

# Package: unifir (via r-universe)

August 30, 2026

**Type** Package

**Title** A Unifying API for Calling the 'Unity' '3D' Video Game Engine

**Version** 0.2.4.9000

**Description** Functions for the creation and manipulation of scenes and objects within the 'Unity' '3D' video game engine (<<https://unity.com/>>). Specific focuses include the creation and import of terrain data and 'GameObjects' as well as scene management.

**License** MIT + file LICENSE

**Depends** R (>= 3.5.0)

**Imports** glue, methods, proceduralnames, R6, utils

**Suggests** terrainr, covr, knitr, lintr, pkgdown, rmarkdown, styler, testthat (>= 3.0.0), terra, sf

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.1

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**URL** <https://docs.ropensci.org/unifir/>,  
<https://github.com/ropensci/unifir>

**BugReports** <https://github.com/ropensci/unifir/issues>

**VignetteBuilder** knitr

**Roxygen** list(markdown = TRUE)

**Repository** <https://ropensci.r-universe.dev>

**Date/Publication** 2024-02-01 18:22:29 UTC

**RemoteUrl** <https://github.com/ropensci/unifir>

**RemoteRef** main

**RemoteSha** 197a3d5dd5df79e97684532e190c3b59ab5abd05

Contents

|                                 |           |
|---------------------------------|-----------|
| action . . . . .                | 2         |
| add_default_player . . . . .    | 3         |
| add_light . . . . .             | 6         |
| add_prop . . . . .              | 7         |
| add_texture . . . . .           | 8         |
| associate_coordinates . . . . . | 9         |
| available_assets . . . . .      | 10        |
| check_debug . . . . .           | 11        |
| create_if_not . . . . .         | 11        |
| create_terrain . . . . .        | 11        |
| create_unity_project . . . . .  | 13        |
| find_unity . . . . .            | 14        |
| get_asset . . . . .             | 15        |
| import_asset . . . . .          | 15        |
| instantiate_prefab . . . . .    | 16        |
| load_png . . . . .              | 18        |
| load_scene . . . . .            | 19        |
| make_script . . . . .           | 20        |
| new_scene . . . . .             | 21        |
| read_raw . . . . .              | 22        |
| save_scene . . . . .            | 23        |
| set_active_scene . . . . .      | 24        |
| set_script_defaults . . . . .   | 25        |
| unifir_prop . . . . .           | 25        |
| unity_version . . . . .         | 26        |
| validate_path . . . . .         | 27        |
| waiver . . . . .                | 28        |
| <b>Index</b>                    | <b>30</b> |

---

|        |                                          |
|--------|------------------------------------------|
| action | <i>Build and execute a unifir_script</i> |
|--------|------------------------------------------|

---

Description

Build and execute a unifir\_script

Usage

action(script, write = TRUE, exec = TRUE, quit = TRUE)

**Arguments**

|        |                                                                                                           |
|--------|-----------------------------------------------------------------------------------------------------------|
| script | The unifir_script object (as generated by <a href="#">make_script</a> ) to build and execute.             |
| write  | Boolean: Write the generated script to a file?                                                            |
| exec   | Boolean: Execute the script inside of the Unity project? Note that if write = FALSE, exec cannot be TRUE. |
| quit   | Boolean: Quit Unity after execution?                                                                      |

**Value**

If exec = FALSE, the original unifir\_script object passed to script. If exec = TRUE, the same unifir\_script object with its props replaced by the C# they generate.

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script",
  unity = waiver()
)

# Then add any number of props to it:
script <- add_light(script)

# Then call `action` to execute the script!

if (interactive()) {
  action(script)
}
```

---

|                    |                                    |
|--------------------|------------------------------------|
| add_default_player | <i>Add assets to a Unity scene</i> |
|--------------------|------------------------------------|

---

**Description**

These functions add assets available at [https://github.com/mikemahoney218/unity\\_assets/](https://github.com/mikemahoney218/unity_assets/) to a Unity scene.

**Usage**

```
add_default_player(
  script,
  controller = c("Player", "FootstepsPlayer", "JetpackPlayer", "Third Person"),
  asset_directory = NULL,
  lazy = TRUE,
```

```

        method_name = NULL,
        destination_scene = NULL,
        x_position = 0,
        y_position = 0,
        z_position = 0,
        x_scale = 1,
        y_scale = 1,
        z_scale = 1,
        x_rotation = 0,
        y_rotation = 0,
        z_rotation = 0,
        exec = TRUE
    )

add_default_tree(
    script,
    tree,
    asset_directory = NULL,
    lazy = TRUE,
    method_name = NULL,
    destination_scene = NULL,
    x_position = 0,
    y_position = 0,
    z_position = 0,
    x_scale = 1,
    y_scale = 1,
    z_scale = 1,
    x_rotation = 0,
    y_rotation = 0,
    z_rotation = 0,
    exec = TRUE
)

```

## Arguments

|                 |                                                                                                                                                                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| script          | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function.                                                                                                                                                                  |
| controller      | Which controller to use. "Player", the default, is a simple first-person controller. "FootstepsPlayer" adds footsteps to this controller, while "JetpackPlayer" adds a "jetpack" with limited fuel. ""Third Person" lets you control a small cylinder in third person. |
| asset_directory | A file path to the directory containing the asset, or alternatively, to which the default assets should be saved. Defaults to tools::R_user_dir("unifir").                                                                                                             |
| lazy            | Boolean: if TRUE, unifir will attempt to only copy the files once per run of a script; if FALSE, unifir will copy the files as many times as requested, overwriting pre-existing files each time.                                                                      |

|                                    |                                                                                                                                                                                    |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| method_name                        | The internal name to use for the C# method created. Will be randomly generated if not set.                                                                                         |
| destination_scene                  | Optionally, the scene to instantiate the prefabs in. Ignored if NULL, the default.                                                                                                 |
| x_position, y_position, z_position | The position of the GameObject in world space.                                                                                                                                     |
| x_scale, y_scale, z_scale          | The scale of the GameObject (relative to its parent object).                                                                                                                       |
| x_rotation, y_rotation, z_rotation | The rotation of the GameObject to create, as Euler angles.                                                                                                                         |
| exec                               | Logical: Should the C# method be included in the set executed by MainFunc?                                                                                                         |
| tree                               | Which tree to use. There are currently 12 generic tree objects available, named "tree_1" through "tree_12". The number of a tree (1-12) can be specified instead of the full name. |

### Details

In effect, these functions provide a thin wrapper across [instantiate\\_prefab](#) and [import\\_asset](#). By providing the directory an asset is stored in, and the path to the prefab file once that directory has been copied into Unity, these files will add prefabs to specified locations throughout the scene. This function will also download the necessary assets and handles specifying file paths.

[add\\_default\\_player](#) adds "player" controllers to a Unity scene. [add\\_default\\_tree](#) adds tree GameOb-jects.

### Value

The `unifir_script` object passed to script, with props for adding assets appended.

### See Also

Other props: [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

### Examples

```
if (interactive()) {
  # First, create a script object.
  # CRAN doesn't have Unity installed, so pass
  # a waiver object to skip the Unity-lookup stage:
  script <- make_script("example_script", unity = waiver())

  # Now add props:
  script <- add_default_player(script)
  script <- add_default_tree(script, 1)
```

```

    script <- save_scene(script)
  }

  # Lastly, execute the script via the `action` function

```

---

|           |                                     |
|-----------|-------------------------------------|
| add_light | <i>Add a light to a Unity scene</i> |
|-----------|-------------------------------------|

---

### Description

This function creates light objects within a Unity scene. This function can only add one light at a time – call the function multiple times to add more than one light.

### Usage

```

add_light(
  script,
  light_type = c("Directional", "Point", "Spot", "Area"),
  method_name = NULL,
  light_name = "Light",
  x_position = 0,
  y_position = 0,
  z_position = 0,
  x_scale = 1,
  y_scale = 1,
  z_scale = 1,
  x_rotation = 50,
  y_rotation = -30,
  z_rotation = 0,
  exec = TRUE
)

```

### Arguments

|                                    |                                                                                                                                                                                       |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| script                             | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function.                                                                                 |
| light_type                         | One of "Directional", "Point", "Spot", or "Area". See <a href="https://docs.unity3d.com/Manual/Lighting.html">https://docs.unity3d.com/Manual/Lighting.html</a> for more information. |
| method_name                        | The internal name to use for the C# method created. Will be randomly generated if not set.                                                                                            |
| light_name                         | The name to assign the Light object.                                                                                                                                                  |
| x_position, y_position, z_position | The position of the GameObject in world space.                                                                                                                                        |
| x_scale, y_scale, z_scale          | The scale of the GameObject (relative to its parent object).                                                                                                                          |

x\_rotation, y\_rotation, z\_rotation

The rotation of the GameObject to create, as Euler angles.

exec

Logical: Should the C# method be included in the set executed by MainFunc?

### Value

The unifir\_script object passed to script, with props for adding lights appended.

### See Also

Other props: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

### Examples

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script", unity = waiver())

# Now add props:
script <- add_light(script)

# Lastly, execute the script via the `action` function
```

---

add\_prop

*Add a prop to a unifir script*

---

### Description

This function is exported so that developers can add their own props in new packages, without needing to re-implement the prop and script classes themselves. It is not expected that end users will need this function.

### Usage

```
add_prop(script, prop, exec = TRUE)
```

### Arguments

script      A script object (from [make\\_script](#)) to append the prop to.

prop        A unifir\_prop object (from [unifir\\_prop](#)) to add to the script.

exec        Logical: Should the method created by the prop be called in the MainFunc method?

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

**Examples**

```
script <- make_script("example_script", unity = waiver())
prop <- unifir_prop(
  prop_file = waiver(), # Must be a file that exists or waiver()
  method_name = NULL, # Auto-generated if NULL or NA
  method_type = "ExampleProp", # Length-1 character vector
  parameters = list(), # Not validated, usually a list
  build = function(script, prop, debug) {},
  using = character(0)
)
script <- add_prop(script, prop)
```

---

add\_texture

---

*Add a Texture2D layer to a terrain tile object*


---

**Description**

This function adds a helper method, `AddTexture`, to the C# script. This function is typically used to add textures to heightmaps in a Unity scene, for instance by functions like [create\\_terrain](#). It requires some arguments be provided at the C# level, and so is almost always called with `exec = FALSE`.

**Usage**

```
add_texture(script, method_name = NULL, exec = FALSE)
```

**Arguments**

|             |                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| script      | A <code>unifir_script</code> object, created by <a href="#">make_script</a> or returned by an <code>add_prop_*</code> function. |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.                                      |
| exec        | Logical: Should the C# method be included in the set executed by <code>MainFunc</code> ?                                        |

**Value**

The `unifir_script` object passed to `script`, with an `AddTexture` method appended.

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script",
  unity = waiver()
)

# Now add props:
script <- add_texture(script)

# Lastly, execute the script via the `action` function
```

---

associate\_coordinates Associate vector coordinates with a raster surface for Unity import

---

**Description**

Unity uses a left-handed coordinate system, which is effectively "flipped" from our normal way of thinking about spatial coordinate systems. It also can only import terrain as square tiles of side  $2^x + 1$ , for  $x$  between 5 and 12. As a result, importing objects into a Unity scene so that they align with terrain surfaces is trickier than you'd expect. This function "associates" the XY coordinates from some sf object, likely a point data set, with some raster object.

**Usage**

```
associate_coordinates(object, raster, side_length = 4097)
```

**Arguments**

|             |                                                                                                                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object      | The sf object to take coordinates from. The object will be reprojected (via <a href="#">sf::st_transform</a> ) to align with raster.                                                                                                                                                                        |
| raster      | A raster or file path to a raster to associate coordinates with. Note that different rasters will produce different coordinate outputs; you should run this function with the same raster you plan on bringing into Unity. Any file or object that can be read via <a href="#">terra::rast</a> can be used. |
| side_length | The side length of terrain tiles, in map units, you intend to bring into Unity. Must be a value equal to $2^x + 1$ , for $x$ between 5 and 12. All functions in the <code>unifir</code> family default to 4097.                                                                                             |

**Value**

A data.frame with two columns, X and Y, representing the re-aligned coordinates. If object is point data (or anything object that `sf::st_coordinates` returns a single row for each row of), these rows will be in the same order as object (and so can be appended via `cbind`).

**Examples**

```
## Not run:
if (!isTRUE(as.logical(Sys.getenv("CI")))) {
  simulated_data <- data.frame(
    id = seq(1, 100, 1),
    lat = runif(100, 44.04905, 44.17609),
    lng = runif(100, -74.01188, -73.83493)
  )
  simulated_data <- sf::st_as_sf(
    simulated_data,
    coords = c("lng", "lat"),
    crs = 4326
  )
  output_files <- terrainr::get_tiles(simulated_data)
  temptiff <- tempfile(fileext = ".tif")
  terrainr::merge_rasters(output_files["elevation"][[1]], temptiff)
  associate_coordinates(simulated_data, temptiff)
}

## End(Not run)
```

---

available\_assets

---

*Vector of assets unifir can download and import*


---

**Description**

This object contains the set of assets unifir is able to download and import (through `get_asset` and `import_asset`). These objects are all released under permissive open-source licenses (currently, either CC-0 1.0 or MIT). More information on the assets may be found at [https://github.com/mikemahoney218/unity\\_assets](https://github.com/mikemahoney218/unity_assets).

**Usage**

```
available_assets
```

**Format**

A character vector with 13 elements, each representing an asset which can be imported.

**Source**

[https://github.com/mikemahoney218/unity\\_assets](https://github.com/mikemahoney218/unity_assets)

---

|             |                                                 |
|-------------|-------------------------------------------------|
| check_debug | <i>Check if unifir should run in debug mode</i> |
|-------------|-------------------------------------------------|

---

**Description**

When running in debug mode, unifir will write nothing to disk.

**Usage**

```
check_debug()
```

---

|               |                                             |
|---------------|---------------------------------------------|
| create_if_not | <i>Create directory if it doesn't exist</i> |
|---------------|---------------------------------------------|

---

**Description**

Create directory if it doesn't exist

**Usage**

```
create_if_not(path, recur = FALSE)
```

**Arguments**

|       |                                          |
|-------|------------------------------------------|
| path  | The path to be created                   |
| recur | Boolean: create directories recursively? |

---

|                |                                                          |
|----------------|----------------------------------------------------------|
| create_terrain | <i>Create a terrain tile with optional image overlay</i> |
|----------------|----------------------------------------------------------|

---

**Description**

Create a terrain tile with optional image overlay

**Usage**

```
create_terrain(
  script,
  method_name = NULL,
  heightmap_path,
  x_pos,
  z_pos,
  width,
  height,
  length,
  heightmap_resolution,
  texture_path = "",
  exec = TRUE
)
```

**Arguments**

|                       |                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------|
| script                | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function. |
| method_name           | The internal name to use for the C# method created. Will be randomly generated if not set.            |
| heightmap_path        | The file path to the heightmap to import as terrain.                                                  |
| x_pos, z_pos          | The position of the corner of the terrain.                                                            |
| width, height, length | The dimensions of the terrain tile, in linear units.                                                  |
| heightmap_resolution  | The resolution of the heightmap image.                                                                |
| texture_path          | Optional: the file path to the image to use as a terrain overlay.                                     |
| exec                  | Logical: Should the C# method be included in the set executed by MainFunc?                            |

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

**Examples**

```
if (requireNamespace("terra", quietly = TRUE)) {
  raster <- tempfile(fileext = ".tiff")
  r <- terra::rast(matrix(rnorm(1000^2, mean = 100, sd = 20), 1000),
    extent = terra::ext(0, 1000, 0, 1000)
  )
  terra::writeRaster(r, raster)

  script <- make_script("example_script",
    unity = waiver()
  )
}
```

```

        create_terrain(
            script,
            heightmap_path = raster,
            x_pos = 0,
            z_pos = 0,
            width = 1000,
            height = terra::minmax(r)[[2]],
            length = 1000,
            heightmap_resolution = 1000
        )
    }

```

---

create\_unity\_project    *Create a new Unity project.*

---

## Description

Create a new Unity project.

## Usage

```
create_unity_project(path, quit = TRUE, unity = NULL)
```

## Arguments

|       |                                                                                                                                                                                                                                                                                             |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path  | The path to create a new Unity project at.                                                                                                                                                                                                                                                  |
| quit  | Logical: quit Unity after creating the project?                                                                                                                                                                                                                                             |
| unity | The path to the Unity executable on your system (importantly, <i>not</i> the Unity-Hub executable). If NULL, checks to see if the environment variable or option unifir_unity_path is set; if so, uses that path (preferring the environment variable over the option if the two disagree). |

## Value

TRUE, invisibly.

## See Also

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

## Examples

```
if (interactive()) create_unity_project(file.path(tempdir(), "project"))
```

---

find\_unity

*Find the Unity executable on a machine.*


---

## Description

If the path to Unity is not provided to a function, this function is invoked to attempt to find it. To do so, it goes through the following steps:

1. Attempt to load the "unifir\_unity\_path" environment variable.
2. Attempt to load the "unifir\_unity\_path" option.

Assuming that neither points to an actual file, this function will then check the default installation paths for Unity on the user's operating system. If not found, this function will error.

## Usage

```
find_unity(unity = NULL, check_path = TRUE)
```

## Arguments

|            |                                                                                                                                                                                                                                                                         |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| unity      | Character: If provided, this function will quote the provided string (if necessary) and return it.                                                                                                                                                                      |
| check_path | Logical: If TRUE, this function will check if the Unity executable provided as an argument, environment variable, or option exists. If it does not, this function will then attempt to find one, and will error if not found. If FALSE, this function will never error. |

## Value

The path to the Unity executable on the user's machine, as a length-1 character vector.

## See Also

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

## Examples

```
if (interactive()) {
  try(find_unity())
}
```

---

|           |                                   |
|-----------|-----------------------------------|
| get_asset | <i>Download prefabs for Unity</i> |
|-----------|-----------------------------------|

---

### Description

This is a simple helper function downloading the assets stored at [https://github.com/mikemahoney218/unity\\_assets](https://github.com/mikemahoney218/unity_assets).

### Usage

```
get_asset(asset, directory = NULL)
```

### Arguments

|           |                                                                                                                                          |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------|
| asset     | The asset to download. Available asset names are provided in <a href="#">available_assets</a> .                                          |
| directory | Optionally, the directory to extract the downloaded models in. If NULL, the default, saves to <code>tools::R_user_dir("unifir")</code> . |

### See Also

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

### Examples

```
if (interactive()) {
  get_asset(asset = "tree_1", directory = tempdir())
}
```

---

|              |                                  |
|--------------|----------------------------------|
| import_asset | <i>Import assets into Unity.</i> |
|--------------|----------------------------------|

---

### Description

Import assets into Unity.

### Usage

```
import_asset(script, asset_path, lazy = TRUE)
```

**Arguments**

|            |                                                                                                                                                                                                                                                                                                                 |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| script     | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function.                                                                                                                                                                                                           |
| asset_path | The file path to the asset to import. If a directory, the entire directory will be recursively copied. Note that this function doesn't have a method_name argument: the asset_path is used as the method name. This function is not currently vectorized; call it separately for each asset you need to import. |
| lazy       | Boolean: if TRUE, unifir will attempt to only copy the files once per run of a script; if FALSE, unifir will copy the files as many times as requested, overwriting pre-existing files each time.                                                                                                               |

**Value**

script with a new prop.

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script",
  unity = waiver()
)

# CRAN also doesn't have any props to install,
# so we'll make a fake prop location:
prop_directory <- file.path(tempdir(), "props")
dir.create(prop_directory)

# Now add props:
script <- import_asset(script, prop_directory)

# Lastly, execute the script via the `action` function
```

## Description

This function creates objects (specifically, prefabs) within a Unity scene. This function is vectorized over all functions from `prefab_path` through `z_rotation`; to add multiple objects, simply provide vectors to each argument. Note that all arguments will be automatically recycled if not the same length; this may produce undesired results. This function is only capable of altering a single scene at once – call the function multiple times if you need to manipulate multiple scenes.

## Usage

```
instantiate_prefab(
    script,
    method_name = NULL,
    destination_scene = NULL,
    prefab_path,
    x_position = 0,
    y_position = 0,
    z_position = 0,
    x_scale = 1,
    y_scale = 1,
    z_scale = 1,
    x_rotation = 0,
    y_rotation = 0,
    z_rotation = 0,
    exec = TRUE
)
```

## Arguments

|                                                 |                                                                                                                                                                                            |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>script</code>                             | A <code>unifir_script</code> object, created by <a href="#">make_script</a> or returned by an <code>add_prop_*</code> function.                                                            |
| <code>method_name</code>                        | The internal name to use for the C# method created. Will be randomly generated if not set.                                                                                                 |
| <code>destination_scene</code>                  | Optionally, the scene to instantiate the prefabs in. Ignored if <code>NULL</code> , the default.                                                                                           |
| <code>prefab_path</code>                        | File path to the prefab to be instantiated. This should be relative to the Unity project root directory, and likely begins with "Assets". Alternatively, if this is one of the elements in |
| <code>x_position, y_position, z_position</code> | The position of the <code>GameObject</code> in world space.                                                                                                                                |
| <code>x_scale, y_scale, z_scale</code>          | The scale of the <code>GameObject</code> (relative to its parent object).                                                                                                                  |
| <code>x_rotation, y_rotation, z_rotation</code> | The rotation of the <code>GameObject</code> to create, as Euler angles.                                                                                                                    |
| <code>exec</code>                               | Logical: Should the C# method be included in the set executed by <code>MainFunc</code> ?                                                                                                   |

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script", unity = waiver())

# Now add props:
script <- instantiate_prefab(script, prefab_path = "Assets/some.prefab")

# Lastly, execute the script via the `action` function
```

load\_png

*Create a Texture2D from a PNG file***Description**

This function adds a helper method, LoadPNG, to the C# script. This function is typically used by other C# methods to bring in textures into a Unity scene, for instance by functions like [create\\_terrain](#). It requires some arguments be provided at the C# level, and so is almost always called with `exec = FALSE`.

**Usage**

```
load_png(script, method_name = NULL, exec = FALSE)
```

**Arguments**

|             |                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| script      | A <code>unifir_script</code> object, created by <a href="#">make_script</a> or returned by an <code>add_prop_*</code> function. |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.                                      |
| exec        | Logical: Should the C# method be included in the set executed by <code>MainFunc</code> ?                                        |

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script", unity = waiver())

# Then add any number of props to it:
script <- load_png(script)

# Then call `action` to execute the script!
```

---

|            |                                         |
|------------|-----------------------------------------|
| load_scene | <i>Load a scene in a Unity project.</i> |
|------------|-----------------------------------------|

---

**Description**

Load a scene in a Unity project.

**Usage**

```
load_scene(script, scene_name, method_name = NULL, exec = TRUE)
```

**Arguments**

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| script      | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function. |
| scene_name  | The name of the scene to load.                                                                        |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.            |
| exec        | Logical: Should the C# method be included in the set executed by MainFunc?                            |

**See Also**

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script", unity = waiver())
```

```
# Now add props:
script <- load_scene(script, scene_name = "some_scene")

# Lastly, execute the script via the `action` function
```

---

|             |                                       |
|-------------|---------------------------------------|
| make_script | Create an empty unifir_script object. |
|-------------|---------------------------------------|

---

## Description

unifir relies upon "script" objects, which collect "prop" objects (C# methods) which then may be executed within a Unity project via the [action](#) function.

## Usage

```
make_script(
  project,
  script_name = NULL,
  scene_name = NULL,
  unity = find_unity(),
  initialize_project = NULL
)
```

## Arguments

|                    |                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| project            | The directory path of the Unity project.                                                                                                                                                             |
| script_name        | The file name to save the script at. The folder location and file extensions will be added automatically.                                                                                            |
| scene_name         | The default scene to operate within. If a function requires a scene name and one is not provided, this field will be used.                                                                           |
| unity              | The location of the Unity executable to create projects with.                                                                                                                                        |
| initialize_project | If TRUE, will call <a href="#">create_unity_project</a> to create a Unity project at project. If FALSE, will not create a new project. If NULL, will create a new project if project does not exist. |

## Value

A unifir\_script object.

## Examples

```
# Create an empty script file
# In practice, you'll want to set `project` to the project path to create
# and `unity` to `NULL` (the default)
make_script(project = waiver(), unity = waiver())
```

---

|           |                                               |
|-----------|-----------------------------------------------|
| new_scene | <i>Create a new scene in a Unity project.</i> |
|-----------|-----------------------------------------------|

---

## Description

Create a new scene in a Unity project.

## Usage

```
new_scene(
  script,
  setup = c("EmptyScene", "DefaultGameObjects"),
  mode = c("Additive", "Single"),
  method_name = NULL,
  exec = TRUE
)
```

## Arguments

|             |                                                                                                                                                                               |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| script      | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function.                                                                         |
| setup       | One of "EmptyScene" ("No game objects are added to the new Scene.") or "DefaultGameObjects" ("Adds default game objects to the new Scene (a light and camera).")              |
| mode        | One of "Additive" ("The newly created Scene is added to the current open Scenes.") or "Single" ("All current open Scenes are closed and the newly created Scene are opened.") |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.                                                                                    |
| exec        | Logical: Should the C# method be included in the set executed by MainFunc?                                                                                                    |

## See Also

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

## Examples

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script",
```

```

    unity = waiver()
  )

  # Now add props:
  script <- new_scene(script)

  # Lastly, execute the script via the `action` function

```

---

read\_raw

*Read a RAW file in as a float array*


---

## Description

This function adds a helper method, ReadRaw, to the C# script. This function is typically used to bring in heightmaps into a Unity scene, for instance by functions like [create\\_terrain](#). It requires some arguments be provided at the C# level, and so is almost always called with `exec = FALSE`.

## Usage

```
read_raw(script, method_name = NULL, exec = FALSE)
```

## Arguments

|             |                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| script      | A <code>unifir_script</code> object, created by <a href="#">make_script</a> or returned by an <code>add_prop_*</code> function. |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.                                      |
| exec        | Logical: Should the C# method be included in the set executed by <code>MainFunc</code> ?                                        |

## See Also

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [save\\_scene\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

## Examples

```

# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script", unity = waiver())

# Now add props:
script <- read_raw(script)

# Lastly, execute the script via the `action` function

```

---

|            |                                         |
|------------|-----------------------------------------|
| save_scene | <i>Save a scene in a Unity project.</i> |
|------------|-----------------------------------------|

---

## Description

Save a scene in a Unity project.

## Usage

```
save_scene(script, scene_name = NULL, method_name = NULL, exec = TRUE)
```

## Arguments

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| script      | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function. |
| scene_name  | The name to save the scene to.                                                                        |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.            |
| exec        | Logical: Should the C# method be included in the set executed by MainFunc?                            |

## See Also

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [set\\_active\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

## Examples

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script",
  unity = waiver()
)

# Now add props:
script <- save_scene(script, scene_name = "some_scene")

# Lastly, execute the script via the `action` function
```

---

|                  |                                      |
|------------------|--------------------------------------|
| set_active_scene | <i>Set a single scene to active.</i> |
|------------------|--------------------------------------|

---

## Description

Set a single scene to active.

## Usage

```
set_active_scene(script, scene_name = NULL, method_name = NULL, exec = FALSE)
```

## Arguments

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| script      | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function. |
| scene_name  | The name of the scene to set as the active scene.                                                     |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.            |
| exec        | Logical: Should the C# method be included in the set executed by MainFunc?                            |

## See Also

Other props: [add\\_default\\_player\(\)](#), [add\\_light\(\)](#), [add\\_prop\(\)](#), [add\\_texture\(\)](#), [create\\_terrain\(\)](#), [import\\_asset\(\)](#), [instantiate\\_prefab\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [validate\\_path\(\)](#)

Other utilities: [add\\_default\\_player\(\)](#), [add\\_prop\(\)](#), [create\\_unity\\_project\(\)](#), [find\\_unity\(\)](#), [get\\_asset\(\)](#), [load\\_png\(\)](#), [load\\_scene\(\)](#), [new\\_scene\(\)](#), [read\\_raw\(\)](#), [save\\_scene\(\)](#), [validate\\_path\(\)](#), [waiver\(\)](#)

## Examples

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script",
  unity = waiver()
)

# Now add props:
script <- set_active_scene(script, scene_name = "some_scene")

# Lastly, execute the script via the `action` function
```

---

|                     |                                       |
|---------------------|---------------------------------------|
| set_script_defaults | <i>Fill in plot holes in a script</i> |
|---------------------|---------------------------------------|

---

**Description**

Fill in plot holes in a script

**Usage**

```
set_script_defaults(script, debug)
```

**Arguments**

|        |                                      |
|--------|--------------------------------------|
| script | The unfir_script to fill elements of |
| debug  | Boolean: run in debug mode?          |

---

|            |                                         |
|------------|-----------------------------------------|
| unfir_prop | <i>The class for unfir prop objects</i> |
|------------|-----------------------------------------|

---

**Description**

This function is exported so that developers can add their own props in new packages, without needing to re-implement the prop and script classes themselves. It is not expected that end users will need this function.

**Usage**

```
unfir_prop(prop_file, method_name, method_type, parameters, build, using)
```

**Arguments**

|             |                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| prop_file   | The system location for the C# template file                                                                                                                                             |
| method_name | The name of the method, in C# code                                                                                                                                                       |
| method_type | The type of the method (usually matches its file name); scripts can have multiple versions of the same method, each with different method_name values, all sharing the same method_type. |
| parameters  | Method-specific parameters, typically used in the build stage.                                                                                                                           |
| build       | A function that takes three arguments, script, prop, and debug, and uses those to construct the C# method.                                                                               |
| using       | A character vector of imports required for the method.                                                                                                                                   |

## Details

This function will check each argument for correctness. To be specific, it performs the following checks:

- `prop_file` must be either a waiver object (created by [waiver](#)) or a file path of length 1 pointing to a file that exists
- `method_name` will be automatically generated if not existing. If it exists, it must be a character vector of length 1
- `method_type` must be a character vector of length 1
- `build` must be a function with the arguments `script`, `prop`, and `debug` (in that order, with no other arguments). Any other arguments needed by your build function should be passed as `prop` parameters.
- `using` must be a character vector (of any length, including 0)

If your `prop` needs data or arguments beyond these, store them as a list in `parameters`, which is entirely unchecked.

## Value

An R6 object of class `unifir_prop`

## The debug argument

When `Sys.getenv(unifir_debugmode)` returns anything other than `""`, [action](#) runs in "debug mode". In addition to setting `exec` and `write` to `FALSE` in [action](#), this mode also attempts to disable any `prop` functionality that would make changes to the user's disk – no files or directories should be altered. In this mode, [action](#) will pass `debug = TRUE` as an argument to your `prop`; your `prop` should respect the debug mode and avoid making any changes.

## Examples

```
unifir_prop(
  prop_file = waiver(), # Must be a file that exists or waiver()
  method_name = NULL, # Auto-generated if NULL or NA
  method_type = "ExampleProp", # Length-1 character vector
  parameters = list(), # Not validated, usually a list
  build = function(script, prop, debug) {},
  using = character(0)
)
```

---

unity\_version

*Print the version of the Unity Editor in use.*

---

## Description

Print the version of the Unity Editor in use.

## Usage

```
unity_version(unity = NULL)
```

## Arguments

|       |                                                                                                                                                                                                                                                                                             |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| unity | The path to the Unity executable on your system (importantly, <i>not</i> the Unity-Hub executable). If NULL, checks to see if the environment variable or option unifir_unity_path is set; if so, uses that path (preferring the environment variable over the option if the two disagree). |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Value

A character vector of length 1 containing the version of Unity in use.

## Examples

```
try(
  unity_version()
)
```

---

|               |                                    |
|---------------|------------------------------------|
| validate_path | <i>Validate a file path exists</i> |
|---------------|------------------------------------|

---

## Description

[validate\\_path](#) creates a generic C# method which takes a single argument and checks to make sure it exists. Your C# code calling the method must provide the path to validate. [validate\\_single\\_path](#) hard-codes the path to check in the C# code. This allows you to specify the path to check from R.

## Usage

```
validate_path(script, method_name = NULL, exec = FALSE)
```

```
validate_single_path(script, path, method_name = NULL, exec = TRUE)
```

## Arguments

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| script      | A unifir_script object, created by <a href="#">make_script</a> or returned by an add_prop_* function. |
| method_name | The internal name to use for the C# method created. Will be randomly generated if not set.            |
| exec        | Logical: Should the C# method be included in the set executed by MainFunc?                            |
| path        | The file path to validate                                                                             |

**See Also**

Other props: `add_default_player()`, `add_light()`, `add_prop()`, `add_texture()`, `create_terrain()`, `import_asset()`, `instantiate_prefab()`, `load_png()`, `load_scene()`, `new_scene()`, `read_raw()`, `save_scene()`, `set_active_scene()`

Other utilities: `add_default_player()`, `add_prop()`, `create_unity_project()`, `find_unity()`, `get_asset()`, `load_png()`, `load_scene()`, `new_scene()`, `read_raw()`, `save_scene()`, `set_active_scene()`, `waiver()`

**Examples**

```
# First, create a script object.
# CRAN doesn't have Unity installed, so pass
# a waiver object to skip the Unity-lookup stage:
script <- make_script("example_script", unity = waiver())

# Now add props:
script <- validate_path(script) # Don't specify the path in R
script <- validate_single_path( # Specify the path in R
  script,
  "file_that_exists.txt"
)
```

---

waiver

*A waiver object.*


---

**Description**

This function is borrowed from `ggplot2`. It creates a "flag" object indicating that a value has been intentionally left blank (because it will be filled in by something else). Often, a function argument being missing or `NULL` will result in an error, while passing `waiver()` will cause the function to look elsewhere in the script for an acceptable value.

**Usage**

```
waiver()
```

**Value**

An empty list of class `waiver`.

**References**

H. Wickham. `ggplot2`: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

**See Also**

Other utilities: `add_default_player()`, `add_prop()`, `create_unity_project()`, `find_unity()`, `get_asset()`, `load_png()`, `load_scene()`, `new_scene()`, `read_raw()`, `save_scene()`, `set_active_scene()`, `validate_path()`

`waiver`

29

### **Examples**

`waiver()`

# Index

- \* **datasets**
  - available\_assets, 10
- \* **props**
  - add\_default\_player, 3
  - add\_light, 6
  - add\_prop, 7
  - add\_texture, 8
  - create\_terrain, 11
  - import\_asset, 15
  - instantiate\_prefab, 16
  - load\_png, 18
  - load\_scene, 19
  - new\_scene, 21
  - read\_raw, 22
  - save\_scene, 23
  - set\_active\_scene, 24
  - validate\_path, 27
- \* **scripts**
  - make\_script, 20
- \* **utilities**
  - add\_default\_player, 3
  - add\_prop, 7
  - create\_unity\_project, 13
  - find\_unity, 14
  - get\_asset, 15
  - load\_png, 18
  - load\_scene, 19
  - new\_scene, 21
  - read\_raw, 22
  - save\_scene, 23
  - set\_active\_scene, 24
  - validate\_path, 27
  - waiver, 28
- action, 2, 20, 26
- add\_default\_player, 3, 5, 7–9, 12–16, 18, 19, 21–24, 28
- add\_default\_tree, 5
- add\_default\_tree (add\_default\_player), 3
- add\_light, 5, 6, 8, 9, 12, 16, 18, 19, 21–24, 28
- add\_prop, 5, 7, 7, 9, 12–16, 18, 19, 21–24, 28
- add\_texture, 5, 7, 8, 8, 12, 16, 18, 19, 21–24, 28
- associate\_coordinates, 9
- available\_assets, 10, 15
- cbind, 10
- check\_debug, 11
- create\_if\_not, 11
- create\_terrain, 5, 7–9, 11, 16, 18, 19, 21–24, 28
- create\_unity\_project, 5, 8, 13, 14, 15, 18–24, 28
- find\_unity, 5, 8, 13, 14, 15, 18, 19, 21–24, 28
- get\_asset, 5, 8, 10, 13, 14, 15, 18, 19, 21–24, 28
- import\_asset, 5, 7–10, 12, 15, 18, 19, 21–24, 28
- instantiate\_prefab, 5, 7–9, 12, 16, 16, 18, 19, 21–24, 28
- load\_png, 5, 7–9, 12–16, 18, 18, 19, 21–24, 28
- load\_scene, 5, 7–9, 12–16, 18, 19, 21–24, 28
- make\_script, 3, 4, 6–8, 12, 16–19, 20, 21–24, 27
- new\_scene, 5, 7–9, 12–16, 18, 19, 21, 22–24, 28
- read\_raw, 5, 7–9, 12–16, 18, 19, 21, 22, 23, 24, 28
- save\_scene, 5, 7–9, 12–16, 18, 19, 21, 22, 23, 24, 28
- set\_active\_scene, 5, 7–9, 12–16, 18, 19, 21–23, 24, 28
- set\_script\_defaults, 25

`sf::st_coordinates`, [10](#)  
`sf::st_transform`, [9](#)  
  
`terra::rast`, [9](#)  
  
`unifir_prop`, [7](#), [25](#)  
`unity_version`, [26](#)  
  
`validate_path`, [5](#), [7–9](#), [12–16](#), [18](#), [19](#), [21–24](#),  
    [27](#), [27](#), [28](#)  
`validate_single_path`, [27](#)  
`validate_single_path(validate_path)`, [27](#)  
  
`waiver`, [5](#), [8](#), [13–15](#), [18](#), [19](#), [21–24](#), [26](#), [28](#), [28](#)