raster

```
scale2d.add_value_raster ( point1 : ksp::math::Vec2,
                           point2 : ksp::math::Vec2,
                           values : float[],
                           width : int,
                           height : int,
                           gradientWrapper : ksp::ui::Gradient ) ->␣
↪ksp::ui::ValueRaster2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
values	float[]		
width	int		
height	int		
gradientWrapper	ksp::ui::Gradient		

clear

```
scale2d.clear ( ) -> Unit
```

Slider

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
value	float	R/W	

Methods

bind

```
slider.bind ( boundValue : Cell<T> ) -> ksp::ui::Slider
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		

on_change

```
slider.on_change ( onChange : sync fn(float) -> T ) -> Unit
```

Parameters

Name	Type	Optional	Description
onChange	sync fn(float) -> T		

remove

```
slider.remove ( ) -> Unit
```

StringInputField

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
font_size	float	R/W	
value	string	R/W	

Methods

bind

```
stringinputfield.bind ( boundValue : Cell<T> ) -> ksp::ui::StringInputField
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		

on_change

```
stringinputfield.on_change ( onChange : sync fn(string) -> T ) -> Unit
```

Parameters

Name	Type	Optional	Description
onChange	sync fn(string) -> T		

remove

```
stringinputfield.remove ( ) -> Unit
```

Text2D

Fields

Name	Type	Read-only	Description
color	*ksp::console::RgbaColor*	R/W	
degrees	float	R/W	
font_size	float	R/W	
pivot	*ksp::math::Vec2*	R/W	
position	*ksp::math::Vec2*	R/W	
text	string	R/W	

Toggle

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
font_size	float	R/W	
label	string	R/W	
value	bool	R/W	

Methods

bind

```
toggle.bind ( boundValue : Cell<T> ) -> ksp::ui::Toggle
```

Parameters

Name	Type	Optional	Description
boundValue	Cell		

on_change

```
toggle.on_change ( onChange : sync fn(bool) -> T ) -> Unit
```

Parameters

Name	Type	Optional	Description
onChange	sync fn(bool) -> T		

remove

```
toggle.remove ( ) -> Unit
```

Translate2D

Fields

Name	Type	Read-only	Description
translate	*ksp::math::Vec2*	R/W	

Methods

add_line

```
translate2d.add_line ( points : ksp::math::Vec2[],
                       strokeColor : ksp::console::RgbaColor,
                       closed : bool,
                       thickness : float ) -> ksp::ui::Line2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	
thickness	float	x	

add_pixel_line

```
translate2d.add_pixel_line ( points : ksp::math::Vec2[],
                             strokeColor : ksp::console::RgbaColor,
                             closed : bool ) -> ksp::ui::PixelLine2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
strokeColor	ksp::console::RgbaColor		
closed	bool	x	

add_polygon

```
translate2d.add_polygon ( points : ksp::math::Vec2[],
                          fillColor : ksp::console::RgbaColor ) -> ksp::ui::Polygon2D
```

Parameters

Name	Type	Optional	Description
points	ksp::math::Vec2[]		
fillColor	ksp::console::RgbaColor		

add_rect

```
translate2d.add_rect ( point1 : ksp::math::Vec2,
                       point2 : ksp::math::Vec2,
                       fillColor : ksp::console::RgbaColor ) -> ksp::ui::Rect2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
fillColor	ksp::console::RgbaColor		

add_rotate

```
translate2d.add_rotate ( degrees : float ) -> ksp::ui::Rotate2D
```

Parameters

Name	Type	Optional	Description
degrees	float		

add_scale

```
translate2d.add_scale ( scale : ksp::math::Vec2 ) -> ksp::ui::Scale2D
```

Parameters

Name	Type	Optional	Description
scale	ksp::math::Vec2		

add_text

```
translate2d.add_text ( position : ksp::math::Vec2,
                       text : string,
                       fontSize : float,
                       color : ksp::console::RgbaColor,
                       degrees : float,
                       pivot : ksp::math::Vec2 ) -> ksp::ui::Text2D
```

Parameters

Name	Type	Optional	Description
position	ksp::math::Vec2		
text	string		
fontSize	float		
color	ksp::console::RgbaColor		
degrees	float	x	
pivot	ksp::math::Vec2	x	

add_translate

```
translate2d.add_translate ( translate : ksp::math::Vec2 ) -> ksp::ui::Translate2D
```

Parameters

Name	Type	Optional	Description
translate	ksp::math::Vec2		

add_value_raster

```
translate2d.add_value_raster ( point1 : ksp::math::Vec2,
                               point2 : ksp::math::Vec2,
                               values : float[],
                               width : int,
                               height : int,
                               gradientWrapper : ksp::ui::Gradient ) ->␣
↪ksp::ui::ValueRaster2D
```

Parameters

Name	Type	Optional	Description
point1	ksp::math::Vec2		
point2	ksp::math::Vec2		
values	float[]		
width	int		
height	int		
gradientWrapper	ksp::ui::Gradient		

clear

```
translate2d.clear ( ) -> Unit
```

ValueRaster2D

Fields

Name	Type	Read-only	Description
gradient	*ksp::ui::Gradient*	R/W	
point1	*ksp::math::Vec2*	R/W	
point2	*ksp::math::Vec2*	R/W	
raster_height	int	R/O	
raster_width	int	R/O	
values	float[]	R/W	

Window

Fields

Name	Type	Read-only	Description
is_closed	bool	R/O	Check if the window has been closed (either be user or script)
min_size	*ksp::math::Vec2*	R/O	Get minimum size of window
position	*ksp::math::Vec2*	R/W	Get or change position of window
size	*ksp::math::Vec2*	R/W	Get or change size of window

Methods

add_button

```
window.add_button ( label : string,
                    align : ksp::ui::Align,
                    stretch : float ) -> ksp::ui::Button
```

Add button to the container

Parameters

Name	Type	Optional	Description
label	string		
align	ksp::ui::Align	x	Alignment of the button in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_canvas

```
window.add_canvas ( minWidth : float,
                    minHeight : float,
                    align : ksp::ui::Align,
                    stretch : float ) -> ksp::ui::Canvas
```

Add canvas to the container

Parameters

Name	Type	Optional	Description
minWidth	float		Minimum width of the canvas
minHeight	float		Minimum height of the canvas
align	ksp::ui::Align	x	Alignment of the canvas in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_dropdown

```
window.add_dropdown ( options : string[],
                      align : ksp::ui::Align,
                      stretch : float ) -> ksp::ui::Dropdown
```

Add dropdown field to the container

Parameters

Name	Type	Optional	Description
options	string[]		Selectable options
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_float_input

```
window.add_float_input ( align : ksp::ui::Align,
                         stretch : float ) -> ksp::ui::FloatInputField
```

Add float input field to the container

Parameters

Name	Type	Optional	Description
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_horizontal

```
window.add_horizontal ( gap : float,
                        align : ksp::ui::Align,
                        stretch : float ) -> ksp::ui::Container
```

Add sub container with horizontal layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the container
align	ksp::ui::Align	x	Alignment of the sub container in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_horizontal_panel

```
window.add_horizontal_panel ( gap : float,
                              align : ksp::ui::Align,
                              stretch : float ) -> ksp::ui::Container
```

Add sub panel with horizontal layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the panel
align	ksp::ui::Align	x	Alignment of the panel in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_horizontal_slider

```
window.add_horizontal_slider ( min : float,
                               max : float,
                               align : ksp::ui::Align,
                               stretch : float ) -> ksp::ui::Slider
```

Add horizontal slider to the container

Parameters

Name	Type	Optional	Description
min	float		Minimum value of the slider
max	float		Maximum value of the slider
align	ksp::ui::Align	x	Alignment of the slider in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_int_input

```
window.add_int_input ( align : ksp::ui::Align,
                       stretch : float ) -> ksp::ui::IntInputField
```

Add integer input field to the container

Parameters

Name	Type	Optional	Description
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_label

```
window.add_label ( label : string,
                   align : ksp::ui::Align,
                   stretch : float ) -> ksp::ui::Label
```

Add label to the container

Parameters

Name	Type	Optional	Description
label	string		
align	ksp::ui::Align	x	Alignment of the label in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_spacer

```
window.add_spacer ( size : float,
                    stretch : float ) -> Unit
```

Add empty space between elements

Parameters

Name	Type	Optional	Description
size	float		Minimum amount of space between elements
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_string_input

```
window.add_string_input ( align : ksp::ui::Align,
                          stretch : float ) -> ksp::ui::StringInputField
```

Add string input field to the container

Parameters

Name	Type	Optional	Description
align	ksp::ui::Align	x	Alignment of the input field in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_toggle

```
window.add_toggle ( label : string,
                    align : ksp::ui::Align,
                    stretch : float ) -> ksp::ui::Toggle
```

Add toggle to the container

Parameters

Name	Type	Optional	Description
label	string		
align	ksp::ui::Align	x	Alignment of the toggle in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_vertical

```
window.add_vertical ( gap : float,
                      align : ksp::ui::Align,
                      stretch : float ) -> ksp::ui::Container
```

Add sub container with vertical layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the container
align	ksp::ui::Align	x	Alignment of the sub container in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_vertical_panel

```
window.add_vertical_panel ( gap : float,
                            align : ksp::ui::Align,
                            stretch : float ) -> ksp::ui::Container
```

Add sub panel with vertical layout to the container

Parameters

Name	Type	Optional	Description
gap	float	x	Gap between each element of the panel
align	ksp::ui::Align	x	Alignment of the panel in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

add_vertical_scroll

```
window.add_vertical_scroll ( minWidth : float,
                             minHeight : float,
                             gap : float,
                             align : ksp::ui::Align,
                             stretch : float ) -> ksp::ui::Container
```

Add vertical scroll view to the container

Parameters

Name	Type	Optional	Description
minWidth	float		Minimum width of the scroll view
minHeight	float		Minimum height of the scroll view
gap	float	x	Gap between each element of the panel
align	ksp::ui::Align	x	Alignment of the panel in its parent container
stretch	float	x	Relative amount of available space to acquire (beyond minimal space)

center

```
window.center ( ) -> Unit
```

Center window on the screen.

close

```
window.close ( ) -> Unit
```

Close the window

compact

```
window.compact ( ) -> Unit
```

Resize window to its minimum size

4.20.2 Constants

Name	Type	Description
Align	ksp::ui::AlignConsta	Alignment of the element in off direction (horizontal for vertical container and vice versa)
CONSOLE_WINDOW	ksp::ui::ConsoleWin	Main console window

4.20.3 Functions

gradient

```
pub sync fn gradient ( start : ksp::console::RgbaColor,
                       end : ksp::console::RgbaColor ) -> ksp::ui::Gradient
```

Parameters

Name	Type	Optional	Description
start	ksp::console::RgbaColor		
end	ksp::console::RgbaColor		

open_centered_window

```
pub sync fn open_centered_window ( title : string,
                                   width : float,
                                   height : float ) -> ksp::ui::Window
```

Parameters

Name	Type	Optional	Description
title	string		
width	float		
height	float		

open_window

```
pub sync fn open_window ( title : string,
                          x : float,
                          y : float,
                          width : float,
                          height : float ) -> ksp::ui::Window
```

Parameters

Name	Type	Optional	Description
title	string		
x	float		
y	float		
width	float		
height	float		

screen_size

```
pub sync fn screen_size ( ) -> ksp::math::Vec2
```

4.21 ksp::vessel

Collection of types and functions to get information and control in-game vessels.

4.21.1 Types

ActionGroups

Fields

Name	Type	Read-only	Description
abort	bool	R/W	
brakes	bool	R/W	
custom1	bool	R/W	
custom10	bool	R/W	
custom2	bool	R/W	
custom3	bool	R/W	
custom4	bool	R/W	
custom5	bool	R/W	
custom6	bool	R/W	
custom7	bool	R/W	
custom8	bool	R/W	
custom9	bool	R/W	
gear	bool	R/W	
light	bool	R/W	
radiator_panels	bool	R/W	
rcs	bool	R/W	
sas	bool	R/W	
science	bool	R/W	
solar_panels	bool	R/W	

ActuatorMode

Actuator mode of a reaction wheel

Methods

to_string

```
actuatormode.to_string ( ) -> string
```

String representation of the number

ActuatorModeConstants

Fields

Name	Type	Read-only	Description
All	*ksp::vessel::ActuatorMode*	R/O	Always on
ManualOnly	*ksp::vessel::ActuatorMode*	R/O	Only active in manual control
SASOnly	*ksp::vessel::ActuatorMode*	R/O	Only active with SAS

Methods

from_string

```
actuatormodeconstants.from_string ( value : string ) -> Option<ksp::vessel::ActuatorMode>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

Autopilot

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
global_lock_direction	*ksp::math::GlobalDirection*	R/W	
global_target_orientation	*ksp::math::GlobalVector*	R/W	
lock_direction	*ksp::math::Direction*	R/W	
mode	*ksp::vessel::AutopilotMode*	R/W	
target_orientation	*ksp::math::Vec3*	R/W	

AutopilotMode

Vessel autopilot (SAS) mode

Methods

to_string

```
autopilotmode.to_string ( ) -> string
```

String representation of the number

AutopilotModeConstants

Fields

Name	Type	Read-only	Description
AntiTarget	*ksp::vessel::Auto*	R/O	Align the vessel to the vector pointing away from its target (if a target is set).
Antinormal	*ksp::vessel::Auto*	R/O	Align the vessel to the anti-normal vector of its orbit.
Autopilot	*ksp::vessel::Auto*	R/O	Align the vessel to the `vessel.autopilot.target_orientation` vector. (probably no difference to AutopilotMode.Navigation)
Maneuver	*ksp::vessel::Auto*	R/O	Align the vessel to the burn vector of the next maneuver node (if a maneuver node exists).
Navigation	*ksp::vessel::Auto*	R/O	Align the vessel to the `vessel.autopilot.target_orientation` vector.
Normal	*ksp::vessel::Auto*	R/O	Align the vessel to the normal vector of its orbit.
Prograde	*ksp::vessel::Auto*	R/O	Align the vessel to the prograde vector of its orbit.
RadialIn	*ksp::vessel::Auto*	R/O	Align the vessel to the radial-in vector of its orbit.
RadialOut	*ksp::vessel::Auto*	R/O	Align the vessel to the radial-out vector of its orbit.
Retrograde	*ksp::vessel::Auto*	R/O	Align the vessel to the retrograde vector of its orbit.
StabilityAssist	*ksp::vessel::Auto*	R/O	Stability assist mode. The autopilot tries to stop the rotation of the vessel.
Target	*ksp::vessel::Auto*	R/O	Align the vessel to the vector pointing to its target (if a target is set).

Methods

from_string

```
autopilotmodeconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::AutopilotMode>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

CommandControlState

Current state of a command module

Methods

to_string

```
commandcontrolstate.to_string ( ) -> string
```

String representation of the number

CommandControlStateConstants

Fields

Name	Type	Read-only	Description
Disabled	*ksp::vessel::CommandControlSt*	R/O	Command module disabled.
FullyFunctional	*ksp::vessel::CommandControlSt*	R/O	Command module is functional.
Hibernating	*ksp::vessel::CommandControlSt*	R/O	Command module is hibernating.
NoCommNetConnection	*ksp::vessel::CommandControlSt*	R/O	Command module has no comm net connection.
None	*ksp::vessel::CommandControlSt*	R/O	
NotEnoughCrew	*ksp::vessel::CommandControlSt*	R/O	Command module has not enough crew.
NotEnoughResources	*ksp::vessel::CommandControlSt*	R/O	Command module has not resource crew.

Methods

from_string

```
commandcontrolstateconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::CommandControlState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ConnectionNodeStatus

State of the comm-net connection

Methods

to_string

```
connectionnodestatus.to_string ( ) -> string
```

String representation of the number

ConnectionNodeStatusConstants

Fields

Name	Type	Read-only	Description
Connected	*ksp::vessel::ConnectionNodeStatus*	R/O	Connected
Disconnected	*ksp::vessel::ConnectionNodeStatus*	R/O	Disconnected
Invalid	*ksp::vessel::ConnectionNodeStatus*	R/O	Invalid
Pending	*ksp::vessel::ConnectionNodeStatus*	R/O	Pending

Methods

from_string

```
connectionnodestatusconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::ConnectionNodeStatus>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

DeltaVSituation

Vessel situation for delta-v calculation

Methods

to_string

```
deltavsituation.to_string ( ) -> string
```

String representation of the number

DeltaVSituationConstants

Fields

Name	Type	Read-only	Description
Altitude	*ksp::vessel::DeltaVSitua*	R/O	Calculate delta-v at the current altitude of the vessel.
SeaLevel	*ksp::vessel::DeltaVSitua*	R/O	Calculate delta-v at sea level of the current main body.
Vacuum	*ksp::vessel::DeltaVSitua*	R/O	Calculate delta-v in vacuum. (Typo is in the game, maybe will get fixed some day)

Methods

from_string

```
deltavsituationconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::DeltaVSituation>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

DeployableDeployState

Current state of a deployable part (like CargoBays)

Methods

to_string

```
deployabledeploystate.to_string ( ) -> string
```

String representation of the number

DeployableDeployStateConstants

Fields

Name	Type	Read-only	Description
Broken	*ksp::vessel::DeployableDeployState*	R/O	Part is broken
Extended	*ksp::vessel::DeployableDeployState*	R/O	Part is extended
Extending	*ksp::vessel::DeployableDeployState*	R/O	Part is currently extending
Retracted	*ksp::vessel::DeployableDeployState*	R/O	Part is retracted
Retracting	*ksp::vessel::DeployableDeployState*	R/O	Part is currently retracting

Methods

from_string

```
deployabledeploystateconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::DeployableDeployState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

DockingState

Current state of a docking node

Methods

to_string

```
dockingstate.to_string ( ) -> string
```

String representation of the number

DockingStateConstants

Fields

Name	Type	Read-only	Description
Acquire_Dockee	*ksp::vessel::DockingState*	R/O	Acquiring dockee
Acquire_Docker	*ksp::vessel::DockingState*	R/O	Acquiring docker
Disengaged	*ksp::vessel::DockingState*	R/O	Vessel is disengaged
Docked	*ksp::vessel::DockingState*	R/O	Vessel is docked
None	*ksp::vessel::DockingState*	R/O	
Ready	*ksp::vessel::DockingState*	R/O	Docking port is ready for docking

Methods

from_string

```
dockingstateconstants.from_string ( value : string ) -> Option<ksp::vessel::DockingState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

EngineDeltaV

Fields

Name	Type	Read-only	Description
engine	*ksp::vessel::ModuleEngine*	R/O	Deprecated: Use `.engine` instead
engine_module	*ksp::vessel::ModuleEngine*	R/O	Corresponding engine module of the vessel
part	*ksp::vessel::Part*	R/O	Deprecated: Use `.engine` instead
start_burn_stage	int	R/O	Number of the stage when engine is supposed to start

Methods

get_ISP

```
enginedeltav.get_ISP ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated ISP of the engine in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_global_thrust_vector

```
enginedeltav.get_global_thrust_vector ( situation : ksp::vessel::DeltaVSituation ) ->␣
↪ksp::math::GlobalVector
```

Coordinate independent estimated thrust vector of the engine in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_thrust

```
enginedeltav.get_thrust ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated thrust of the engine in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_thrust_vector

```
enginedeltav.get_thrust_vector ( situation : ksp::vessel::DeltaVSituation ) ->␣
↪ksp::math::Vec3
```

Estimated thrust vector of the engine in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

EngineMode

Fields

Name	Type	Read-only	Description
allow_restart	bool	R/O	
allow_shutdown	bool	R/O	
engine_type	*ksp::vessel::EngineType*	R/O	
max_thrust	float	R/O	
min_thrust	float	R/O	
name	string	R/O	
propellant	*ksp::resource::ResourceDefinition*	R/O	
throttle_locked	bool	R/O	

EngineType

Engine types

Methods

to_string

```
enginetype.to_string ( ) -> string
```

String representation of the number

EngineTypeConstants

Fields

Name	Type	Read-only	Description
Antimatter	*ksp::vessel::EngineType*	R/O	
Electric	*ksp::vessel::EngineType*	R/O	
Generic	*ksp::vessel::EngineType*	R/O	Generic engine type (not specified)
Helium3	*ksp::vessel::EngineType*	R/O	
MetallicHydrogen	*ksp::vessel::EngineType*	R/O	
Methalox	*ksp::vessel::EngineType*	R/O	Methan-oxigene rocket engine
MonoProp	*ksp::vessel::EngineType*	R/O	Mono-propellant engine
Nuclear	*ksp::vessel::EngineType*	R/O	Nuclear engine
NuclearSaltwater	*ksp::vessel::EngineType*	R/O	
Piston	*ksp::vessel::EngineType*	R/O	
ScramJet	*ksp::vessel::EngineType*	R/O	
SolidBooster	*ksp::vessel::EngineType*	R/O	Engine is a solid fuel booster
Turbine	*ksp::vessel::EngineType*	R/O	Turbine engine

Methods

from_string

```
enginetypeconstants.from_string ( value : string ) -> Option<ksp::vessel::EngineType>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

FlightCtrlState

Current state of the (pilots) flight controls.

Fields

Name	Type	Read-only	Description
breaks	bool	R/W	Brakes
gear_down	bool	R/W	Gear down
gear_up	bool	R/W	Gear up
kill_rot	bool	R/W	Kill rotation
main_throttle	float	R/W	Setting for the main throttle (0 - 1)
pitch	float	R/W	Setting for pitch rotation (-1 - 1)
pitch_trim	float	R/W	Current trim value for pitch
roll	float	R/W	Setting for roll rotation (-1 - 1)
roll_trim	float	R/W	Current trim value for roll
stage	bool	R/W	Stage
wheel_steer	float	R/W	Setting for wheel steering (-1 - 1, applied to Rovers)
wheel_steer_trim	float	R/W	Current trim value for wheel steering
wheel_throttle	float	R/W	Setting for wheel throttle (0 - 1, applied to Rovers)
wheel_throttle_trim	float	R/W	Current trim value for wheel throttle
x	float	R/W	Setting for x-translation (-1 - 1)
y	float	R/W	Setting for y-translation (-1 - 1)
yaw	float	R/W	Setting for yaw rotation (-1 - 1)
yaw_trim	float	R/W	Current trim value for yaw
z	float	R/W	Setting for z-translation (-1 - 1)

Maneuver

Fields

Nam	Type	Read only	Description
nodes	*ksp::vessel::M*	R/O	
trajectory	*ksp::orbit::Tr*	R/O	Get the planed trajectory of the vessel if all maneuvers are successfully executed.The list of orbit patch will always start after the first maneuvering node.I.e. if not maneuvers are planed this list will be empty.

Methods

add

```
maneuver.add ( ut : float,
               radialOut : float,
               normal : float,
               prograde : float ) -> Result<ksp::vessel::ManeuverNode>
```

Parameters

Name	Type	Optional	Description
ut	float		
radialOut	float		
normal	float		
prograde	float		

add_burn_vector

```
maneuver.add_burn_vector ( ut : float,
                           burnVector : ksp::math::Vec3 ) -> Result
↪<ksp::vessel::ManeuverNode>
```

Add a maneuver node at a given time `ut` with a given `burnVector`. Note: Contrary to `orbit.perturbed_orbit` the maneuver node calculation take the expected burn-time of the vessel into account. Especially for greater delta-v this will lead to different results.

Parameters

Name	Type	Optional	Description
ut	float		
burnVector	ksp::math::Vec3		

next_node

```
maneuver.next_node ( ) -> Result<ksp::vessel::ManeuverNode>
```

remove_all

```
maneuver.remove_all ( ) -> Unit
```

Remove all maneuver nodes

ManeuverNode

Fields

Name	Type	Read-only	Description
ETA	float	R/W	Get/set the estimated time of arrival to the maneuver
burn_duration	float	R/O	Get the estimated burn duration of the maneuver
burn_vector	*ksp::math::Vec3*	R/W	Get/set the burn vector of the maneuver in the celestial frame of the main body
expected_orbit	*ksp::orbit::OrbitPatch*	R/O	Get the expected orbit patch after the maneuver has been executed
global_burn_vec	*ksp::math::GlobalVel*	R/W	Get/set coordinate independent the burn vector of the maneuver
normal	float	R/W	Get/set the orbital normal part of the maneuver
orbit_patch	*ksp::orbit::OrbitPatch*	R/O	Get the orbit patch the maneuver should be executed on
prograde	float	R/W	Get/set the orbital prograde part of the maneuver
radial_out	float	R/W	Get/set the orbital radial part of the maneuver
time	float	R/W	Get/set the universal time the maneuver should be executed

Methods

remove

```
maneuvernode.remove ( ) -> Unit
```

Remove maneuver node from the maneuver plan

ModuleAirIntake

Fields

Name	Type	Read-only	Description
enabled	bool	R/O	Enable/disable module
flow_rate	float	R/O	Resource flow rate
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
resource	*ksp::resource::ResourceDefinition*	R/O	
resource_units	float	R/O	
toogle_intake	bool	R/W	Toggle air intake.

ModuleCommand

Fields

Name	Type	Read-only	Description
control_state	*ksp::vessel::CommandControlState*	R/O	
has_hibernation	bool	R/O	
hibernation_multiplier	float	R/O	
is_hibernating	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	

Methods

control_from_here

```
modulecommand.control_from_here ( ) -> Unit
```

ModuleControlSurface

Fields

Name	Type	Read-only	Description
angle_of_attack	float	R/O	
authority_limiter	float	R/W	
drag	float	R/O	
drag_force	*ksp::math::Vec3*	R/O	
drag_position	*ksp::math::Vec3*	R/O	
enable_pitch	bool	R/W	
enable_roll	bool	R/W	
enable_yaw	bool	R/W	
global_drag_force	*ksp::math::GlobalVector*	R/O	
global_drag_position	*ksp::math::GlobalPosition*	R/O	
global_lift_force	*ksp::math::GlobalVector*	R/O	
global_lift_position	*ksp::math::GlobalPosition*	R/O	
invert_control	bool	R/W	
lift	float	R/O	
lift_drag_ratio	float	R/O	
lift_force	*ksp::math::Vec3*	R/O	
lift_position	*ksp::math::Vec3*	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

ModuleDecoupler

Fields

Name	Type	Read-only	Description
ejection_impulse	float	R/W	
is_decoupled	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

Methods

decouple

```
moduledecoupler.decouple ( ) -> bool
```

ModuleDeployable

Fields

Name	Type	Read-only	Description
deploy_limit	float	R/W	
deploy_state	*ksp::vessel::DeployableDeployState*	R/O	
extendable	bool	R/O	
extended	bool	R/W	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
retractable	bool	R/O	

Methods

set_extended

```
moduledeployable.set_extended ( extend : bool ) -> Unit
```

Parameters

Name	Type	Optional	Description
extend	bool		

ModuleDockingNode

Fields

Name	Type	Read-only	Description
docking_state	*ksp::vessel::DockingState*	R/O	
is_deployable_docking_port	bool	R/O	
node_types	string[]	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

Methods

control_from_here

```
moduledockingnode.control_from_here ( ) -> Unit
```

ModuleDrag

Fields

Name	Type	Read-only	Description
body_lift_force	*ksp::math::Vec3*	R/O	
body_lift_position	*ksp::math::Vec3*	R/O	
drag_force	*ksp::math::Vec3*	R/O	
drag_position	*ksp::math::Vec3*	R/O	
exposed_area	float	R/O	
global_body_lift_force	*ksp::math::GlobalVector*	R/O	
global_body_lift_position	*ksp::math::GlobalPosition*	R/O	
global_drag_force	*ksp::math::GlobalVector*	R/O	
global_drag_position	*ksp::math::GlobalPosition*	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
reference_area	float	R/O	
total_area	float	R/O	

ModuleEngine

Fields

Name	Type	Read-only	Description
auto_switch_mode	bool	R/W	
current_engine_mode	*ksp::vessel::EngineMode*	R/O	Get the current engine mode
current_propellant	*ksp::resource::ResourceDefinition*	R/O	Get the propellant of the current engine mode
current_throttle	float	R/O	
current_thrust	float	R/O	
engine_modes	*ksp::vessel::EngineMode*[]	R/O	Get all engine modes
fairing	Option<*ksp::vessel::ModuleFairing*>	R/O	
gimbal	Option<*ksp::vessel::ModuleGimbal*>	R/O	
global_thrust_direction	*ksp::math::GlobalVector*	R/O	Coordinate independent direction of thrust.
has_fairing	bool	R/O	Check if engine has fairing
has_ignited	bool	R/O	Check if engine has ignited
independent_throttle	float	R/W	Current independent throttle between 0.0 - 1.0
independent_throttle_enabled	bool	R/W	Toggle independent throttle

continues on

Table 2 – continued from previous page

Name	Type	Read-only	Description
is_flameout	bool	R/O	Check if engine had a flame-out
is_gimbal	bool	R/O	Check if engine has gimbal
is_operational	bool	R/O	Check if engine is operational
is_propellant_starved	bool	R/O	
is_shutdown	bool	R/O	Check if engine is shutdown
is_staged	bool	R/O	
max_fuel_flow	float	R/O	
max_thrust_output_atm	float	R/O	
max_thrust_output_vac	float	R/O	
min_fuel_flow	float	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
propellants	*ksp::resource::ResourceDefinition*[]	R/O	Get the propellants of the different engine mod
real_isp	float	R/O	
throttle_min	float	R/O	
thrust_direction	*ksp::math::Vec3*	R/O	Direction of thrust in the celestial frame of the
thrust_limiter	float	R/W	Current thrust limit value between 0.0 - 1.0

Methods

calc_max_thrust_output_atm

```
moduleengine.calc_max_thrust_output_atm ( atmPressurekPa : float,
                                          atmTemp : float,
                                          atmDensity : float,
                                          machNumber : float ) -> float
```

Calculate maximum thrust in atmosphere given atmospheric parameters

Parameters

Name	Type	Optional	Description
atmPressurekPa	float	x	
atmTemp	float	x	
atmDensity	float	x	
machNumber	float	x	

change_mode

```
moduleengine.change_mode ( name : string ) -> bool
```

Parameters

Name	Type	Optional	Description
name	string		

ModuleFairing

Fields

Name	Type	Read-only	Description
ejection_force	float	R/W	
enabled	bool	R/W	
is_jettisoned	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

Methods

jettison

```
modulefairing.jettison ( ) -> bool
```

Jettison the fairing

ModuleGenerator

Fields

Name	Type	Read-only	Description
enabled	bool	R/W	
generator_output	float	R/O	
is_always_active	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
resource_setting	*ksp::resource::ResourceSetting*	R/O	

ModuleGimbal

Fields

Name	Type	Read-only	Description
enable_pitch	bool	R/W	
enable_roll	bool	R/W	
enable_yaw	bool	R/W	
enabled	bool	R/W	
limiter	float	R/W	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
pitch_yaw_roll	*ksp::math::Vec3*	R/W	

ModuleHeatshield

Fields

Name	Type	Read-only	Description
ablator_ratio	float	R/O	
is_ablating	bool	R/O	
is_ablator_exhausted	bool	R/O	
is_deployed	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	

ModuleLaunchClamp

Fields

Name	Type	Read-only	Description
is_released	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

Methods

release

```
modulelaunchclamp.release ( ) -> bool
```

ModuleLiftingSurface

Fields

Name	Type	Read-only	Description
angle_of_attack	float	R/O	
drag_force	*ksp::math::Vec3*	R/O	
drag_position	*ksp::math::Vec3*	R/O	
drag_scalar	float	R/O	
global_drag_force	*ksp::math::GlobalVector*	R/O	
global_drag_position	*ksp::math::GlobalPosition*	R/O	
global_lift_force	*ksp::math::GlobalVector*	R/O	
global_lift_position	*ksp::math::GlobalPosition*	R/O	
lift_drag_ratio	float	R/O	
lift_force	*ksp::math::Vec3*	R/O	
lift_position	*ksp::math::Vec3*	R/O	
lift_scalar	float	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

ModuleLight

Fields

Name	Type	Read-only	Description
blink_enabled	bool	R/W	
blink_rate	float	R/W	
has_resources_to_operate	bool	R/O	
light_color	*ksp::math::Vec3*	R/W	
light_enabled	bool	R/W	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
pitch	float	R/W	
required_resource	*ksp::resource::ResourceSetting*	R/O	
rotation	float	R/W	

ModuleParachute

Fields

Name	Type	Read-only	Description
armed	bool	R/W	
chute_safety	*ksp::vessel::ParachuteSafeStates*	R/O	
deploy_altitude	float	R/W	
deploy_mode	*ksp::vessel::ParachuteDeployMode*	R/W	
deploy_state	*ksp::vessel::ParachuteDeployState*	R/O	
min_air_pressure	float	R/W	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

Methods

cut

```
moduleparachute.cut ( ) -> bool
```

deploy

```
moduleparachute.deploy ( ) -> bool
```

repack

```
moduleparachute.repack ( ) -> bool
```

ModuleRCS

Fields

Name	Type	Read-only	Description
enable_pitch	bool	R/W	
enable_roll	bool	R/W	
enable_x	bool	R/W	
enable_y	bool	R/W	
enable_yaw	bool	R/W	
enable_z	bool	R/W	
enabled	bool	R/W	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
propellant	*ksp::resource::ResourceDefinition*	R/O	
thrust_limiter	float	R/W	

ModuleReactionWheel

Fields

Name	Type	Read-only	Description
has_resources_to_operate	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
potential_torque	*ksp::math::Vec3*	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	
toggle_torque	bool	R/W	
wheel_actuator_mode	*ksp::vessel::ActuatorMode*	R/W	
wheel_authority	float	R/W	
wheel_state	*ksp::vessel::ReactionWheelState*	R/O	

ModuleScienceExperiment

Fields

Name	Type	Read-only	Description
experiments	*ksp::science::Experiment*[]	R/O	
is_deployed	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	

ModuleSolarPanel

Fields

Name	Type	Read-only	Description
base_flow_ra	float	R/O	Base flow rate
blocking_body	Option<*ksp::orbit::B*	R/O	
efficiency_multi	float	R/O	
energy_flow	float	R/O	
max_flow	float	R/O	Maximum flow rate in current situation. Shorthand for `base_flow_rate * star_energy_scale * efficiency_multiplier`
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
resource_settir	*ksp::resource::Res*	R/O	
star_energy_	float	R/O	

ModuleTransmitter

Fields

Name	Type	Read-only	Description
active_transmission_completed	float	R/O	
active_transmission_size	float	R/O	
communication_range	float	R/O	
data_packet_size	float	R/O	
data_transmission_interval	float	R/O	
has_resources_to_operate	bool	R/O	
is_transmitting	bool	R/O	
part	*ksp::vessel::Part*	R/O	
part_name	string	R/O	
required_resources	*ksp::resource::ResourceSetting*[]	R/O	

ParachuteDeployMode

Parachute deploy mode

Methods

to_string

```
parachutedeploymode.to_string ( ) -> string
```

String representation of the number

ParachuteDeployModeConstants

Fields

Name	Type	Read-only	Description
IMMEDIATE	*ksp::vessel::ParachuteDeployl*	R/O	Parachute will deploy immediately regardless of conditions
RISKY	*ksp::vessel::ParachuteDeployl*	R/O	Parachute might deploy in risky conditions (i.e. might tear off)
SAFE	*ksp::vessel::ParachuteDeployl*	R/O	Parachute will deploy only under safe condition (i.e. will not tear of)

Methods

from_string

```
parachutedeploymodeconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::ParachuteDeployMode>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ParachuteDeployState

Parachute deploy state

Methods

to_string

```
parachutedeploystate.to_string ( ) -> string
```

String representation of the number

ParachuteDeployStateConstants

Fields

Name	Type	Read-only	Description
ARMED	*ksp::vessel::ParachuteDeployS*	R/O	Parachute is armed (i.e. will deploy in the right condition)
CUT	*ksp::vessel::ParachuteDeployS*	R/O	Parachute has been cut
DEPLOYED	*ksp::vessel::ParachuteDeployS*	R/O	Parachute is fully deployed
SEMIDEPLOYED	*ksp::vessel::ParachuteDeployS*	R/O	Parachute is partly deployed (i.e. not fully extended)
STOWED	*ksp::vessel::ParachuteDeployS*	R/O	Parachute is stowed

Methods

from_string

```
parachutedeploystateconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::ParachuteDeployState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ParachuteSafeStates

Parachute safe states

Methods

to_string

```
parachutesafestates.to_string ( ) -> string
```

String representation of the number

ParachuteSafeStatesConstants

Fields

Name	Type	Read-only	Description
NONE	*ksp::vessel::ParachuteSafeStates*	R/O	
RISKY	*ksp::vessel::ParachuteSafeStates*	R/O	Deployment of parachute is risky
SAFE	*ksp::vessel::ParachuteSafeStates*	R/O	Parachute can be safely deployed
UNSAFE	*ksp::vessel::ParachuteSafeStates*	R/O	Deployment of parachute is unsafe

Methods

from_string

```
parachutesafestatesconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::ParachuteSafeStates>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

Part

Fields

Name	Type	Read-only	Description
activation_stage	int	R/O	
air_intake	Option<*ksp::vessel::ModuleAirIntake*>	R/O	
command_module	Option<*ksp::vessel::ModuleCommand*>	R/O	
control_surface	Option<*ksp::vessel::ModuleControlSurface*>	R/O	
decouple_stage	int	R/O	
decoupler	Option<*ksp::vessel::ModuleDecoupler*>	R/O	
deployable	Option<*ksp::vessel::ModuleDeployable*>	R/O	
docking_node	Option<*ksp::vessel::ModuleDockingNode*>	R/O	
drag	Option<*ksp::vessel::ModuleDrag*>	R/O	
dry_mass	float	R/O	Dry mass of the part
engine	Option<*ksp::vessel::ModuleEngine*>	R/O	
engine_module	Option<*ksp::vessel::ModuleEngine*>	R/O	Deprecated: Use `.engine` instead
fairing	Option<*ksp::vessel::ModuleFairing*>	R/O	
generator	Option<*ksp::vessel::ModuleGenerator*>	R/O	
global_position	*ksp::math::GlobalPosition*	R/O	Get coordinate independent position of t

continu

Table 3 – continued from previous page

Name	Type	Read-only	Description
global_rotation	*ksp::math::GlobalDirection*	R/O	
green_mass	float	R/O	Green mass (Kerbals) of the part
heatshield	Option<*ksp::vessel::ModuleHeatshield*>	R/O	
id	string	R/O	Unique part id. Note: this will be differe
is_cargo_bay	bool	R/O	
is_decoupler	bool	R/O	
is_deployable	bool	R/O	
is_drag	bool	R/O	
is_engine	bool	R/O	
is_fairing	bool	R/O	
is_generator	bool	R/O	
is_heatshield	bool	R/O	
is_launch_clamp	bool	R/O	
is_lifting_surface	bool	R/O	
is_light	bool	R/O	
is_parachute	bool	R/O	
is_rcs	bool	R/O	
is_reaction_wheel	bool	R/O	
is_science_experiment	bool	R/O	
is_solar_panel	bool	R/O	
is_transmitter	bool	R/O	
launch_clamp	Option<*ksp::vessel::ModuleLaunchClamp*>	R/O	
lifting_surface	Option<*ksp::vessel::ModuleLiftingSurface*>	R/O	
light	Option<*ksp::vessel::ModuleLight*>	R/O	
max_temperature	float	R/O	Maximum temperature of the part
parachute	Option<*ksp::vessel::ModuleParachute*>	R/O	
part_category	*ksp::vessel::PartCategory*	R/O	
part_description	string	R/O	
part_name	string	R/O	
part_title	string	R/O	
position	*ksp::math::Vec3*	R/O	Get position of the part in celestial frame
rcs	Option<*ksp::vessel::ModuleRCS*>	R/O	
reaction_wheel	Option<*ksp::vessel::ModuleReactionWheel*>	R/O	
resource_mass	float	R/O	Resource mass of the part
resource_thermal_mass	float	R/O	
resources	*ksp::resource::ResourceContainer*	R/O	
science_experiment	Option<*ksp::vessel::ModuleScienceExperiment*>	R/O	
solar_panel	Option<*ksp::vessel::ModuleSolarPanel*>	R/O	
splashed	bool	R/O	Indicate if the part has splashed
temperature	float	R/O	Temperature of the part
thermal_mass	float	R/O	
total_mass	float	R/O	Total mass of the part
transmitter	Option<*ksp::vessel::ModuleTransmitter*>	R/O	
vessel	*ksp::vessel::Vessel*	R/O	
wet_mass	float	R/O	

PartCategory

Vessel part category

Methods

to_string

```
partcategory.to_string ( ) -> string
```

String representation of the number

PartCategoryConstants

Fields

Name	Type	Read-only	Description
Aero	*ksp::vessel::PartCategory*	R/O	Aerodynamics
Amenities	*ksp::vessel::PartCategory*	R/O	Amenities
ColonyEssentials	*ksp::vessel::PartCategory*	R/O	ColonyEssentials
Communication	*ksp::vessel::PartCategory*	R/O	Communication
Control	*ksp::vessel::PartCategory*	R/O	Production
Coupling	*ksp::vessel::PartCategory*	R/O	Coupling
Electrical	*ksp::vessel::PartCategory*	R/O	Electrical
Engine	*ksp::vessel::PartCategory*	R/O	Engine
Favorites	*ksp::vessel::PartCategory*	R/O	Favorites
FuelTank	*ksp::vessel::PartCategory*	R/O	FuelTank
Ground	*ksp::vessel::PartCategory*	R/O	Ground
Payload	*ksp::vessel::PartCategory*	R/O	Payload
Pods	*ksp::vessel::PartCategory*	R/O	Pods
Production	*ksp::vessel::PartCategory*	R/O	Production
Science	*ksp::vessel::PartCategory*	R/O	Science
Storage	*ksp::vessel::PartCategory*	R/O	Storage
Structural	*ksp::vessel::PartCategory*	R/O	Production
SubAssemblies	*ksp::vessel::PartCategory*	R/O	SubAssemblies
Thermal	*ksp::vessel::PartCategory*	R/O	Thermal
Utility	*ksp::vessel::PartCategory*	R/O	Utility
none	*ksp::vessel::PartCategory*	R/O	No category

Methods

from_string

```
partcategoryconstants.from_string ( value : string ) -> Option<ksp::vessel::PartCategory>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

PilotInput

Contains the flight control values as desired by the pilot.

Fields

Name	Type	Read-only	Description
main_throttle	float	R/O	Main throttle input by the pilot
override_main_throt	Option<floa	R/W	Override main throttle. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_pitch	Option<floa	R/W	Override pitch input. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_pitch_trim	Option<floa	R/W	Override pitch trim. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_roll	Option<floa	R/W	Override roll input. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_roll_trim	Option<floa	R/W	Override roll trim. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_translate_x	Option<floa	R/W	Override RCS translation x input. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_translate_y	Option<floa	R/W	Override RCS translation y input. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_translate_z	Option<floa	R/W	Override RCS translation z input. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_wheel_stee	Option<floa	R/W	Override wheel steering. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_wheel_stee	Option<floa	R/W	Override wheel steering trim. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_wheel_thro	Option<floa	R/W	Override wheel throttle. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_wheel_thro	Option<floa	R/W	Override wheel throttle trim. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_yaw	Option<floa	R/W	Override yaw input. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
override_yaw_trim	Option<floa	R/W	Override yaw trim. As long as this is defined (i.e. set to Some(value)) the pilot input will be overriden. Set to None to give control back to the pilot.
pitch	float	R/O	Pitch input by the pilot
pitch_trim	float	R/O	Pitch trim as set by the pilot
roll	float	R/O	Roll input by the pilot
roll_trim	float	R/O	Roll trim as set by the pilot
translate_x	float	R/O	RCS translate x input by the pilot
translate_y	float	R/O	RCS translate y input by the pilot
translate_z	float	R/O	RCS translate z input by the pilot
wheel_steer	float	R/O	Wheel steering input by the pilot
wheel_steer_tri	float	R/O	Wheel trim as set by the pilot
wheel_throttle	float	R/O	Wheel throttle input by the pilot
wheel_throttle_	float	R/O	Wheel throttle trim as set by the pilot
yaw	float	R/O	Yaw input by the pilot
yaw_trim	float	R/O	Yaw trim as set by the pilot

Methods

release

```
pilotinput.release ( ) -> Unit
```

resume

```
pilotinput.resume ( ) -> Unit
```

ReactionWheelState

State of a reaction wheel

Methods

to_string

```
reactionwheelstate.to_string ( ) -> string
```

String representation of the number

ReactionWheelStateConstants

Fields

Name	Type	Read-only	Description
Active	*ksp::vessel::ReactionWheelState*	R/O	Wheel active
Broken	*ksp::vessel::ReactionWheelState*	R/O	Wheel broken
Disabled	*ksp::vessel::ReactionWheelState*	R/O	Wheel disabled

Methods

from_string

```
reactionwheelstateconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::ReactionWheelState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

ScienceStorage

Represents the science storage / research inventory of a vessel.

Fields

Name	Type	Read-only	Description
active_transmitter	Option<*ksp::vessel::ModuleTransmitter*>	R/O	
is_active	bool	R/O	
research_reports	*ksp::science::ResearchReport*[]	R/O	

Methods

start_transmit_all

```
sciencestorage.start_transmit_all ( ) -> bool
```

StageDeltaV

Fields

Name	Type	Read-only	Description
active_engines	*ksp::vessel::EngineDeltaV*[]	R/O	
burn_time	float	R/O	Estimated burn time of the stage.
dry_mass	float	R/O	Dry mass of the stage.
end_mass	float	R/O	End mass of the stage.
engines	*ksp::vessel::EngineDeltaV*[]	R/O	
fuel_mass	float	R/O	Mass of the fuel in the stage.
parts	*ksp::vessel::Part*[]	R/O	
stage	int	R/O	The stage number.
start_mass	float	R/O	Start mass of the stage.

Methods

get_ISP

```
stagedeltav.get_ISP ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated ISP of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_TWR

```
stagedeltav.get_TWR ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated TWR of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_deltav

```
stagedeltav.get_deltav ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated delta-v of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

get_thrust

```
stagedeltav.get_thrust ( situation : ksp::vessel::DeltaVSituation ) -> float
```

Estimated thrust of the stage in a given `situation`

Parameters

Name	Type	Optional	Description
situation	ksp::vessel::DeltaVSituation		

Staging

Fields

Name	Type	Read-only	Description
count	int	R/O	
current	int	R/O	
ready	bool	R/O	
total_count	int	R/O	

Methods

next

```
staging.next ( ) -> bool
```

parts_in_stage

```
staging.parts_in_stage ( stage : int ) -> ksp::vessel::Part[]
```

Parameters

Name	Type	Optional	Description
stage	int		

Targetable

Fields

Name	Type	Read-only	Description
body	Option<*ksp::orbit::Body*>	R/O	Get the targeted celestial body, if target is a body.
docking_node	Option<*ksp::vessel::ModuleDockingNoa*	R/O	Get the targeted docking node, if target is a docking node.
name	string	R/O	Name of the vessel target.
orbit	*ksp::orbit::Orbit*	R/O	Orbit of the vessel target.
vessel	Option<*ksp::vessel::Vessel*>	R/O	Get the targeted vessel, if target is a vessel.

Vessel

Represents an in-game vessel, which might be a rocket, plane, rover ... or actually just a Kerbal in a spacesuite.

Fields

Name	Type	Read-only	Description
CoM	*ksp::math::Vec3*	R/O	Position of the center of mass of the vessel.
actions	*ksp::vessel::ActionGroups*	R/O	Collection of methods to trigger action group
air_intakes	*ksp::vessel::ModuleAirIntake*[]	R/O	Get a list of all air intake parts of the vessel.
altitude_scenery	float	R/O	
altitude_sealevel	float	R/O	Altitude from sea level.
altitude_terrain	float	R/O	Altitude from terrain.
angular_velocity	*ksp::math::Vec3*	R/O	Get the coordinate angular velocity in the cel
atmosphere_density	float	R/O	
autopilot	*ksp::vessel::Autopilot*	R/O	Collection of methods to interact with the SA
available_thrust	float	R/O	Returns the maximum thrust of all engines in
body_frame	*ksp::math::TransformFrame*	R/O	The body/rotating reference frame of the vess
celestial_frame	*ksp::math::TransformFrame*	R/O	The celestial/non-rotating reference frame of
command_modules	*ksp::vessel::ModuleCommand*[]	R/O	Get a list of all command module parts of the
connection_status	*ksp::vessel::ConnectionNodeStatus*	R/O	Current connection status to the comm-net.
control_frame	*ksp::math::TransformFrame*	R/O	Reference frame for the current control positi
control_status	*ksp::vessel::VesselControlState*	R/O	Current control status of the vessel.
control_surfaces	*ksp::vessel::ModuleControlSurface*[]	R/O	Get a list of all control service parts of the ve
current_control_part	Option<*ksp::vessel::Part*>	R/O	Get the part (command module) that is curren
delta_v	*ksp::vessel::VesselDeltaV*	R/O	Collection of methods to obtain delta-v inform
docking_nodes	*ksp::vessel::ModuleDockingNode*[]	R/O	Get a list of all docking node parts of the vess
dynamic_pressure_kpa	float	R/O	Current dynamic atmospheric pressure in kPa
east	*ksp::math::Vec3*	R/O	Get the horizon east vector in the celestial fra
engines	*ksp::vessel::ModuleEngine*[]	R/O	Get a list of all engine parts of the vessel.
facing	*ksp::math::Direction*	R/O	Get the current facing direction of the vessel
geo_coordinates	*ksp::orbit::GeoCoordinates*	R/O	Get the current geo-coordinate of the vessel.
global_angular_velocity	*ksp::math::GlobalAngularVelocity*	R/O	Get the coordinate system independent angul
global_center_of_mass	*ksp::math::GlobalPosition*	R/O	Coordinate independent position of the center
global_east	*ksp::math::GlobalVector*	R/O	Get the coordinate system independent horizo

Name	Type	Read-only	Description
global_facing	*ksp::math::GlobalDirection*	R/O	Get the coordinate system independent facing
global_moment_of_inertia	*ksp::math::GlobalVector*	R/O	Get the coordinate system independent mome
global_north	*ksp::math::GlobalVector*	R/O	Get the coordinate system independent horizo
global_position	*ksp::math::GlobalPosition*	R/O	Coordinate independent position of the vesse
global_up	*ksp::math::GlobalVector*	R/O	Get the coordinate system independent horizo
global_velocity	*ksp::math::GlobalVelocity*	R/O	Get the coordinate independent velocity of th
has_launched	bool	R/O	Check if the vessel has been launched.
heading	float	R/O	Heading of the vessel relative to current main
horizon_frame	*ksp::math::TransformFrame*	R/O	Reference frame for the horizon at the curren
horizontal_surface_speed	float	R/O	Horizontal surface speed relative to current m
id	string	R/O	Unique vessel id
is_active	bool	R/O	Check if the vessel is currently active.
is_controllable	bool	R/O	Check if the vessel is generally controllable, i
is_flying	bool	R/O	Check if the vessel is flyging.
launch_time	float	R/O	Universal time when vessel has been launche
mach_number	float	R/O	Current mach number
main_body	*ksp::orbit::Body*	R/O	The main body of the current SOI the vessel i
maneuver	*ksp::vessel::Maneuver*	R/O	Collection of methods to interact with the ma
mass	float	R/O	Total mass of the vessel.
name	string	R/W	The name of the vessel.
north	*ksp::math::Vec3*	R/O	Get the horizon north vector in the celestial fr
offset_ground	float	R/O	
orbit	*ksp::orbit::OrbitPatch*	R/O	Current orbit patch of the vessel.
orbital_velocity	*ksp::math::Vec3*	R/O	Orbital velocity of the vessel relative to the m
parts	*ksp::vessel::Part*[]	R/O	Get a list of all vessel parts.
pilot_input	*ksp::vessel::PilotInput*	R/O	Get the desired pilot input for this vessel.
pitch_yaw_roll	*ksp::math::Vec3*	R/O	Returns the pitch, yaw/heading and roll of the
position	*ksp::math::Vec3*	R/O	Coordinate position of the vessel in the celest
research_location	Option<*ksp::science::ResearchLocation*>	R/O	Get the current research location of the vesse
science_storage	Option<*ksp::vessel::ScienceStorage*>	R/O	Access the science storage/research inventory
situation	*ksp::vessel::VesselSituation*	R/O	Get the current situation of the vessel.
solar_panels	*ksp::vessel::ModuleSolarPanel*[]	R/O	Get a list of all solar panel parts of the vessel.
sound_speed	float	R/O	Current speed of sound.
staging	*ksp::vessel::Staging*	R/O	Collection of methods to obtain information a
static_pressure_kpa	float	R/O	Current static atmospheric pressure in kPa.
surface_velocity	*ksp::math::Vec3*	R/O	Surface velocity of the vessel relative to the n
target	Option<*ksp::vessel::Targetable*>	R/W	Get the currently selected target of the vessel.
time_since_launch	float	R/O	Number of seconds since launch.
total_torque	*ksp::math::Vec3*	R/O	Get the available torque of relative to its cont
trajectory	*ksp::orbit::Trajectory*	R/O	Get the entire trajectory of the vessel containi
up	*ksp::math::Vec3*	R/O	Get the horizon up vector in the celestial fram
vertical_speed	float	R/O	Get the vertical speed of the vessel.
vertical_surface_speed	float	R/O	Vertical surface seed relative to current main

Methods

global_heading_direction

```
vessel.global_heading_direction ( degreesFromNorth : float,
                                  pitchAboveHorizon : float,
                                  roll : float ) -> ksp::math::GlobalDirection
```

Calculate a coordinate system independent direction based on heading, pitch and roll relative to the horizon.

Parameters

Name	Type	Optional	Description
degreesFromNorth	float		
pitchAboveHorizon	float		
roll	float		

heading_direction

```
vessel.heading_direction ( degreesFromNorth : float,
                           pitchAboveHorizon : float,
                           roll : float ) -> ksp::math::Direction
```

Calculate a direction in the celestial frame of the main body based on heading, pitch and roll relative to the horizon.

Parameters

Name	Type	Optional	Description
degreesFromNorth	float		
pitchAboveHorizon	float		
roll	float		

make_active

```
vessel.make_active ( ) -> bool
```

Make this vessel the active vessel.

manage_rcs_translate

```
vessel.manage_rcs_translate ( translateProvider : sync fn(float) -> ksp::math::Vec3 ) ->␣
↪ksp::control::RCSTranslateManager
```

Parameters

Name	Type	Optional	Description
translateProvider	sync fn(float) -> ksp::math::Vec3		

manage_steering

```
vessel.manage_steering ( pitchYawRollProvider : sync fn(float) -> ksp::math::Vec3 ) ->␣
↪ksp::control::SteeringManager
```

Parameters

Name	Type	Optional	Description
pitchYawRollProvider	sync fn(float) -> ksp::math::Vec3		

manage_throttle

```
vessel.manage_throttle ( throttleProvider : sync fn(float) -> float ) ->␣
↪ksp::control::ThrottleManager
```

Parameters

Name	Type	Optional	Description
throttleProvider	sync fn(float) -> float		

manage_wheel_steering

```
vessel.manage_wheel_steering ( wheelSteeringProvider : sync fn(float) -> float ) ->␣
↪ksp::control::WheelSteeringManager
```

Parameters

Name	Type	Optional	Description
wheelSteeringProvider	sync fn(float) -> float		

manage_wheel_throttle

```
vessel.manage_wheel_throttle ( wheelThrottleProvider : sync fn(float) -> float ) ->␣
↪ksp::control::WheelThrottleManager
```

Parameters

Name	Type	Optional	Description
wheelThrottleProvider	sync fn(float) -> float		

override_input_pitch

```
vessel.override_input_pitch ( value : float ) -> Unit
```

Deprecated: Use corresponding field in `pilot_input`. One time override for the pitch control. Note: This has to be refreshed regularly to have an impact.

Parameters

Name	Type	Optional	Description
value	float		

override_input_roll

```
vessel.override_input_roll ( value : float ) -> Unit
```

Deprecated: Use corresponding field in `pilot_input`. One time override for the roll control. Note: This has to be refreshed regularly to have an impact.

Parameters

Name	Type	Optional	Description
value	float		

override_input_translate_x

```
vessel.override_input_translate_x ( value : float ) -> Unit
```

Deprecated: Use corresponding field in `pilot_input`. One time override for the translate x control. Note: This has to be refreshed regularly to have an impact.

Parameters

Name	Type	Optional	Description
value	float		

override_input_translate_y

```
vessel.override_input_translate_y ( value : float ) -> Unit
```

Deprecated: Use corresponding field in `pilot_input`. One time override for the translate y control. Note: This has to be refreshed regularly to have an impact.

Parameters

Name	Type	Optional	Description
value	float		

override_input_translate_z

```
vessel.override_input_translate_z ( value : float ) -> Unit
```

Deprecated: Use corresponding field in `pilot_input`. One time override for the translate z control. Note: This has to be refreshed regularly to have an impact.

Parameters

Name	Type	Optional	Description
value	float		

override_input_yaw

```
vessel.override_input_yaw ( value : float ) -> Unit
```

Deprecated: Use corresponding field in `pilot_input`. One time override for the yaw control. Note: This has to be refreshed regularly to have an impact.

Parameters

Name	Type	Optional	Description
value	float		

release_control

```
vessel.release_control ( ) -> Unit
```

Unhook all autopilots from the vessel.

set_rcs_translate

```
vessel.set_rcs_translate ( translate : ksp::math::Vec3 ) ->␣
↪ksp::control::RCSTranslateManager
```

Parameters

Name	Type	Optional	Description
translate	ksp::math::Vec3		

set_steering

```
vessel.set_steering ( pitchYawRoll : ksp::math::Vec3 ) -> ksp::control::SteeringManager
```

Parameters

Name	Type	Optional	Description
pitchYawRoll	ksp::math::Vec3		

set_throttle

```
vessel.set_throttle ( throttle : float ) -> ksp::control::ThrottleManager
```

Parameters

Name	Type	Optional	Description
throttle	float		

set_wheel_steering

```
vessel.set_wheel_steering ( wheelSteering : float ) -> ksp::control::WheelSteeringManager
```

Parameters

Name	Type	Optional	Description
wheelSteering	float		

set_wheel_throttle

```
vessel.set_wheel_throttle ( wheelThrottle : float ) -> ksp::control::WheelThrottleManager
```

Parameters

Name	Type	Optional	Description
wheelThrottle	float		

VesselControlState

Vessel control state

Methods

to_string

```
vesselcontrolstate.to_string ( ) -> string
```

String representation of the number

VesselControlStateConstants

Fields

Name	Type	Read-only	Description
FullControl	*ksp::vessel::VesselControlState*	R/O	Vessel can be fully controlled.
FullControlHibernation	*ksp::vessel::VesselControlState*	R/O	Vessel is in hibernation with full control.
NoCommNet	*ksp::vessel::VesselControlState*	R/O	Vessel has no connection to mission control.
NoControl	*ksp::vessel::VesselControlState*	R/O	Vessel can not be controlled.

Methods

from_string

```
vesselcontrolstateconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::VesselControlState>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

VesselDeltaV

Fields

Name	Type	Read-only	Description
stages	*ksp::vessel::StageDeltaV*[]	R/O	

Methods

stage

```
vesseldeltav.stage ( stage : int ) -> Option<ksp::vessel::StageDeltaV>
```

Get delta-v information for a specific `stage` of the vessel, if existent.

Parameters

Name	Type	Optional	Description
stage	int		

VesselSituation

Vessel situation

Methods

to_string

```
vesselsituation.to_string ( ) -> string
```

String representation of the number

VesselSituationConstants

Fields

Name	Type	Read-only	Description
Escaping	*ksp::vessel::VesselSituation*	R/O	Vessel is on an escape trajectory.
Flying	*ksp::vessel::VesselSituation*	R/O	Vessel is flying.
Landed	*ksp::vessel::VesselSituation*	R/O	Vessel has landed.
Orbiting	*ksp::vessel::VesselSituation*	R/O	Vessel is orbiting its main body.
PreLaunch	*ksp::vessel::VesselSituation*	R/O	Vessel is in pre-launch situation.
Splashed	*ksp::vessel::VesselSituation*	R/O	Vessel has splashed in water.
SubOrbital	*ksp::vessel::VesselSituation*	R/O	Vessel is on a sub-orbital trajectory.
Unknown	*ksp::vessel::VesselSituation*	R/O	Vessel situation is unknown.

Methods

from_string

```
vesselsituationconstants.from_string ( value : string ) -> Option
↪<ksp::vessel::VesselSituation>
```

Parse from string

Parameters

Name	Type	Optional	Description
value	string		Enum value to lookup

4.21.2 Constants

Name	Type	Description
ActuatorMode	ksp::vessel::ActuatorModeConstants	Actuator mode of a reaction wheel
AutopilotMode	ksp::vessel::AutopilotModeConstants	Vessel autopilot (SAS) mode
CommandControlState	ksp::vessel::CommandControlStateConstar	Current state of a command module
ConnectionNodeStatus	ksp::vessel::ConnectionNodeStatusConstar	State of the comm-net connection
DeltaVSituation	ksp::vessel::DeltaVSituationConstants	Vessel situation for delta-v calculation
DeployableDeployState	ksp::vessel::DeployableDeployStateConsta	Current state of a deployable part (like CargoBays)
DockingState	ksp::vessel::DockingStateConstants	Current state of a docking node
EngineType	ksp::vessel::EngineTypeConstants	Engine types
ParachuteDeployMode	ksp::vessel::ParachuteDeployModeConstar	Parachute deploy mode
ParachuteDeployState	ksp::vessel::ParachuteDeployStateConstant	Parachute deploy state
ParachuteSafeStates	ksp::vessel::ParachuteSafeStatesConstants	Parachute safe states
PartCategory	ksp::vessel::PartCategoryConstants	Vessel part category
ReactionWheelState	ksp::vessel::ReactionWheelStateConstants	State of a reaction wheel
VesselControlState	ksp::vessel::VesselControlStateConstants	Vessel control state
VesselSituation	ksp::vessel::VesselSituationConstants	Vessel situation

4.21.3 Functions

active_vessel

```
pub sync fn active_vessel ( ) -> Result<ksp::vessel::Vessel>
```

Try to get the currently active vessel. Will result in an error if there is none.

get_all_owned_vessels

```
pub sync fn get_all_owned_vessels ( ) -> ksp::vessel::Vessel[]
```

Get all vessels owned by the current player.

get_vessels_in_range

```
pub sync fn get_vessels_in_range ( ) -> ksp::vessel::Vessel[]
```

Get all vessels in range of the current view.

4.22 std::atmo

4.22.1 Functions

atmo_launch

```
pub fn atmo_launch ( vessel : ksp::vessel::Vessel,
                     target_apoapsis : float,
                     heading : float,
                     low_turn : float,
                     high_turn : float ) -> Result<Unit>
```

Automatically launch a rocket from an atmosphere to a circular orbit.

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target_apoapsis	float		
heading	float		
low_turn	float	x	
high_turn	float	x	

atmo_launch_ascent

```
pub fn atmo_launch_ascent ( vessel : ksp::vessel::Vessel,
                            target_apoapsis : float,
                            heading : float,
                            low_turn : float,
                            high_turn : float ) -> Unit
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target_apoapsis	float		
heading	float		
low_turn	float	x	
high_turn	float	x	

4.23 std::control::steering

4.23.1 Types

SteeringControl

Helper to use `SteeringController` with a `SteeringManager` of a vessel. Use the `control_steering` function to set it up correctly.

Fields

Name	Type	Read-only	Description
controller	*std::control::steering::SteeringController*	R/W	
manager	*ksp::control::SteeringManager*	R/W	

Methods

release

```
steeringcontrol.release ( ) -> Unit
```

resume

```
steeringcontrol.resume ( ) -> Unit
```

set_direction

```
steeringcontrol.set_direction ( dir : ksp::math::Direction ) -> Unit
```

Parameters

Name	Type	Optional	Description
dir	ksp::math::Direction		

set_global_direction

```
steeringcontrol.set_global_direction ( dir : ksp::math::GlobalDirection ) -> Unit
```

Parameters

Name	Type	Optional	Description
dir	ksp::math::GlobalDirection		

set_heading

```
steeringcontrol.set_heading ( degrees_from_north : float,
                              pitch_above_horizon : float,
                              roll : float ) -> Unit
```

Parameters

Name	Type	Optional	Description
degrees_from_north	float		
pitch_above_horizon	float		
roll	float		

SteeringController

Fields

Name	Type	Read-only	Description
acc_pitch	float	R/W	
acc_roll	float	R/W	
acc_yaw	float	R/W	
adjust_torque	*ksp::math::Vec3*	R/W	
angular_acceleration	*ksp::math::Vec3*	R/W	
center_of_mass	*ksp::math::Vec3*	R/W	
control_torque	*ksp::math::Vec3*	R/W	
current_rot	*ksp::math::Direction*	R/W	
enable_torque_adjust	bool	R/W	
max_pitch_omega	float	R/W	
max_roll_omega	float	R/W	
max_stopping_time	float	R/W	
max_yaw_omega	float	R/W	
measured_torque	*ksp::math::Vec3*	R/W	
moment_of_inertia	*ksp::math::Vec3*	R/W	

continues on next page

Table 5 – continued from previous page

Name	Type	Read-only	Description
omega	*ksp::math::Vec3*	R/W	
phi	float	R/W	
phi_pitch	float	R/W	
phi_roll	float	R/W	
phi_yaw	float	R/W	
pitch_pi	*ksp::control::TorquePI*	R/W	
pitch_rate_pi	*ksp::control::PIDLoop*	R/W	
pitch_torque_adjust	float	R/W	
pitch_torque_calc	*ksp::control::MovingAverage*	R/W	
pitch_torque_factor	float	R/W	
raw_torque	*ksp::math::Vec3*	R/W	
roll_control_angle_range	float	R/W	
roll_pi	*ksp::control::TorquePI*	R/W	
roll_rate_pi	*ksp::control::PIDLoop*	R/W	
roll_torque_adjust	float	R/W	
roll_torque_calc	*ksp::control::MovingAverage*	R/W	
roll_torque_factor	float	R/W	
target_direction	*ksp::math::GlobalDirection*	R/W	
target_rot	*ksp::math::Direction*	R/W	
tgt_pitch_omega	float	R/W	
tgt_pitch_torque	float	R/W	
tgt_roll_omega	float	R/W	
tgt_roll_torque	float	R/W	
tgt_yaw_omega	float	R/W	
tgt_yaw_torque	float	R/W	
vessel	*ksp::vessel::Vessel*	R/W	
yaw_pi	*ksp::control::TorquePI*	R/W	
yaw_rate_pi	*ksp::control::PIDLoop*	R/W	
yaw_torque_adjust	float	R/W	
yaw_torque_calc	*ksp::control::MovingAverage*	R/W	
yaw_torque_factor	float	R/W	

Methods

print_debug

```
steeringcontroller.print_debug ( ) -> Unit
```

reset_i

```
steeringcontroller.reset_i ( ) -> Unit
```

update

```
steeringcontroller.update ( delta_t : float ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
delta_t	float		

update_control

```
steeringcontroller.update_control ( ) -> ksp::math::Vec3
```

update_prediction_pi

```
steeringcontroller.update_prediction_pi ( delta_t : float ) -> Unit
```

Parameters

Name	Type	Optional	Description
delta_t	float		

update_state_vectors

```
steeringcontroller.update_state_vectors ( delta_t : float ) -> Unit
```

Parameters

Name	Type	Optional	Description
delta_t	float		

update_torque

```
steeringcontroller.update_torque ( ) -> Unit
```

4.23.2 Functions

SteeringControl

```
pub sync fn SteeringControl ( manager : ksp::control::SteeringManager,
                              controller : std::control::steering::SteeringController ) -
↪> std::control::steering::SteeringControl
```

Helper to use `SteeringController` with a `SteeringManager` of a vessel. Use the `control_steering` function to set it up correctly.

Parameters

Name	Type	Optional	Description
manager	ksp::control::SteeringManager		
controller	std::control::steering::SteeringController		

SteeringController

```
pub sync fn SteeringController ( vessel : ksp::vessel::Vessel,
                                 target_direction : ksp::math::GlobalDirection ) ->␣
↪std::control::steering::SteeringController
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target_direction	ksp::math::GlobalDirection		

control_steering

```
pub fn control_steering ( vessel : ksp::vessel::Vessel ) ->␣
↪std::control::steering::SteeringControl
```

Control the steering of a vessel. Example usage:

```
use { control_steering } from std::control::steering
...
const control = control_steering(vessel)
```

(continues on next page)

(continued from previous page)

```
// vessel is now controlled and will keep its current rotation

control.set_heading(30, 10, 5) // change the desired rotation
sleep(5) // Give the controller time to steer the vessel
...
control.release() // release control of the vessel
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

4.24 std::lambert

4.24.1 Functions

solve_lambert

```
pub sync fn solve_lambert ( r1 : ksp::math::Vec3,
                            r2 : ksp::math::Vec3,
                            tof : float,
                            mu : float,
                            clockwise : bool ) -> (iters : int, v1 : ksp::math::Vec3, v2␣
↪: ksp::math::Vec3)
```

Solve Lambert's problem, i.e. calculate the Kepler orbit to get from point `r1` to point `r2` in time `tof` (time of flight).

- `mu` is the standard gravitational parameter of the central body
- `clockwise` defines if a clockwise or counter-clockwise orbit should be calculated
- The result `v1` is the required velocity at `r1`
- The result `v2` is the required velocity at `r2`

This is based on the solver developed by Dario Izzo Details can be found here: https://arxiv.org/pdf/1403.2705.pdf Released under the GNU GENERAL PUBLIC LICENSE as part of the PyKEP library:

Parameters

Name	Type	Optional	Description
r1	ksp::math::Vec3		
r2	ksp::math::Vec3		
tof	float		
mu	float		
clockwise	bool		

4.25 std::land::landing_simulation

4.25.1 Types

BodyParameters

Fields

Name	Type	Read-only	Description
aerobraked_radius	float	R/W	
angular_velocity	*ksp::math::Vec3*	R/W	
celestial_frame	*ksp::math::TransformFrame*	R/W	
decel_radius	float	R/W	
epoch	float	R/W	
grav_paramete	float	R/W	
landing_radius	float	R/W	
lat0_lon0_at_s	*ksp::math::Vec3*	R/W	
lat90_at_start	*ksp::math::Vec3*	R/W	
lot0_lon90_at_	*ksp::math::Vec3*	R/W	
position	*ksp::math::Vec3*	R/W	
rotation_period	float	R/W	
speed_policy	[sync fn(ksp::math::Vec3, ksp::math::Vec3) -> float](/reference/sync fn(ksp/math_Vec3, ksp_math.md#vec3) -> float)	R/W	

Methods

find_freefall_end_time

```
bodyparameters.find_freefall_end_time ( orbit : ksp::orbit::Orbit,
                                        ut : float ) -> float
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
ut	float		

freefall_ended

```
bodyparameters.freefall_ended ( orbit : ksp::orbit::Orbit,
                                ut : float ) -> bool
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
ut	float		

grav_accel

```
bodyparameters.grav_accel ( pos : ksp::math::Vec3 ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
pos	ksp::math::Vec3		

reset

```
bodyparameters.reset ( body : ksp::orbit::Body ) -> Unit
```

Parameters

Name	Type	Optional	Description
body	ksp::orbit::Body		

surface_position

```
bodyparameters.surface_position ( pos : ksp::math::Vec3,
                                  UT : float ) -> (latitude : float, longitude : float)
```

Parameters

Name	Type	Optional	Description
pos	ksp::math::Vec3		
UT	float		

surface_velocity

```
bodyparameters.surface_velocity ( pos : ksp::math::Vec3,
                                  vel : ksp::math::Vec3 ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
pos	ksp::math::Vec3		
vel	ksp::math::Vec3		

total_accel

```
bodyparameters.total_accel ( pos : ksp::math::Vec3,
                             vel : ksp::math::Vec3 ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
pos	ksp::math::Vec3		
vel	ksp::math::Vec3		

ReentrySimulation

Fields

Name	Type	Read-only	Description
body	*std::land::landing_simulation::BodyParameters*	R/W	
deltav_expended	float	R/W	
dt	float	R/W	
max_thrust_accel	float	R/W	
min_dt	float	R/W	
start_dt	float	R/W	
steps	int	R/W	
t	float	R/W	
v	*ksp::math::Vec3*	R/W	
x	*ksp::math::Vec3*	R/W	

Methods

bs34_step

```
reentrysimulation.bs34_step ( ) -> Unit
```

escaping

```
reentrysimulation.escaping ( ) -> bool
```

landed

```
reentrysimulation.landed ( ) -> bool
```

limit_speed

```
reentrysimulation.limit_speed ( ) -> Unit
```

reset

```
reentrysimulation.reset ( vessel : ksp::vessel::Vessel,
                          start_ut : float ) -> std::land::landing_
↪simulation::ReentrySimulation
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
start_ut	float		

run

```
reentrysimulation.run ( ) -> (brake_time : float, end_latitude : float, end_longitude :␣
↪float, end_time : float, path : ksp::math::GlobalPosition[])
```

ReentryTrajectory

Fields

Name	Type	Read-only	Description
brake_time	float	R/W	
end_latitude	float	R/W	
end_longitude	float	R/W	
end_time	float	R/W	
path	*ksp::math::GlobalPosition*[]	R/W	

4.25.2 Functions

BodyParameters

```
pub sync fn BodyParameters ( body : ksp::orbit::Body,
                             decel_end_altitude_asl : float,
                             landing_altitude_asl : float,
                             speed_policy : sync fn(ksp::math::Vec3, ksp::math::Vec3) ->␣
↪float ) -> std::land::landing_simulation::BodyParameters
```

Parameters

Name	Type	Optional	Description
body	ksp::orbit::Body		
decel_end_altitude_asl	float		
landing_altitude_asl	float		
speed_policy	sync fn(ksp::math::Vec3, ksp::math::Vec3) -> float		

ReentrySimulation

```
pub sync fn ReentrySimulation ( body : std::land::landing_simulation::BodyParameters,
                                start_dt : float,
                                min_dt : float,
                                max_thrust_accel : float ) -> std::land::landing_
↪simulation::ReentrySimulation
```

Parameters

Name	Type	Optional	Description
body	std::land::landing_simulation::BodyParameters		
start_dt	float		
min_dt	float		
max_thrust_accel	float		

init_simulation

```
pub sync fn init_simulation ( vessel : ksp::vessel::Vessel,
                              start_ut : float,
                              start_dt : float,
                              min_dt : float,
                              max_thrust_accel : float,
                              landing_altitude_asl : float,
                              speed_policy : sync fn(ksp::math::Vec3, ksp::math::Vec3) ->
↪ float ) -> std::land::landing_simulation::ReentrySimulation
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
start_ut	float		
start_dt	float		
min_dt	float		
max_thrust_accel	float		
landing_altitude_asl	float		
speed_policy	sync fn(ksp::math::Vec3, ksp::math::Vec3) -> float		

4.26 std::land::lib

4.26.1 Functions

land_deorbit_delta_v

```
pub sync fn land_deorbit_delta_v ( vessel : ksp::vessel::Vessel,
                                   alt : float ) -> float
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
alt	float		

land_time_to_longitude

```
pub sync fn land_time_to_longitude ( vessel : ksp::vessel::Vessel,
                                     longitude : float ) -> float
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
longitude	float		

4.27 std::land::speed_policy

4.27.1 Types

SpeedPolicy

4.27.2 Functions

gravity_turn_speed_policy

```
pub sync fn gravity_turn_speed_policy ( terrain_radius : float,
                                        g : float,
                                        thrust : float ) -> sync fn(ksp::math::Vec3,␣
↪ksp::math::Vec3) -> float
```

Parameters

Name	Type	Optional	Description
terrain_radius	float		
g	float		
thrust	float		

powered_coast_speed_policy

```
pub sync fn powered_coast_speed_policy ( terrain_radius : float,
                                         g : float,
                                         thrust : float ) -> sync fn(ksp::math::Vec3,␣
↪ksp::math::Vec3) -> float
```

Parameters

Name	Type	Optional	Description
terrain_radius	float		
g	float		
thrust	float		

safe_speed_policy

```
pub sync fn safe_speed_policy ( terrain_radius : float,
                                g : float,
                                thrust : float ) -> sync fn(ksp::math::Vec3,␣
↪ksp::math::Vec3) -> float
```

Parameters

Name	Type	Optional	Description
terrain_radius	float		
g	float		
thrust	float		

4.28 std::land::vac

4.28.1 Functions

vac_break_zero

```
pub fn vac_break_zero ( vessel : ksp::vessel::Vessel ) -> Unit
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

vac_deceleration_burn

```
pub fn vac_deceleration_burn ( vessel : ksp::vessel::Vessel,
                               simulation : std::land::landing_
↪simulation::ReentrySimulation,
                               initial_result : (brake_time : float, end_latitude :␣
↪float, end_longitude : float, end_time : float, path : ksp::math::GlobalPosition[]),
                               landing_site : ksp::orbit::GeoCoordinates,
                               speed_policy : sync fn(ksp::math::Vec3, ksp::math::Vec3) -
↪> float ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
simulation	std::land::landing_simulation::ReentrySimulation		
initial_result	(brake_time : float, end_latitude : float, end_longitude : float, end_time : float, path : ksp::math::GlobalPosition[])		
landing_site	ksp::orbit::GeoCoordinates		
speed_poli	sync fn(ksp::math::Vec3, ksp::math::Vec3) -> float		

vac_land

```
pub fn vac_land ( vessel : ksp::vessel::Vessel,
                  landing_side : ksp::orbit::GeoCoordinates,
                  land_stage : int ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
landing_side	ksp::orbit::GeoCoordinates		
land_stage	int		

vac_land_course_correct

```
pub fn vac_land_course_correct ( vessel : ksp::vessel::Vessel,
                                 simulation : std::land::landing_
↪simulation::ReentrySimulation,
                                 initial_result : (brake_time : float, end_latitude :␣
↪float, end_longitude : float, end_time : float, path : ksp::math::GlobalPosition[]),
                                 landing_site : ksp::orbit::GeoCoordinates ) -> Result
↪<(brake_time : float, end_latitude : float, end_longitude : float, end_time : float,␣
↪path : ksp::math::GlobalPosition[])>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
simulation	std::land::landing_simulation::ReentrySimulation		
initial_result	(brake_time : float, end_latitude : float, end_longitude : float, end_time : float, path : ksp::math::GlobalPosition[])		
landing_site	ksp::orbit::GeoCoordinates		

vac_land_prepare_deorbit

```
pub fn vac_land_prepare_deorbit ( vessel : ksp::vessel::Vessel,
                                  landing_site : ksp::orbit::GeoCoordinates ) -> Result
↪<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
landing_site	ksp::orbit::GeoCoordinates		

vac_touchdown

```
pub fn vac_touchdown ( vessel : ksp::vessel::Vessel ) -> Unit
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

4.29 std::maneuvers

4.29.1 Functions

bi_impulsive_transfer

```
pub sync fn bi_impulsive_transfer ( start : ksp::orbit::Orbit,
                                    target : ksp::orbit::Orbit,
                                    min_UT : float,
                                    max_UT : float ) -> Result<(TT : float, UT : float,␣
↪delta_v : ksp::math::Vec3)>
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		
min_UT	float		
max_UT	float	x	

bi_impulsive_transfer_body

```
pub sync fn bi_impulsive_transfer_body ( start : ksp::orbit::Orbit,
                                         target : ksp::orbit::Body,
                                         min_UT : float,
                                         target_periapsis : float ) -> Result<(TT :␣
↪float, UT : float, delta_v : ksp::math::Vec3)>
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Body		
min_UT	float		
target_periapsis	float		

bi_impulsive_transfer_near

```
pub sync fn bi_impulsive_transfer_near ( start : ksp::orbit::Orbit,
                                         target : ksp::orbit::Orbit,
                                         UT : float,
                                         TT : float ) -> Result<(TT : float, UT : float,␣
↪delta_v : ksp::math::Vec3)>
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		
UT	float		
TT	float		

change_apoapsis

```
pub sync fn change_apoapsis ( orbit : ksp::orbit::Orbit,
                              UT : float,
                              apoapsis_radius : float ) -> Result<ksp::math::Vec3>
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
UT	float		
apoapsis_radius	float		

change_periapsis

```
pub sync fn change_periapsis ( orbit : ksp::orbit::Orbit,
                               UT : float,
                               periapsis_radius : float ) -> Result<ksp::math::Vec3>
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
UT	float		
periapsis_radius	float		

cheapest_course_correction

```
pub sync fn cheapest_course_correction ( orbit : ksp::orbit::Orbit,
                                         min_UT : float,
                                         target : ksp::orbit::Orbit ) -> (UT : float,␣
↪delta_v : ksp::math::Vec3)
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
min_UT	float		
target	ksp::orbit::Orbit		

circularize_orbit

```
pub sync fn circularize_orbit ( orbit : ksp::orbit::Orbit ) -> Result<(UT : float, delta_
↪v : ksp::math::Vec3)>
```

Calculate the required delta-v and time to change the given `orbit` to a (mostly) circular orbit at the next apoapsis (if `orbit` is elliptic) or periapsis (if `orbit` is hyperbolic).

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		

circularize_orbit_at

```
pub sync fn circularize_orbit_at ( orbit : ksp::orbit::Orbit,
                                   UT : float ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
UT	float		

circularize_orbit_pe

```
pub sync fn circularize_orbit_pe ( orbit : ksp::orbit::Orbit ) -> Result<(UT : float,␣
↪delta_v : ksp::math::Vec3)>
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		

course_correction_body

```
pub sync fn course_correction_body ( start : ksp::orbit::Orbit,
                                     target : ksp::orbit::Body,
                                     UT : float,
                                     target_periapsis : float ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Body		
UT	float		
target_periapsis	float		

ellipticize

```
pub sync fn ellipticize ( orbit : ksp::orbit::Orbit,
                          UT : float,
                          periapsis : float,
                          apoapsis : float ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
orbit	ksp::orbit::Orbit		
UT	float		
periapsis	float		
apoapsis	float		

ideal_ejection

```
pub sync fn ideal_ejection ( body : ksp::orbit::Body,
                             UT : float,
                             radius : float,
                             normal : ksp::math::Vec3,
                             exit_velocity : ksp::math::Vec3 ) -> ksp::orbit::Orbit
```

Parameters

Name	Type	Optional	Description
body	ksp::orbit::Body		
UT	float		
radius	float		
normal	ksp::math::Vec3		
exit_velocity	ksp::math::Vec3		

intercept_at

```
pub sync fn intercept_at ( start : ksp::orbit::Orbit,
                           start_UT : float,
                           target : ksp::orbit::Orbit,
                           target_UT : float,
                           offset_distance : float ) -> (start_velocity :␣
↪ksp::math::Vec3, target_velocity : ksp::math::Vec3)
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
start_UT	float		
target	ksp::orbit::Orbit		
target_UT	float		
offset_distance	float	x	

match_apoapsis

```
pub sync fn match_apoapsis ( start : ksp::orbit::Orbit,
                             target : ksp::orbit::Orbit ) -> Result<(UT : float, delta_v␣
↪: ksp::math::Vec3)>
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		

match_inclination

```
pub sync fn match_inclination ( start : ksp::orbit::Orbit,
                                target : ksp::orbit::Orbit ) -> (UT : float, delta_v :␣
↪ksp::math::Vec3)
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		

match_periapsis

```
pub sync fn match_periapsis ( start : ksp::orbit::Orbit,
                              target : ksp::orbit::Orbit ) -> (UT : float, delta_v :␣
↪ksp::math::Vec3)
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		

match_velocities

```
pub sync fn match_velocities ( start : ksp::orbit::Orbit,
                               target : ksp::orbit::Orbit ) -> (UT : float, delta_v :␣
↪ksp::math::Vec3)
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		

next_closest_approach_time

```
pub sync fn next_closest_approach_time ( start : ksp::orbit::Orbit,
                                         target : ksp::orbit::Orbit,
                                         UT : float ) -> float
```

Parameters

Name	Type	Optional	Description
start	ksp::orbit::Orbit		
target	ksp::orbit::Orbit		
UT	float		

4.30 std::navball

4.30.1 Functions

global_navball

```
pub sync fn global_navball ( vessel : ksp::vessel::Vessel,
                             dir : ksp::math::GlobalDirection ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
dir	ksp::math::GlobalDirection		

navball

```
pub sync fn navball ( vessel : ksp::vessel::Vessel,
                      dir : ksp::math::Direction ) -> ksp::math::Vec3
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
dir	ksp::math::Direction		

4.31 std::numerics::amoeba_optimize

4.31.1 Functions

amoeba_optimize

```
pub sync fn amoeba_optimize ( func : sync fn(float, float) -> float,
                              start_points : ksp::math::Vec2[],
                              tolerance : float,
                              max_iters : int ) -> Result<(iters : int, x : float, y :␣
↪float)>
```

Parameters

Name	Type	Optional	Description
func	sync fn(float, float) -> float		
start_points	ksp::math::Vec2[]		
tolerance	float		
max_iters	int		

amoeba_optimize_perturbation

```
pub sync fn amoeba_optimize_perturbation ( func : sync fn(float, float) -> float,
                                           guess : ksp::math::Vec2,
                                           perturbation : ksp::math::Vec2,
                                           tolerance : float,
                                           max_iters : int ) -> Result<(iters : int, x :␣
↪float, y : float)>
```

Parameters

Name	Type	Optional	Description
func	sync fn(float, float) -> float		
guess	ksp::math::Vec2		
perturbation	ksp::math::Vec2		
tolerance	float		
max_iters	int		

4.32 std::numerics::anneal_optimize

4.32.1 Functions

anneal_optimize

```
pub sync fn anneal_optimize ( func : sync fn(float, float) -> float,
                              min : ksp::math::Vec2,
                              max : ksp::math::Vec2,
                              max_temp : float,
                              iters : int,
                              num_particles : int,
                              cooling_rate : float ) -> (best : (f : float, x : float, y␣
↪: float), points : (f : float, x : float, y : float)[])
```

Parameters

Name	Type	Optional	Description
func	sync fn(float, float) -> float		
min	ksp::math::Vec2		
max	ksp::math::Vec2		
max_temp	float		
iters	int	x	
num_particles	int	x	
cooling_rate	float	x	

4.33 std::numerics::bessel

4.33.1 Functions

J0

```
pub sync fn J0 ( x : float ) -> float
```

Parameters

Name	Type	Optional	Description
x	float		

J1

```
pub sync fn J1 ( x : float ) -> float
```

Parameters

Name	Type	Optional	Description
x	float		

4.34 std::numerics::brent_optimize

4.34.1 Functions

brent_optimize

```
pub sync fn brent_optimize ( func : sync fn(float) -> float,
                             start_a : float,
                             start_b : float,
                             tolerance : float,
                             max_iterations : int ) -> Result<(fx : float, x : float)>
```

Parameters

Name	Type	Optional	Description
func	sync fn(float) -> float		
start_a	float		
start_b	float		
tolerance	float		
max_iterations	int		

4.35 std::numerics::regula_falsi_solve

4.35.1 Functions

regula_falsi_solve

```
pub sync fn regula_falsi_solve ( func : sync fn(float) -> float,
                                 start_a : float,
                                 start_b : float,
```

(continues on next page)

(continued from previous page)

```
                                tolerance : float,
                                max_iterations : int ) -> float
```

Parameters

Name	Type	Optional	Description
func	sync fn(float) -> float		
start_a	float		
start_b	float		
tolerance	float		
max_iterations	int		

4.36 std::numerics::runge_kutta

4.36.1 Functions

rk23

```
pub sync fn rk23 ( accel : sync fn(float, ksp::math::Vec3, ksp::math::Vec3) ->␣
↪ksp::math::Vec3,
                   end_condition : sync fn(float, ksp::math::Vec3, ksp::math::Vec3) ->␣
↪bool,
                   start_t : float,
                   start_position : ksp::math::Vec3,
                   start_velocity : ksp::math::Vec3,
                   min_dt : float,
                   max_dt : float ) -> (position : ksp::math::Vec3, t : float, velocity␣
↪: ksp::math::Vec3)[]
```

Parameters

Name	Type	Optional	Description
accel	sync fn(float, ksp::math::Vec3, ksp::math::Vec3) -> ksp::math::Vec3		
end_condition	sync fn(float, ksp::math::Vec3, ksp::math::Vec3) -> bool		
start_t	float		
start_position	ksp::math::Vec3		
start_velocity	ksp::math::Vec3		
min_dt	float		
max_dt	float		

4.37 std::rendezvous::dock

4.37.1 Functions

choose_docking_nodes

```
pub sync fn choose_docking_nodes ( vessel : ksp::vessel::Vessel,
                                   target : ksp::vessel::Targetable ) -> Result<(target_
↪port : ksp::vessel::ModuleDockingNode, vessel_port : ksp::vessel::ModuleDockingNode)>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target	ksp::vessel::Targetable		

dock_approach

```
pub fn dock_approach ( vessel : ksp::vessel::Vessel,
                       target_port : ksp::vessel::ModuleDockingNode ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target_port	ksp::vessel::ModuleDockingNode		

dock_move_correct_side

```
pub fn dock_move_correct_side ( vessel : ksp::vessel::Vessel,
                                target_port : ksp::vessel::ModuleDockingNode ) -> Result
↪<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target_port	ksp::vessel::ModuleDockingNode		

dock_vessel

```
pub fn dock_vessel ( vessel : ksp::vessel::Vessel,
                     target : ksp::vessel::Targetable ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target	ksp::vessel::Targetable		

4.38 std::rendezvous::lib

4.38.1 Functions

intercept_target

```
pub fn intercept_target ( vessel : ksp::vessel::Vessel,
                          target : ksp::vessel::Targetable ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target	ksp::vessel::Targetable		

move_to_target

```
pub fn move_to_target ( vessel : ksp::vessel::Vessel,
                        target : ksp::vessel::Targetable ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target	ksp::vessel::Targetable		

rendezvous_with

```
pub fn rendezvous_with ( vessel : ksp::vessel::Vessel,
                         target : ksp::vessel::Targetable ) -> Result<Unit>
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
target	ksp::vessel::Targetable		

4.39 std::staging

Collection of helper functions to control staging of a vessel

4.39.1 Functions

has_engine_in_stage

```
pub sync fn has_engine_in_stage ( vessel : ksp::vessel::Vessel,
                                  stage : int ) -> bool
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
stage	int		

has_flameout

```
pub sync fn has_flameout ( vessel : ksp::vessel::Vessel ) -> bool
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

has_ignited

```
pub sync fn has_ignited ( vessel : ksp::vessel::Vessel ) -> bool
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

has_still_running

```
pub sync fn has_still_running ( vessel : ksp::vessel::Vessel ) -> bool
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

trigger_staging

```
pub fn trigger_staging ( vessel : ksp::vessel::Vessel ) -> bool
```

Helper function to automatically trigger staging during a burn.

This function is just checking if one of the ignited engines has has a flameout, which in most cases means that the current stage has burned out.

Will return `true` if stating has been triggered.

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

4.40 std::utils

Collection of helper functions not fitting anywhere else

4.40.1 Functions

angle_to_180

```
pub sync fn angle_to_180 ( angle : float ) -> float
```

Parameters

Name	Type	Optional	Description
angle	float		

angle_to_360

```
pub sync fn angle_to_360 ( angle : float ) -> float
```

Parameters

Name	Type	Optional	Description
angle	float		

global_ship_is_facing

```
pub sync fn global_ship_is_facing ( vessel : ksp::vessel::Vessel,
                                    desired_facing : ksp::math::GlobalVector,
                                    max_deviation_degrees : float,
                                    max_angular_velocity : float ) -> bool
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
desired_facing	ksp::math::GlobalVector		
max_deviation_degrees	float		
max_angular_velocity	float		

remove_all_nodes

```
pub sync fn remove_all_nodes ( vessel : ksp::vessel::Vessel ) -> Unit
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

ship_is_facing

```
pub sync fn ship_is_facing ( vessel : ksp::vessel::Vessel,
                             desired_facing : ksp::math::Vec3,
                             max_deviation_degrees : float,
                             max_angular_velocity : float ) -> bool
```

Determine if `vessel` is facing a given direction.

- `max_deviation_degrees` sets a limit how many degrees the angle may differ
- `max_angular_velocity` sets a limit how much the `vessel` may still be turning

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
desired_facing	ksp::math::Vec3		
max_deviation_degrees	float		
max_angular_velocity	float		

4.41 std::vac

4.41.1 Functions

estimate_burn_time

```
pub sync fn estimate_burn_time ( vessel : ksp::vessel::Vessel,
                                 delta_v : float,
                                 stage_delay : float,
                                 throttle_limit : float ) -> (burn_time : float, half_
↪burn_time : float)
```

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		
delta_v	float		
stage_delay	float		
throttle_limit	float		

exec_next_node

```
pub fn exec_next_node ( vessel : ksp::vessel::Vessel ) -> Result<Unit>
```

Execute the next planed maneuver node.

Will result in an error if there are no planed maneuver nodes.

Parameters

Name	Type	Optional	Description
vessel	ksp::vessel::Vessel		

CHAPTER

FIVE

BENCHMARKS

Summary: TO2 is not as fast as it could be.

But lets see the results first. The Benchmark is comparing a solver for *Lambert's problem* written in `to2` and C#. The two implementations can be found here:

- C#: LambertIzzoSolver.cs
- TO2: lambert.to2

Results for the current implementation

Method	testSet	Mean	Error	StdDev
LamberCSharp	new (...)) }, [351]	172.5 ns	1.54 ns	1.36 ns
LamberTO2	new (...)) }, [351]	308.1 ns	6.11 ns	6.00 ns
LamberCSharp	new (...)) }, [353]	177.7 ns	0.23 ns	0.21 ns
LamberTO2	new (...)) }, [353]	313.1 ns	0.90 ns	0.79 ns
LamberCSharp	new (...)) }, [350]	145.4 ns	0.68 ns	0.60 ns
LamberTO2	new (...)) }, [350]	294.8 ns	0.90 ns	0.80 ns
LamberCSharp	new (...)) }, [352]	172.9 ns	0.36 ns	0.32 ns
LamberTO2	new (...)) }, [352]	309.7 ns	5.25 ns	4.65 ns
LamberCSharp	new (...)) }, [351]	146.1 ns	0.27 ns	0.25 ns
LamberTO2	new (...)) }, [351]	292.6 ns	4.03 ns	3.77 ns
LamberCSharp	new (...)) }, [347]	152.9 ns	0.31 ns	0.26 ns
LamberTO2	new (...)) }, [347]	292.9 ns	4.59 ns	4.07 ns
LamberCSharp	new (...)) }, [347]	153.1 ns	0.40 ns	0.38 ns
LamberTO2	new (...)) }, [347]	297.8 ns	0.94 ns	0.88 ns
LamberCSharp	new (...)) }, [349]	175.2 ns	0.20 ns	0.17 ns
LamberTO2	new (...)) }, [349]	303.1 ns	0.83 ns	0.78 ns
LamberCSharp	new (...)) }, [351]	169.9 ns	0.48 ns	0.45 ns
LamberTO2	new (...)) }, [351]	306.2 ns	4.58 ns	4.28 ns
LamberCSharp	new (...)) }, [351]	152.1 ns	0.25 ns	0.23 ns
LamberTO2	new (...)) }, [351]	284.8 ns	2.09 ns	1.96 ns

So the TO2 implementation is about 1.5 - 2 times slower than the C# implementation, but ...

... the TO2 also bakes in several checks to prevent a poorly written script to crash the game:

- Loop timeout: Since TO2 scripts are running in the main game loop/thread an endless loop in a `sync` function could freeze up the game completely.
  - To prevent that every loop in a sync function contains an implicit timeout check, which is currently set at 100ms. So there might be a small hick-up, but now a freeze.

  - This does not affect loops in async functions (which is the default) or if the function is running in a background thread (see `core::background`)
- Stack overflow prevention: Creating to a stack overflow in the main game loop/thread will crash the game
  - To prevent this every function call keeps track of the call-stack an fails early before a hard stack overflow can occur.

All these security measures eat up runtime. If they are removed (aka release the breaks) the same benchmark looks like this:

Method	testSet	Mean	Error	StdDev	Median
LamberCSharp	new (...)) }, [351]	171.0 ns	2.85 ns	2.66 ns	172.2 ns
LamberTO2	new (...)) }, [351]	166.9 ns	1.09 ns	0.97 ns	166.5 ns
LamberCSharp	new (...)) }, [353]	172.0 ns	3.34 ns	3.71 ns	175.1 ns
LamberTO2	new (...)) }, [353]	172.7 ns	0.25 ns	0.23 ns	172.7 ns
LamberCSharp	new (...)) }, [350]	146.3 ns	0.22 ns	0.20 ns	146.3 ns
LamberTO2	new (...)) }, [350]	156.6 ns	3.11 ns	2.91 ns	157.0 ns
LamberCSharp	new (...)) }, [352]	173.5 ns	0.24 ns	0.23 ns	173.5 ns
LamberTO2	new (...)) }, [352]	173.7 ns	2.39 ns	2.24 ns	174.9 ns
LamberCSharp	new (...)) }, [351]	151.0 ns	2.95 ns	3.15 ns	153.0 ns
LamberTO2	new (...)) }, [351]	159.7 ns	0.42 ns	0.39 ns	159.6 ns
LamberCSharp	new (...)) }, [347]	153.1 ns	0.22 ns	0.20 ns	153.1 ns
LamberTO2	new (...)) }, [347]	158.4 ns	1.84 ns	1.72 ns	159.1 ns
LamberCSharp	new (...)) }, [347]	153.0 ns	0.18 ns	0.17 ns	153.0 ns
LamberTO2	new (...)) }, [347]	154.9 ns	1.90 ns	1.77 ns	153.9 ns
LamberCSharp	new (...)) }, [349]	173.0 ns	0.39 ns	0.35 ns	173.0 ns
LamberTO2	new (...)) }, [349]	177.1 ns	0.17 ns	0.15 ns	177.1 ns
LamberCSharp	new (...)) }, [351]	168.1 ns	3.24 ns	3.33 ns	169.9 ns
LamberTO2	new (...)) }, [351]	170.9 ns	3.30 ns	3.08 ns	173.0 ns
LamberCSharp	new (...)) }, [351]	146.9 ns	0.33 ns	0.26 ns	146.9 ns
LamberTO2	new (...)) }, [351]	155.8 ns	3.15 ns	3.63 ns	153.5 ns

... i.e. both implementations are pretty much on par

CHAPTER

SIX

CONTRIBUTING

6.1 Building

6.1.1 Common

- A copy of the game is required (obviously)
- BepInEx and SpaceWarp v0.4.0 should be installed
- By default the game is expected to be installed in `C:\Program Files (x86)\Steam\steamapps\common\Kerbal Space Program 2` if it is somewhere else you have to set a `KSP2_BASE_DIR` enviroment variable or provide it as a global property to msbuild

6.1.2 Windows

- Install Visual Studio 2022 Community edition (or higher)
  - Add Commandline Tools and support for `.NET 8.0` target

In the 'VS` commmand line or powershell

```
dotnet build  -restore -Property:Configuration=Release
```

should build everything to the `dist` folder

There are powershell helper script `build.ps1` and `clean.ps1` to do this with slightly better control.

6.1.3 Linux

Note: Running the game on linux is officially not supported.

If you have installed the game via steam/proton you should set an environment variable:

```
export KSP2_BASE_DIR=$HOME/.local/share/Steam/steamapps/common/Kerbal\ Space\ Program\ 2
```

- Install dotnet-sdk and dotnet-host for version 8.0
  - Package names will vary on distribution. For ArchLinux based distributions this would be `dotnet-sdk` and `dotnet-host`

```
msbuild -t:build,test -restore -Property:Configuration=Release
```

should build everything to the `dist` folder

There are some helper scripts `build.sh` and `clean.sh` to do this with slightly better control.

6.1.4 IDE

Depending on your IDE you might have to set a global msbuild property `KSP2_BASE_DIR`. Otherwise, I had no problem importing the solution to VS Code 2022 or Rider.

6.2 Adding or extending builtin modules

6.2.1 Standard binding

A builtin module usually looks like this:

```
[KSModule("ksp::my", Description = "Description of the module"
)]
public class MyModule {
    [KSFunction(Description = "Description of the function")]
    public static double SomeCalculation(double param1, double param2) {
        return param1 + param2;
    }

    [KSFunction(Description = "Description of the function")]
    public static double CreateMyType(long param1, string param2) {
        return new MyType(param1, param2);
    }


    [KSClass("MyType", Description = "Description of the type")
    public class MyType {
        private readonly long param1;
        private readonly string param2;

        internal MyType(long param1, string param2) {
            this.param1 = param1;
            this.param2 = param2;
        }

        [KSField]
        public long FirstParam => param1;

        [KSField]
        public string SecondParam => param2;

        [KSMethod]
        public double AddSomething(long toAdd) {
            return param1 + toAdd;
        }
    }
}
```

Additionally it has to be added to the `KontrolSystemKSPRegistry` using

```
registry.RegisterModule(BindingGenerator.BindModule(typeof(MyModule)));
```

Then

- The new module will have the name `ksp::my`
- It will contain two functions
  - `fn some_calculation(param1 : float, param2 : float) -> float`
  - `fn create_my_type(param1 : int, param2 : string) -> ksp::my::MyType`
- It will also contain a type `MyType` with
  - Property `.first_param : float`
  - Property `.second_param : string`
  - Method `.add_something(to_add : string)`
- The module class `MyModule` can become very big quickly, that is wy in most cases a `partial class` is used.

Notable things:

- Names will generally be converted from CamelCase to snake_case
- TO2 `int` is a C# `long` (using a C# `int` will not work)
- TO2 `float` is a C# `double` (using a C# `float` will not work)
- TO2 has no `new` operator or statement. I.e. to create an instance of `MyType` one has to provide a function returning it.

6.2.2 Direct binding

Existing types can be bound directly, the `Vector3Binding.cs` is a good example covering all the cases.

Be aware that the type mapping has to be done manually here and might lead to runtime exception if done incorrectly (i.e. standard binding is much easier and safer).

Direct binding should only be used if performance if an issue (i.e. the type might be used a lot) or operator binding is desired.

6.3 Project structure

The project is separeted in the follow sub-projects/assemblies

- `Parsing`
  - This is a pretty simple and generic parser combinator library (i.e. a library supporting the creation of aand kind of text-based parsers)
  - Indipendent to the game or the unity engine
- `Parsing-Test`
  - All the unit tests for the `Parsing` library
- `TO2`
  - The core of the TO2 language containing

        * Parser
        * Compiler
            · Which is done by parsing the script to an abstract syntax tree which then generates the IL-code via `System.Reflection.Emit`
        * Bindings to the `core` moddule
    - Has a dependeny to the `Parsing` library
    - Indipendent to the game or the unity engine
- `TO2-Test`
    - All the unit tests for the `TO2` library
- `KSP2Runtime`
    - The bindings to Kerbal Space Progam 2
        * This is the `ksp` module
    - Has a dependeny to the `TO2` library
    - Also contains the `std` module and example scrips written in TO2 itself
    - Only has dependencies to the game and the unity engine. Should remain indipendent to the modding framework
- `KSP2Runtime-Test`
    - All the unit tests for the `KSP2Runtime` library
- `SpaceWarpMod`
    - The `SpaceWarp` mod tying it all together
    - Has a dependeny to the `KSP2Runtime` library
    - … will most likely be replaced once there is an official mod loader

6.4 Thinks to improve aka. pain points

The following list is be no means priority sorted and might jump from "probably trivial" to megalomania.

6.4.1 Extend the `std::` lib

The plugin ships with a set of example scripts including a `std::` lib containing helpers for various common maneuvers. There are still a lot maneuvers missing and the existing ones have a lot of short comings

- `launch_rocket` is very basic and does not do a real gravity turn. It also might also end in a orbit where the periapsis is inside the atmosphere
- … there is no "launch space plane" script
- `exec_node` does not work well when there is a SOI change
- … there very little support to calculate SOI exit orbits (e.g. planing an ideal Duna transfer)
- … there is no docking support
    - Rendezvous calculation/execution could also be improved on

- … there is not atmospheric land for rockets (the Elon-style … not the "Drop the boosters and open parachute"-style)
- … there is landing for planes
- … and what about rovers?

6.4.2 Extend the `ksp::vessel` bindings

- Bindings for the vessel and the various modules is progressing nicely, but there are probably still a lot of telemetry data that might be interesting to have in a script
- … there is not binding to initiate resource transfer
- … mode switching for engines needs to be tested
- … make individual engine thrust available (to make something like throttle controlled avionics at least possible)

6.4.3 Extend the `ksp::game` bindings

- Improve time-wrap functionality (switching between rails- and physics-mode)

6.4.4 General additions

- Some sort of storage per vessel for scripts to store/restore some sort of state (e.g. where to pick up after hibernation)

6.4.5 UI improvements

- Syntax highlighting for the in-game script editor
- Highlight compilation errors in in-game editor
- On that note: Find a reliable way to disable game-input while an IMGUI text field has focus
  - … maybe migrate away from IMGUI?
- Have keyboard input in the CONSOLE (like `kOS` had)
- On that note: Have some sort of REPL mode to just evaluate expressions

6.4.6 UI for scripts?

- Allow scripts to open there own windows/dialogs?
- … dare I say: Graphs … that would be cool

6.4.7 Improve tooling

- Add some basic autocompletion to VS-code
  - Probably not an lsp-server (cough), but using a generated json containing all the type names and pre-defined functions/modules

6.5 Disclaimer

Any form of help is welcomed and appreciated.