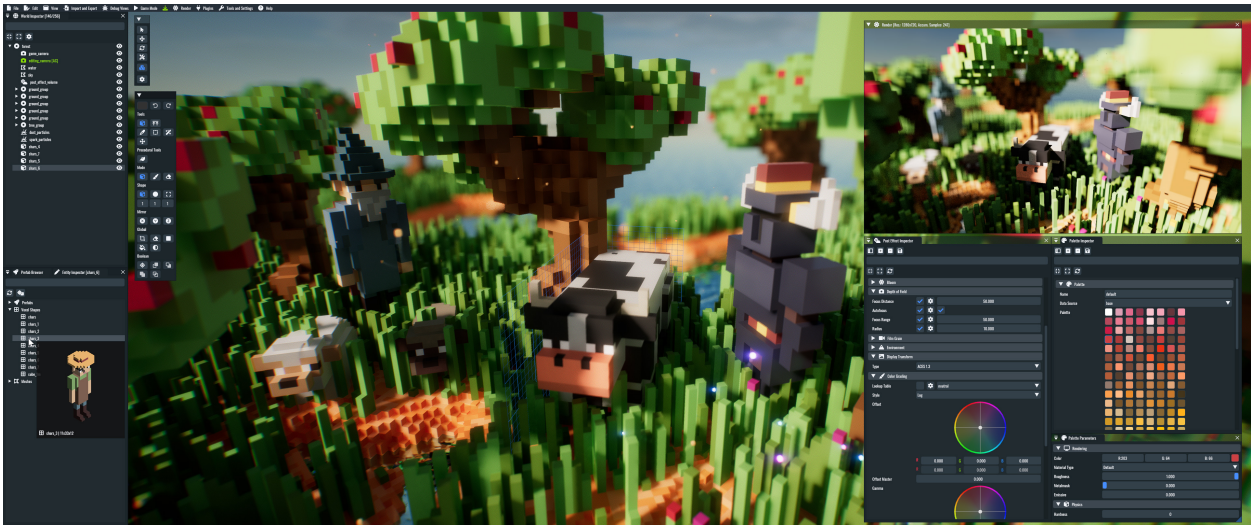
lOLITE - Voxel Game Engine

Missing Deadlines (Benjamin Wrensch)

Jan 26, 2024

CONTENTS

IOLITE is a real-time voxel game engine powered by path tracing. Create the game of your dreams in highly detailed and dynamic worlds using the embedded editor and C/C++ or Lua.



CHAPTER

ONE

FEATURE OVERVIEW

- Highly specialized for dealing with large, highly detailed, and dynamic voxel worlds, made possible by modern desktop-level CPUs and GPUs
- Features fully dynamic real-time global illumination using a mixture of path tracing and screen space techniques; no ray tracing capable GPU required
- Implement your game using C/C++ or Lua; you can even create a custom scripting backend for the language of your choice
- Comes with a fully-featured editor for creating game worlds. Use our open-source plugins to extend the functionality, like, e.g., authoring voxel assets directly in the editor
- Extend the engine and editor functionality using the plugin system in conjunction with the native C/C++ API
- Highly flexible and thus not exclusively tied to creating games. Use IOLITE as a complete game creation package, solely as a renderer, for creating a procedural voxel editor, and more. The possibilities are endless
- Features subsystems for sound, particles, destruction, post-processing, and more right out of the box

CHAPTER

TWO

DOWNLOAD IOLITE

Important: You can download the latest version of IOLITE via the landing page of our website.

CHAPTER

THREE

USEFUL LINKS AND RESOURCES

Here are links to the most important IOLITE resources:

IOLITE website
https://iolite-engine.com

Subscribe to IOLITE PRO
https://iolite-engine.com/subscribe

Subscriber area
https://iolite-engine.com/subscribers

This documentation
https://docs.iolite-engine.com

Development blog
https://iolite-engine.com/blog

Our YouTube channel
https://www.youtube.com/@missing_deadlines

Our Discord community
https://discord.com/invite/SZjfhw7z75

Our public GitHub repository
https://github.com/MissingDeadlines/iolite

Follow us on Mastodon
https://mastodon.missing-deadlines.com/@benjamin

CHAPTER

FOUR

SUPPORT THE DEVELOPMENT

If you would like to support the development of IOLITE, please consider subscribing to IOLITE PRO. In addition to all the features of the free version, the PRO version supports the following:

4.1 Features of IOLITE PRO

- **Native C/C++ API**
  Write highly modular plugins in C, C++, or in any other language that supports C bindings.
- **Powerful triangle mesh voxelization algorithm**
  Turn complex meshes with materials and textures exported from any 3D software that supports glTF, like Blender, 3ds Max, etc. or asset libraries like Quixel Megascans and PlantCatalog into beautiful voxel assets.
- **Extended export options for path-traced renders**
  Export HDR renders in the highest quality using the EXR file format.
- **Option to disable the splash screen**
  IOLITE PRO offers the option to disable the splash screen shown during startup.
- **DRM free**
  Like the free version, IOLITE PRO is shipped without any DRM. Every downloaded build is yours forever to keep.

CHAPTER

FIVE

FREQUENTLY ASKED QUESTIONS

Here's a collection of frequently asked questions. Make sure to go through each one if you're encountering an issue. Chances are high that it is already known and documented here.

Note: If you do not find the answer you are looking for, head to our Discord community and raise the question there.

5.1 What can I do with IOLITE? What is its license? Is it open-source software?

IOLITE is available as a free and paid (PRO) version. You're **free to create either commercial or non-commercial software with both versions**. While IOLITE's engine core is considered proprietary software, some of its functionality is provided as *open-source plugins* licensed using the MIT license. The paid version finances the project's development and adds some additional features, as listed in *Features of IOLITE PRO*.

Both the PRO and the free versions are shipped without any form of DRM and share a straightforward license agreement:

```
IOLITE Software License Agreement
=================================

This license has a single goal: letting you create incredible things with IOLITE without␣
↪worrying about strict licensing rules.

Just remember that your support enables me to work on IOLITE for the long haul.

TL;DR
-----

Feel free to utilize IOLITE as desired, provided that you include all licenses and␣
↪acknowledge that no warranty is implied; use IOLITE at your own risk.

Full License
------------

Copyright 2023 Missing Deadlines (Benjamin Wrensch)

  1. You're allowed to use IOLITE for as many projects as you like (commercial or non-
↪commercial).
  2. This license and all other licenses included in the IOLITE software distribution␣
```

(continues on next page)

(continued from previous page)

```
↪have to be shipped with every project IOLITE is used for.
  3. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A␣
↪PARTICULAR PURPOSE AND NONINFRINGEMENT.
     IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES␣
↪OR OTHER LIABILITY,
     WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
     OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE␣
↪SOFTWARE.

Please do not download or install IOLITE if you do not agree to all of these terms.

If this license does not fit your needs, feel free to contact us any time; I'm sure we
↪'ll be able to work out a custom solution.
```

5.2 What are the overall system requirements?

General Requirements

- **OS:** Windows 10+ or Linux
- **GPU:** Dedicated GPU with support for Vulkan 1.2 or higher
- **CPU:** Recent multi-core CPU with support for the AVX2 instruction set

Optimal Performance

- **GPU:** NVIDIA GTX 1080 (or equivalent) and higher

5.3 Is IOLITE considered a finished product?

IOLITE is currently in its initial development phase and evolves constantly. The first *official release* will be marked by version 1.0.

Note: While creating amazing things with IOLITE is possible, we can not guarantee feature and API stability at this point. Please remember this when deciding to work on a project using IOLITE.

5.4 Help, IOLITE crashes during startup!

Many checks are in place to ensure you end up with a meaningful error message in any possible failure case. If you encounter a startup crash, the chances are high that your GPU driver is outdated. Please use the most recent drivers from NVIDIA or AMD:

Drivers for NVIDIA GPUs
https://www.nvidia.com/download/index.aspx

Drivers for AMD GPUs
https://www.amd.com/en/support

Using the latest proprietary driver is your best bet when using Linux. Please ensure that your GPU driver is working correctly before trying to launch IOLITE.

If you're experiencing issues with the standalone Linux builds, it's potentially due to mismatches between the runtime libraries utilized by IOLITE and your Linux distribution.

5.5 How the hell do I exit the game mode again!?

The default keybinding for opening the editor is `[F3]`. You can adjust this in the `keybindings_global.json` file.

5.6 I only see a simple default scene; where are the samples?

Head over to our GitHub repository and place the contents of the `iolite_samples` directory right next to the IOLITE executable, overwriting any existing files.

5.7 I would love to try writing a native plugin using the C/C++ API. Do I need IOLITE PRO?

The API header file is publicly available via our public GitHub repository. In addition, the free version can also load native plugins, so it's possible to evaluate this feature before deciding to subscribe. Loading non-factory plugins in the free version will trigger a watermark.

5.8 I've encountered an issue that is not referenced in this FAQ. What should I do?

Bugs and feature requests can be submitted via our issue tracker. You should also consider joining our Discord community and raising it there. See you there!

CHAPTER

SIX

GETTING STARTED

This section contains installation instructions and guides to get you started.

6.1 Installing portable builds (Windows and Linux)

Installing IOLITE is trivial. Simply download a release from the landing page or, if you have subscribed to IOLITE PRO, from the subscriber area and unzip it. Finally, launch the IOLITE executable. That's it.

If you encounter an issue when running IOLITE, please consult the section *Frequently asked questions*. If you are unable to solve the issue, please follow the steps depicted in *I've encountered an issue that is not referenced in this FAQ. What should I do?*.

6.2 Beta releases (PRO only)

Beta releases are released in between main releases for PRO subscribers. Please consider the following when working with a beta release of IOLITE:

Important: Beta releases are released more frequently and contain the latest features, but they can be a little less stable.

- Please switch to the `develop` branch in our public GitHub repository and search for a tag matching the version of IOLITE you are using
- Please ensure that you are using a matching version of the open-source plugins. Due to potential API changes, there are separate plugin releases available for the beta versions of IOLITE

6.3 Telemetry (Usage statistics)

IOLITE, by default, gathers some elementary **anonymous usage statistics** to aid the development.

Note: This system will be extended in the future also to report crashes.

The following data is collected for each of the logged events:

UUID
: A unique identifier based on a **hash** of the addresses of **all** the network adapters installed in your system, e.g., `8A36-E187-2C1F-51FB`. This hash **can not be** reversed to get access to the MAC addresses of your adapters

Platform
Either `Windows` or `Linux`

Build
Either `Free` or `Pro`

API version
The version of the API, e.g., `0.4.0`

Event name
The event's name, e.g., `startup`, `shutdown`, etc.

Event metadata
Currently unused and reserved for future use

You can turn off the telemetry system at any time by adding the following member to your `app_metadata.json` file:

```
"disable_telemetry": true
```

6.3.1 Guides

This section contains a collection of short guides and tutorials for IOLITE. All guides assume the default data source is available, and that you have loaded the default `basic` world.

Writing your first Hello World script in Lua

This short tutorial serves as a step-by-step guide to writing your first Lua script in IOLITE.

Important: This guide requires the latest version of the Lua plugin for IOLITE to be installed. Please check the section *Working with plugins* for more details.

Open up your favorite code editor and create a new file. Copy and paste the following Lua script, which logs two strings to the console:

```
Log.load()

-- Logs each time the script gets (re-)loaded
Log.log_info("Hello world! Script loaded!")

function OnActivate(entity)
  -- Logs once the component becomes active
  Log.log_info("Hello world! Component active!")
end
```

After that, continue with the following steps:

1. Store the script in `default/media/scripts/` and name it `hello_world.lua`
2. Open up IOLITE, ensure that the editor is active, and head over to the *World Inspector*
3. Create a new entity with a script component attached to it
4. In the property inspector, set the `Script` property to `hello_world` (without the extension)
5. Switch to the game mode by clicking `[Game Mode]` in the menu bar
6. Press `[F2]` to open up the console and check if the strings have been logged successfully

Keep IOLITE open and modify the strings passed to the log functions. Every time you save the script, it triggers a hot reload. Notice how the global log call gets executed while the call in `OnActivate` is not. This call can be, e.g., triggered by switching back and forth between the game mode and the editor; the editor can be activated using `[F3]`.

Importing a flipbook animation from MagicaVoxel

In this short tutorial, we will load a flipbook animation authored in MagicaVoxel.

1. Create a flipbook animation in MagicaVoxel and add the VOX file to a data source, like, e.g., `default/media/voxels`. For testing purposes, you can also use one of the animated voxel assets that are shipped with MagicaVoxel, like the dinosaur or the deer
2. Create a voxel shape for your asset by dragging it from the *Prefab Browser* to the viewport
3. Attach a *Flipbook Animation* component to the entity you just created
4. Head to the entity inspector and set the `First Frame` and `Last Frame` properties to values matching your animation
5. Set the `Speed` property to a fitting value. 20 frames per second can be a good starting point
6. Enable the `Loop` and `Play` flags

If everything works out, your animation should be playing.

CHAPTER

SEVEN

ENGINE DOCUMENTATION

This section covers some fundamentals of IOLITE's engine core, beginning by introducing the underlying design concepts. In addition, it presents the many subsystems and components which can be utilized for the creation of your game or software.

7.1 Introduction

IOLITE aims to make it easy to create fun and responsive voxel games. Let's start by going over the fundamentals first.

7.1.1 The overall game creation process

Compared to other *game engines* or *game makers*, a single copy of IOLITE does *not* support working with different *projects*. Instead, IOLITE aims to be portable and thus directly sources all data from the directories next to the executable. A single copy of IOLITE *becomes your game*, it's as simple as that.

7.1.2 Shipping your game

When working with IOLITE, there is no differentiation between the *game* and *the editor*. There is no separate editor application; the editor itself is part of IOLITE's runtime. A big plus is that you implicitly ship the tools you used to create your game in the first place, allowing your community to utilize the same toolset for modding.

When launching IOLITE for the first time, the editor is automatically activated after the startup. Whether the application should launch into the editor or directly into the game mode (also referred to as the main game state) can be configured. Accordingly, all you have to do when shipping your game is to make the game mode the default, as depicted in the section *Working with the app metadata file*.

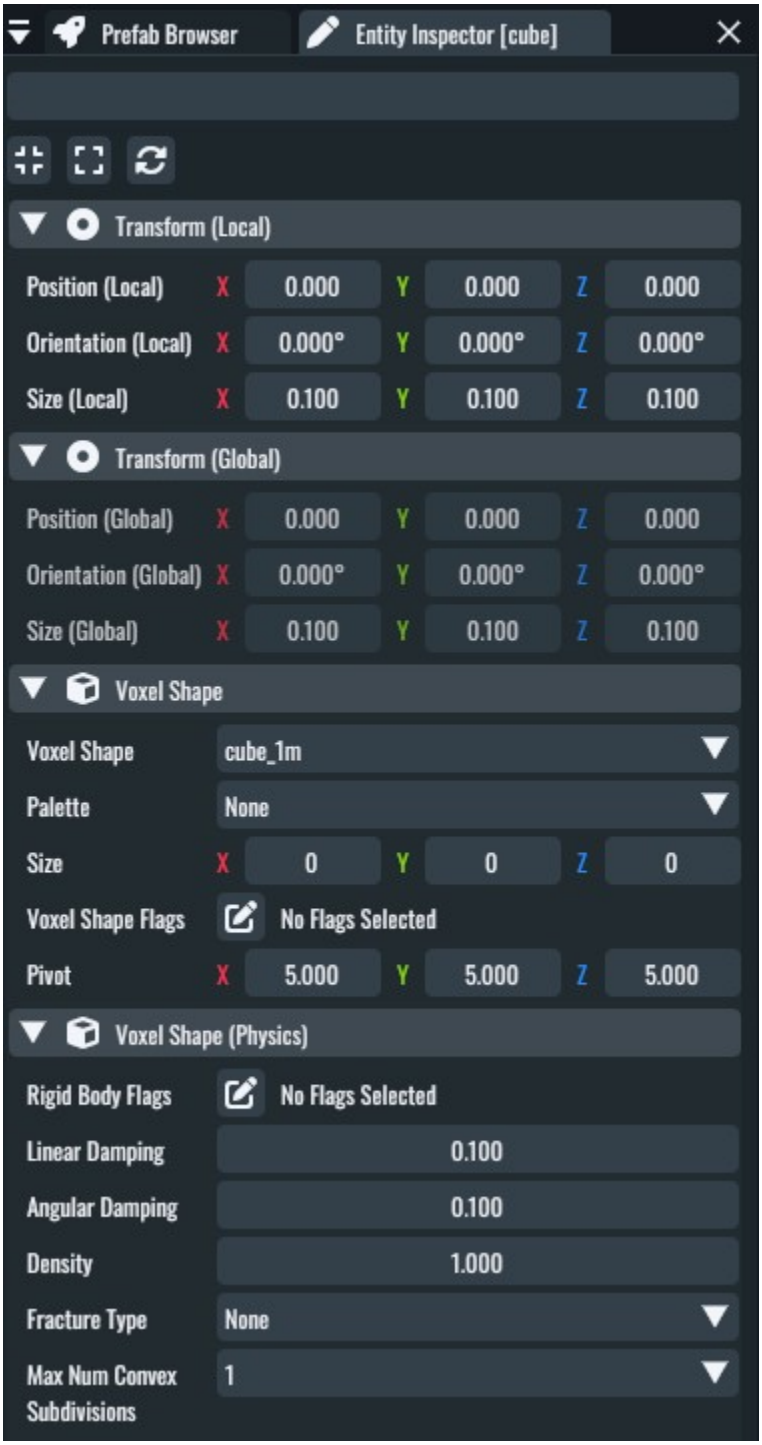
7.1.3 Entity Component System

The ECS (Entity Component System) is your primary tool for creating games with IOLITE.

Entities are functionless objects that can be decorated with components to assign them one or multiple purposes. For example, an entity with a node component can be positioned in three-dimensional space. Adding a character controller component unlocks the ability to interact with the physics environment, and adding a voxel shape component creates a visual representation for your character.

IOLITE's ECS comes with a very flexible *property system*, which allows, on the one hand, storing data with your components, which can then be accessed programmatically, either via the scripting backend or the native API. On the other hand, it allows configuring your components in the editor. The following image shows the property editor for a

selected entity in the editor, making it possible to inspect and modify properties of multiple components attached to a single entity:

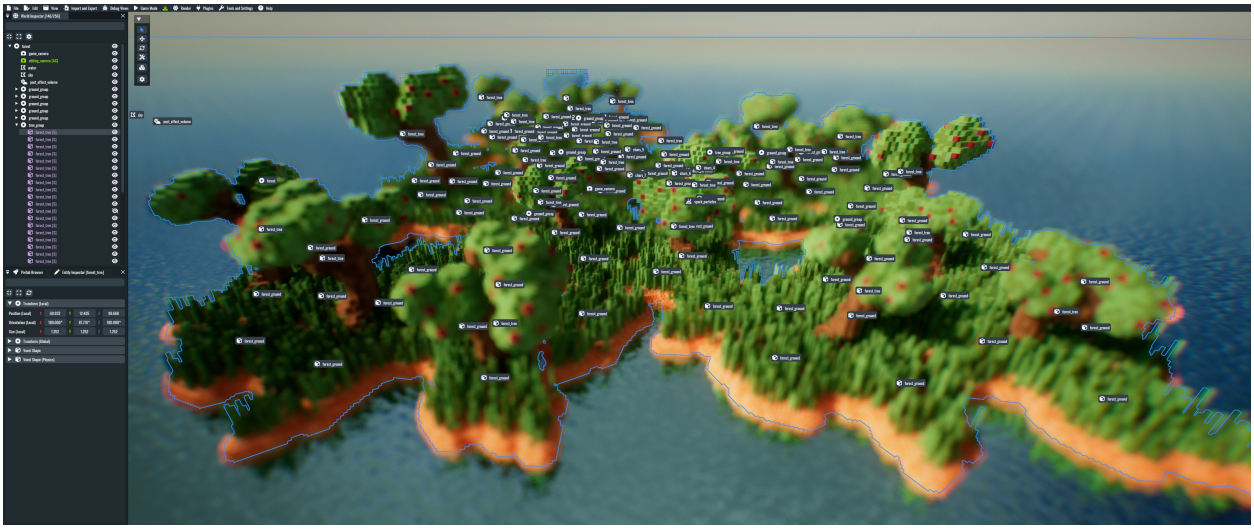


The ECS in IOLITE utilizes data-oriented principles and stores all data in SOA layout (Structure of Arrays) while ensuring the data stays as compact as humanly possible in memory. This makes it possible to work with large data sets very efficiently.

Note: If you are aching for a more in-depth explanation of the topic, you can find more information on Wikipedia.

7.1.4 Worlds

Worlds are your creative containers that store a scene as a hierarchy of nodes. Nodes keep a one-to-many relationship with other nodes, meaning that a single node can have multiple children but only a single parent node. The following screenshot shows the small forest sample world, made up of instantiations of a few different voxel shapes:



The *world inspector* on the editor's left side shows the mentioned node hierarchy.

A world is defined by its name, a single root node, and the hierarchy of nodes attached to it. Worlds can be saved to and loaded from disk, and you can use them to structure your project:

Multiple worlds store the levels for your game, while another single world can house the scripts and logic for your menu screen.

7.2 Data sources and the app metadata file

Every project reads all its data from one or many so-called data sources. Data sources in the development environment are plain directories in the application's root directory, right next to the executable.

When deploying a build, you can opt in to package the data of each of the data sources to an IPKG (IOLITE Package) file. To create packages, you can use the package generator script, available in our repository; see `python_scripts/package_generator.py`.

The portable builds of IOLITE are shipped with two data sources. The mandatory `base` data source is provided as a package, while the `default` data source is available as directories and files, serving the simple default scene. You can use the directory structure of the `default` data source as a reference for creating new data sources.

Important: The `base` data source contains mandatory assets and has to be available. It's loaded by default, and there is no need to load it manually.

7.2.1 Using data sources for modding

Data sources provide an incredibly flexible approach to modding applications created with IOLITE.

Let's say you want to replace a specific voxel asset in a game, like a tree in a different color or a particular script where you've added some little tweaks or new features. All you have to do is to create a new data source directory and place the modified assets in the correct path in the directory structure, matching the file path used in the base data source. After that, you must load your data source before loading the data sources containing the original assets, which shadows the assets in the base data source.

The order in which data sources are loaded is defined via the app metadata file, depicted in detail in the following section.

7.2.2 Working with the app metadata file

The app metadata data file is a JSON file with the following content:

```
{
    "application_name": "IOLITE",
    "organization_name": "Missing Deadlines",
    "version_string": "0.2.0",

    "data_sources": [
        "default"
    ],

    "initial_world": "default",
    "initial_game_state": "Editing"
}
```

The app metadata allows you to adjust basic properties like your application's name and your organization. In addition, it is also in charge of defining the data sources that should be used to source files from. Data sources are loaded in the given order, and data provided by data sources listed first is prioritized.

Here's an overview of all the different parameters:

application_name (String)
: The name of your application.

organization_name (String)
: The name of your organization (if any).

version_string (String)
: Version string following the Semantic Versioning scheme.

data_sources (Array of strings)
: The data sources used for your project. Data sources are loaded in the provided order. The engine starts searching for files in the first data sources and, if the file in question is found, skips searching all the other data sources.

initial_world (String)
: The initial world to load after startup.

initial_game_state (String)
: The initial game state to activate after startup. It can be either `Editing` for the editor or `Main` to start the application in game mode directly.

disable_telemetry (Boolean)
: Set this to `true` to disable the collection of anonymous telemetry data.

These additional settings are available in the PRO version of IOLITE:

show_splash_screen (Boolean, PRO only)
Set to `false` to disable the splash screen shown during startup.

disable_pro_features (Boolean, PRO only)
Set to `false` to disable all features of IOLITE PRO.

7.3 Voxel Shapes

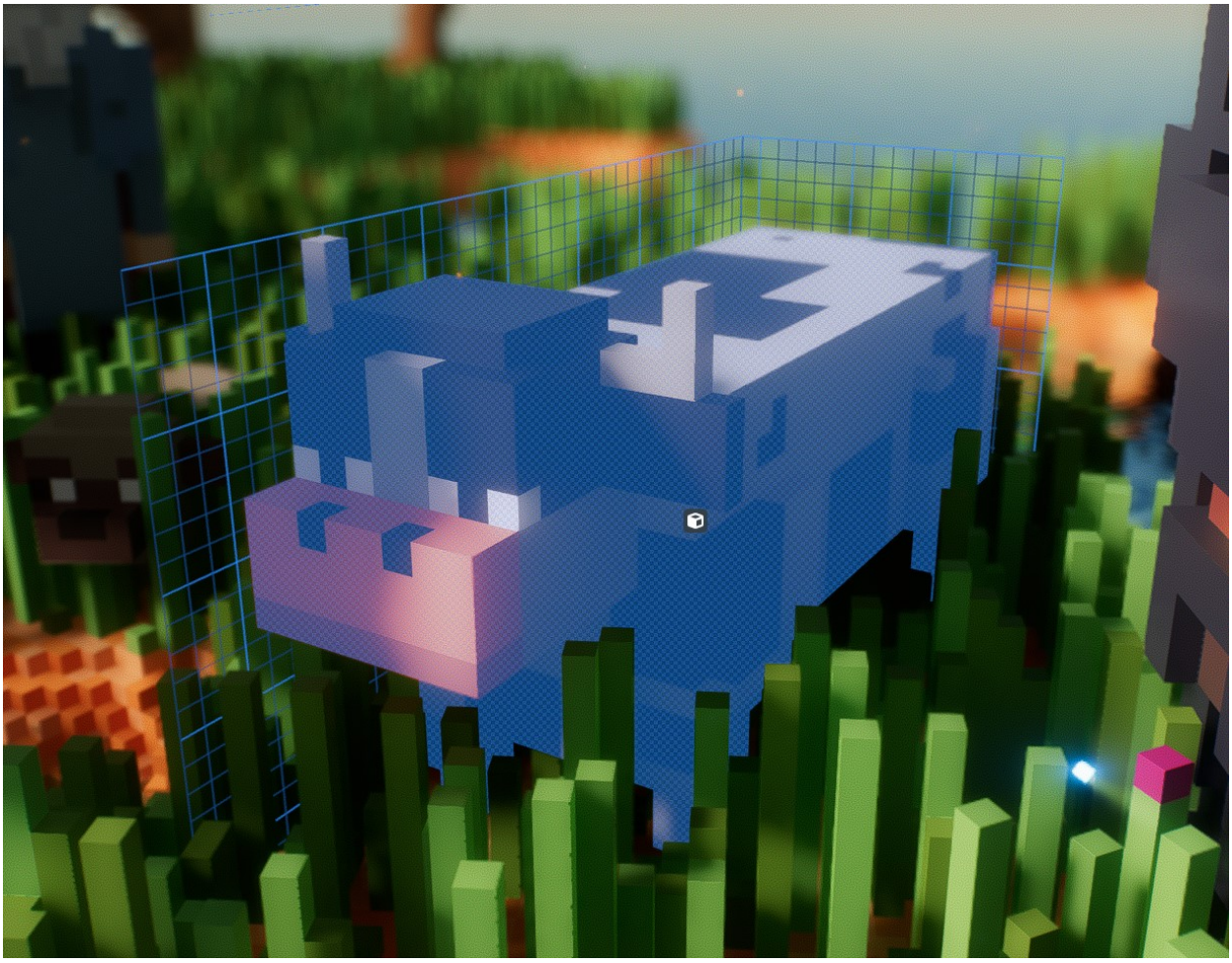
Voxel shapes are the primary tool for building voxel worlds in IOLITE and are exposed in the editor via a component of the same name (see *Voxel Shape*).

Voxel shapes form a three-dimensional grid where each cell stores either a *solid* or an *air* value. Voxels are stored as 8-bit values serving as indices into a color and material palette, allowing you to represent 255 different types plus empty (air) voxels.

- Shapes support storing up to 256^3 voxels
- Each voxel shape can reference a unique color and material palette
- Palette index 0 is reserved for marking empty (air) voxels
- Palette indices 1-255 map to slots 0-254 of the palette

Voxel shapes can either be created procedurally, authored in the editor via the *Voxel Editing* plugin, or imported via the VOX file format generated by voxel editors like Avoyd or MagicaVoxel.

The following screenshot shows a single voxel shape which represents a cow character in the scene:



Since shapes are relatively limited in space, large worlds must be built using compounds of shapes. For procedurally generated terrain, shapes can be used for rendering the world in chunks. This approach is utilized in many games, like, e.g. Minecraft. Depending on the style, characters can be built using more minor compounds of shapes. So, a single shape can be used for the head, one for the torso, and so on. Splitting characters into multiple pieces also allows you to animate the parts of the character separately.

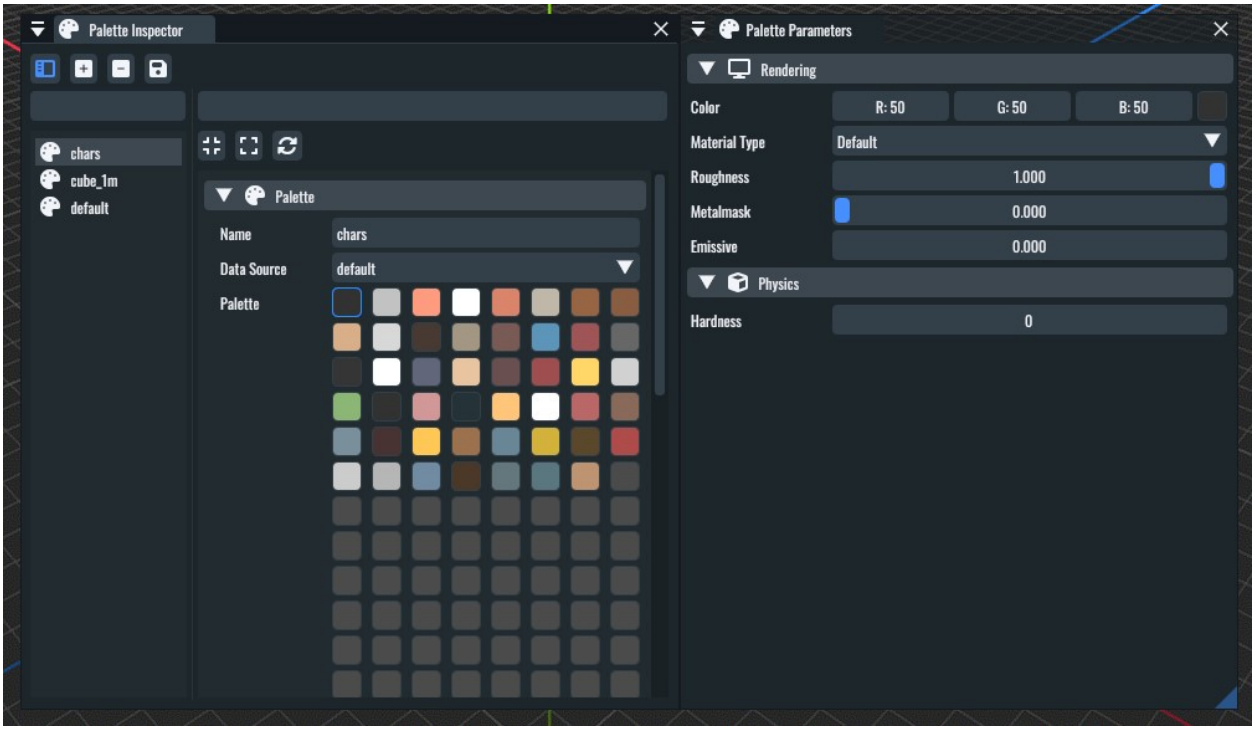
Splitting shapes also has the upside that it allows you to reduce the amount of wasted space to a minimum, which can influence the overall performance and memory usage. This and more is explained in detail in the section *Best practices*.

7.4 Palettes

Palettes are used to assign material parameters to the values of voxel shapes. As mentioned in the previous sections, shapes utilize 8-bit values offering a total of 256 possible unique values. A value of zero is reserved for marking empty/air voxels. Accordingly, a single palette supports storing up to 255 different materials. Each voxel shape can utilize a unique palette, but it is not possible to use multiple palettes for a single shape.

Important: As mentioned previously, due to the fact that a value of zero is reserved for empty/air voxels, the voxel values 1-255 map to the palette indices 0-254.

Palettes can be edited either programmatically or via the *Palette Inspector* in the editor, which is shown in the following screenshot:



Voxel assets also contain palettes which are treated the same as the custom palettes you have created. Asset palettes are named according to their source assets. To overwrite the palette of a voxel asset, simply create a custom palette named like the palette you would like to overwrite. Custom palettes are prioritized during loading.

The following sections describe each of the material parameters a single entry in the palette supports.

7.4.1 Rendering material parameters

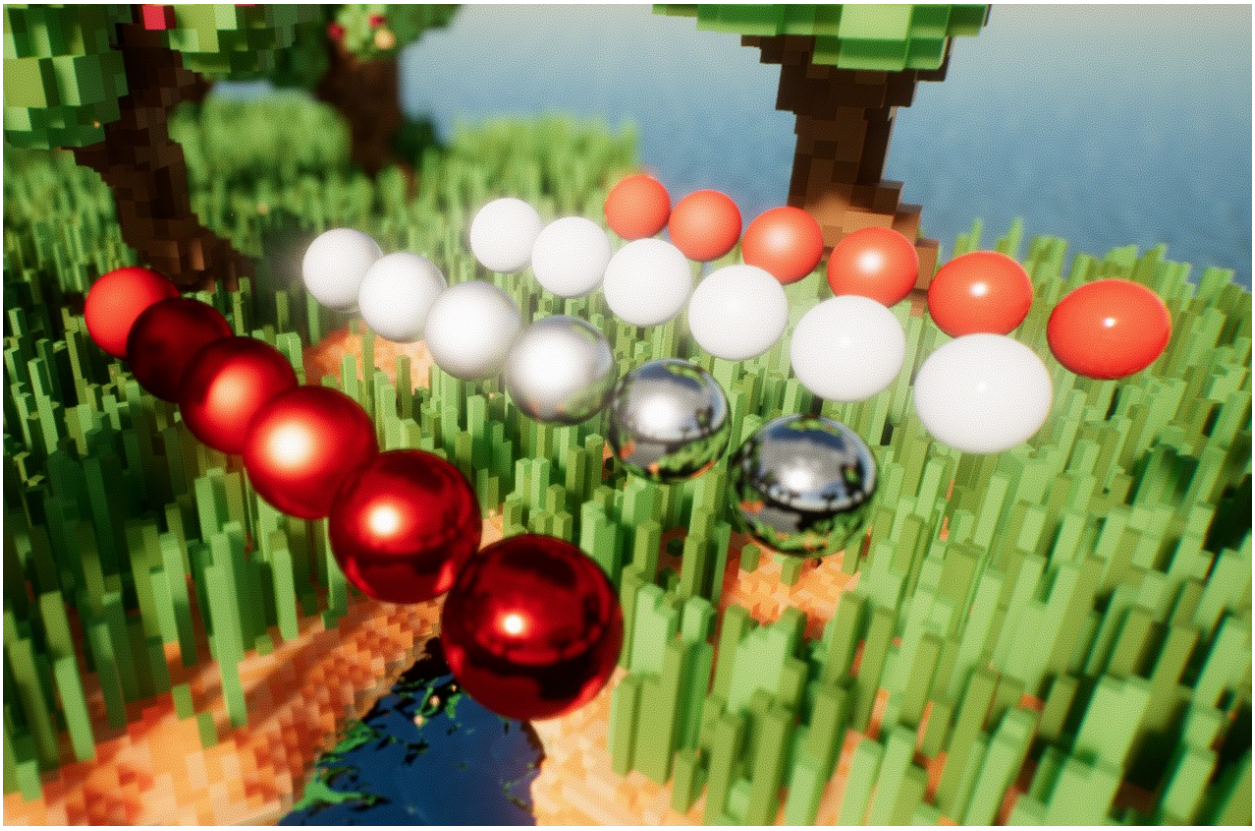
Color
: The albedo color of this material.

Roughness
: Defines the roughness/smoothness of the material.

Matalmask
: Defines whether this material is a nonmetal (value of 0) or metal (value of 1). Values in between can be used for defining hybrid materials like, e.g., rusty metals.

Emissive
: The emissive intensity of this material.

The following screenshot shows the impact of the roughness parameter for a red/white metal and nonmetal:



7.4.2 Physics material parameters

Hardness
: The hardness of the material which is evaluated in the context of the physics-based destruction features.

7.5 Component glossary

This section serves as a glossary for all the available core components.

Note: The properties of the components are directly documented in the editor and can be accessed by hovering over the property name in the entity inspector.

7.5.1 Camera

Cameras are used to render the world from certain positions. Every world contains a `editing_camera`, which is used in the editor, and a `game_camera`, which is used while the game mode is active, by default. It's possible to create and activate additional cameras, either via the editor or via the scripting interface.

7.5.2 Camera Controller

Camera controllers, as the name suggests, control the camera they are attached to. They can be used to create FPS- or third-person-style camera behaviors controllable via the scripting interface with little effort.

7.5.3 Character Controller

Character controllers are used to provide physics-interactions for characters. The component provides a scripting interface for moving the characters.

7.5.4 Custom Data

Custom Data components offer the possibility to store custom properties for an entity. Properties can be read, added, modified, and removed via the editor or the scripting API.

7.5.5 Joint

Joints can be used to connect voxel shapes with varying constraints. The component offers various types of joints to create various physics-based effects.

7.5.6 Kill Plane

Kill planes can be used to automatically destroy voxel shapes which drop below the y-position of the node the component is attached to.

To ensure that the whole shape is below the plane before it is destroyed, the shape's bounding sphere is used. The logic is implemented as follows:

```
if (shape_bounds_center.y + shape_bounds_radius < kill_plane_position.y) {
  despawn_voxel_shape();
}
```

7.5.7 Light

Use this component to add lights to the scene. More details can be found in the chapter *Lighting and GI*.

7.5.8 Mesh

Mesh components are used to render triangle meshes with different materials. They are mainly used to render the sky sphere and water planes with the according materials.

Additional meshes can be provided by adding OBJ files to the `media/meshes` directory in any data source. Currently, only materials named `default`, `water`, or `sky` are supported.

7.5.9 Node

Node components are attached to every entity by default and provide a 3D-transformation, e.g., a position, orientation, and size. They also make it possible to create hierarchies of entities.

7.5.10 Post Effect Volume

Post effect volumes are used to add post effects to certain areas within the world. They are applied hierarchically based on their priority.

7.5.11 Sound

Use this component to play sound effects at certain positions in the world.

7.5.12 Script

This component can be used to add Lua scripts to the world.

7.5.13 Tag

Can be used to add tags to entities. Entities can then be looked up based on the tags via the scripting interface.

7.5.14 Text

Use this component to render text at different positions in the world. It's also useful for adding notes in the editor.

7.5.15 Voxel Shape

Voxel shapes are in charge of rendering and simulating everything voxel-related. Shapes are either initialized from authored VOX files available in the data sources, or they are procedurally filled with content like, e.g., terrain using a script.

In addition, voxel shapes support the generation of support structures and fracturing for physics-based destruction effects.

7.5.16 Flipbook Animation

Use this component to animate voxel shapes using flipbook-style animations. Flipbook animations operate by changing the voxel data of the corresponding shape component on a frame by frame basis.

7.6 Rendering overview

This chapter depicts the rendering approach utilized for rendering IOLITE's voxel worlds. While you won't need every detail in this section, it'll be useful to have some background information for optimizing your project later.

Note: Coming soon.

7.7 Lighting and GI

Note: Coming soon.

7.8 Physics

This chapter serves as a guide for all the physics-based features available.

7.8.1 Destruction

This section discusses the destruction subsystem.

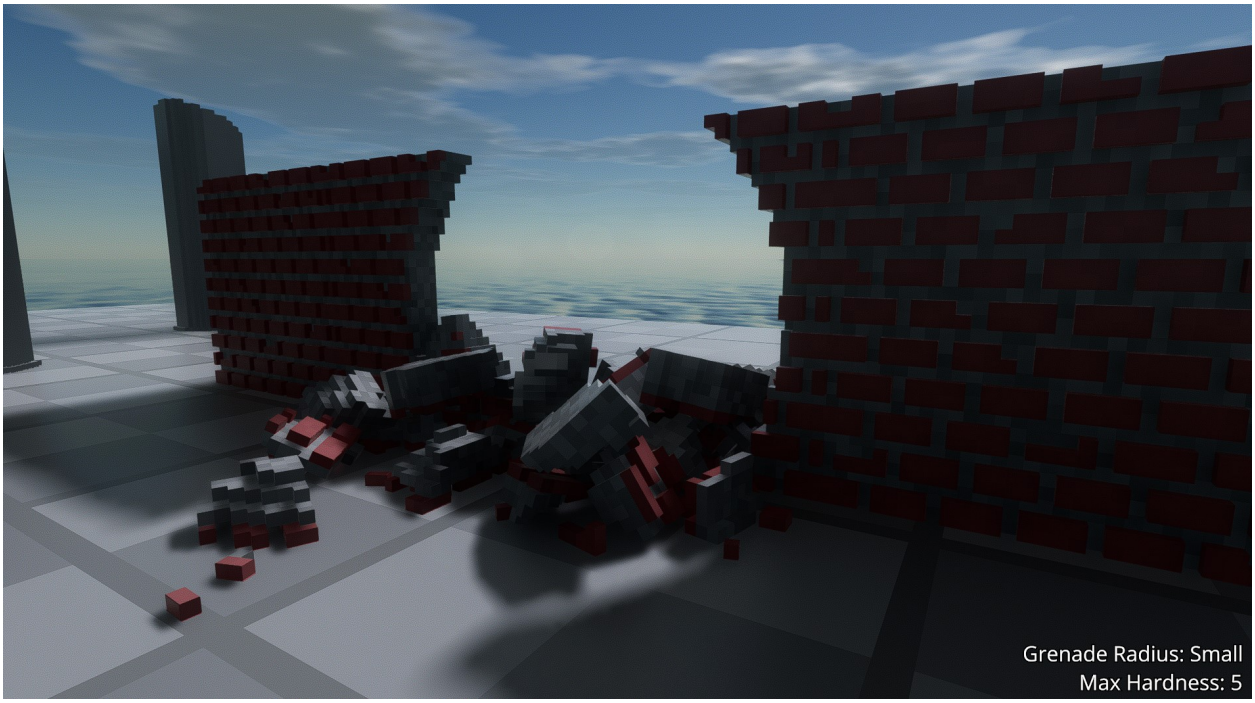
Support structures

To enable the destruction features for a shape, set the `Support Structures` *voxel shape flag* in the component. With this feature enabled, every time a shape is voxelized, the system scans the shape for islands/regions, classifying them using the adjacency of voxels. It then automatically creates new sub-shapes for the regions based on the result. The algorithm uses a 6-connectivity (the six faces of each voxel) for classifying the regions.

The system also scans for regions connected to the ground for static shapes. A region is treated as having a ground connection when it contains a voxel in the bottom row of the shape (the y coordinate of the voxels equals zero). The resulting shape is kept static if a region has a ground connection. If a region loses its ground connection, the resulting shape will be set to dynamic.

Fracturing

The following screenshot shows a wall split into chunks using IOLITE's fracturing system.



Note: To see the fracturing system in action, check out the *physics sample* in our repository.

Voxel connectivity

Shapes can also be split based on the *voxel connectivity system*. This system allows you to specify the connectivity for each of the six faces of a voxel. This system, e.g., eases the implementation of scenarios where the player can cut shapes into pieces.

For example, if you want to cut a wall horizontally into two equally large pieces, call the `disconnect` voxel shape API function on all voxels in the center row, passing the flag for the top face. Doing this will disconnect all voxels from their top neighbors. The wall will be cut in half after calling `voxelize` for the shape.

7.9 Particle system

Note: Coming soon.

7.10 Post-processing system

Note: Coming soon.

7.11 Sound system

The sound system allows you to play sounds and music.

Note: The sound system currently does not feature a user interface and has to be configured by manually authoring JSON files.

7.11.1 Authoring audio files

The sound system supports **stereo** OGG and WAV audio files with a **fixed sample rate of 44100 Hz**. The sound system searches for audio files in the `media/sounds` directory in any of the available data sources.

Note: Audacity is a great free audio editor, which can be used for creating audio files in the right format.

Warning: Please make sure to use the right sample rate and number channels. Neither mono nor audio files with a different sample rate than 44100 Hz are supported.

7.11.2 Preparing sound effects

After preparing your audio files, you need to create the appropriate sound effects. A single sound effect comprises one or multiple audio samplers, which reference the actual audio files on disk.

To create a sound effect, create a `my_sound_effect.effect.json` file in the `media/sounds` directory in a data source of your choice, right next to your audio files. The effect file has the following format:

```
{
  "my_sound_effect": {
    // The member parameters of your effect as
    // as depicted below
    "audio_samplers": [
        {
          // Parameters for a single audio sampler as
          // depicted below
        }
    ]
  }
}
```

A single effect supports the following parameters:

audio_samplers (Array of audio samplers)
The audio samplers for this effect. A single effect can play multiple samplers at the same time.

volume (Float, optional)
The volume for this sound effect. Use a value of `0.0` for a muted and a value of `1.0` for the loudest sound effect.

looping (Boolean, optional)
Set to `true` to automatically restart the effect instance when it has finished playing. Great for background music or ambient effects.

spatial (Boolean, optional)
Set to `true` for spatial/3D sound effects. Spatial effects have a position in 3D space and change their volume depending on the listener's position.

spatial_size (Float, optional)
The spatial size of the effect. If the distance to the listener is less than this value, the effect fades into a non-spatial sound effect.

dist_att_min (Float, optional)
The minimum distance attenuation volume. Sound effects will never get quieter than this volume value when attenuated based on the distance to the listener.

dist_att_power (Float, optional)
Specifies how quickly a sound effect is attenuated based on the distance to the listener.

dist_att_range (Float, optional)
The range world units in which the distance attenuation operates. Effects out of this range will be attenuated to the volume specified by the `dist_att_min` parameter.

Audio samplers support the following parameters:

file_names (Array of strings)
List of file names, including the file extension, of the audio files to play for this sampler. If more than one file name is provided, a random file is picked every time the effect is restarted.

volume (Float, optional)
The volume for this sampler. Behaves the same as the global volume for the effect.

timeline_offset (Float, optional)
The position on the effect's timeline in seconds when this sampler should start playing.

pitch (Array of floats, optional)
Randomly alters the pitch in the provided interval. Requires exactly two values in the array, which can be positive or negative. The first represents the minimum, and the second the maximum relative pitch change in cents.

7.11.3 Playing sound effects

Sound effects can either be played via the Lua scripting interface, the C API, or by placing a *Sound Component* in the scene.

7.12 UI system

Note: Coming soon.

7.13 Best practices

Note: Coming soon.

CHAPTER

EIGHT

EDITOR DOCUMENTATION

This documentation serves as the reference guide for working with IOLITE's editor.

8.1 Basic usage of the editor

This section introduces the editor and its functionality.

Compared to other game engines, the editor is integrated into the runtime. An integrated editor has the tremendous advantage of being usable at any time to, e.g., directly inspect gameplay scenarios while testing your game. It also allows shipping the same toolset for modding purposes without additional effort.

8.1.1 Moving and rotating the camera

By default, the editor starts with the *Free Camera* enabled.

The *Free Camera* can be moved forward, left, back, and right relative to the current viewing direction using the `[W]`, `[A]`, `[S]` and `[D]` keys. It can be moved up and down using `[E]` and `[Q]`. To rotate the camera, right-click, hold, and drag in the viewport.

In addition, IOLITE supports an *Orbit Camera*. To activate it, click `[View]` in the main menu bar and select `[Orbit Camera]` from the dropdown list under *Camera*.

With the *Orbit Camera* active, right-click, hold, and drag in the viewport to rotate the camera around the current center position. To move the camera, center-click, hold, and drag. To zoom in and out, use the mouse wheel.

8.1.2 Running your game

The engine core utilizes the concepts of game states to provide different isolated feature sets to its users.

There are two of them available:

- While pausing some underlying subsystems like, e.g., the physics simulation and scripts, the *editing game state* houses IOLITE's editor.
- In contrast, the *main game state* ticks all gameplay-related subsystems and provides no editing functionality. This is the state that is active when you want to test or, later on, ship your game.

Important: When launching IOLITE in its default configuration, the application starts in the *editing game state*. How to make the *main game state* the default after launch is described in *Working with the app metadata file*.

To switch to the *main game state* in the editor, click on `[Game Mode]` in the menubar of the editor. When clicking this button, IOLITE stores a snapshot of the current world, switches the active camera from the `editing_camera` to the `game_camera`, and activate the *main game state*. By pressing `[F3]` while the *main game state* is active, the *main game state* gets deactivated, the previous state of the world gets restored, and the editor becomes active again.

8.1.3 Working with snapshots

When entering the *main game state*, IOLITE automatically generates a snapshot of the current state of the world. When returning to the editor, the previous state is automatically restored, discarding all changes that occurred while the *main game state* was active.

It might be helpful to skip restoring the state, e.g., to inspect a particular gameplay scenario. Accordingly, it is possible to skip restoring the state automatically by holding `[CTRL]` while clicking on the `[Game Mode]` item in the main menu bar. Back in the editor, the snapshot can then be manually restored by clicking on the *red load icon*.

In addition, it is possible to create and restore snapshots manually. To create a snapshot, click on the *green save icon*. Snapshots are stored as a stack, making it possible to store multiple snapshots and restore them in the exact order they were created. To restore the latest snapshot, click in the *red load icon*, as mentioned before.

8.1.4 Working with entities and components

As teased in *Entity Component System*, entities and components form the basis for creating games with IOLITE. This section covers some basics, like, e.g., creating and removing entities in the editor, how to clone entities, and how entities can be decorated with components.

Selecting entities

Select entities either by clicking on them in the *World Inspector* or visually by clicking on them in the viewport. Visual picking only works if the selection tool is active in the editing toolbar.

Moving and rotating entities

Select the entity you want to transform and select either the transform, rotate, or the combined gizmo from the editing toolbar. Click, hold, and drag the gizmo handles to apply the desired transformation.

Creating and removing entities

To create a new entity in the editor, right-click on the entity in the *World Inspector* you want to attach the new entity to. In the context menu, either select `[Attach New Entity]` or `[Attach New Entity (With Component)]` to create an entity decorated with the selected component from the start.

Remove entities by selecting `[Remove Entity]` from the context menu or pressing `[Del]` after selecting it in the *World Inspector*.

Adding and removing components

To add a component to an existing entity, right-click on it in the *World Inspector*, select `[Add Component]` in the context menu, and select the component you'd like to add.

To remove a component, select `[Remove Component]` in the context menu, and select the name of the component you want to remove.

Cloning entities

Clone entities by selecting them and choosing `[Clone Entity]` from the context menu. It's also possible to clone entities by pressing `[Ctrl + D]`.

Important: Cloning entities also copies the whole node hierarchy and all the attached components.

Renaming entities

Select the entity you want to rename and select `[Rename Entity]` from the context menu. Enter the new name in the text field and finalize the operation by clicking `[Rename]`.

Hiding entities

It's possible to hide entities in the scene by clicking on the eye icon shown right next to them in the World Inspector. Hidden entities are tagged with a crossed-out eye icon. Hiding an entity also implicitly hides all of its descendants. Entities that are hidden implicitly by one of their ancestors are shown with a greyed-out eye icon or a grayed-out crossed one if they're also hidden explicitly.

Parenting entities

Changing the hierarchy of entities is possible by dragging and dropping in the *World Inspector*. To parent an entity with another one, drag and drop it onto the desired parent entity. To unparent an entity, drag it onto the root entity of the world.

8.1.5 Editing worlds

This section introduces various useful features for quickly editing worlds in IOLITE.

Creating, saving, and loading worlds

- To create a new world, open up the `[File]` menu and select `[Create New World]`.
- To load an existing world, hover over `[Load World]` in the `[File]` menu and select the desired world.
- To save the current world, select `[File] => [Save World]`.
- To save the current world *to a separate file*, select `[File] => [Save World As...]`
- To reload the current world from the disk, select `[File] => [Reload World]`

Adding voxel shapes to the world

To quickly position a voxel shape in the world, head over to the *Prefab Browser*. Click and hold the desired shape and drag it into the viewport. Release the mouse button when you're happy with the position.

Another option is to manually create an entity with an attached *Voxel Shape* component and pick the desired shape via the component's properties.

Saving and loading prefabs

Prefabs are hierarchies of entities that can be saved to and loaded from disk. They can either be used to ease editing or to create modules, like, e.g., a character that gets spawned using a script.

To create a prefab, select the root entity in the *World Inspector* and click on `[Save as Prefab]` in the context menu. Choose a name in the dialog and confirm by clicking `[Save]`.

To load a prefab via the editor, head over to the *Prefab Browser*, click and hold the desired prefab and drag it into the viewport. Release the mouse button when you're happy with the spawn position.

Randomized shape and prefab placement

To quickly place variations of prefabs and voxel shapes with randomized size and rotation parameters, head over to the *Prefab Browser* and click the dice icon. In the *Prefab Randomization* window, set the desired randomization intervals for the size and orientation.

When done, go ahead and place prefabs and shapes as usual in the scene using the *Prefab Browser*. With each spawned shape/prefab, the randomized orientation and size is applied to the resulting root node.

Snapping entities to the ground

To ease the placement of entities in the scene, it's possible to snap them to the ground. Snapping internally uses a raycast in combination with the extents of the shape.

To snap an entity and its hierarchy to the ground, either click on `[Edit] => [Snap to Ground]` or press `[V]`.

Centering voxel shapes

Voxel shapes have their origin the lower left corner of their grid. To center a voxel shape, select it and either press `[C]` or click on `[Edit] => Center Entity`. This operation updates the pivot of the shape to the center of its extents.

Moving an entity to the current cursor position

Entities can be moved to the current position of the mouse cursor by pressing `[R]`.

Switching the active camera

When working with the editor, the *editing_camera* is activated by default. To switch the camera, right-click on the desired camera in the *World Inspector* and select `[Camera Actions] => [Activate Camera]` to activate it.

Copying camera transforms

Sometimes, it's desirable to copy one camera's transform to another quickly. There are multiple options to achieve this:

- Select the camera from which you want to copy the transform in the *World Inspector*. Then right-click on the camera you want to copy the transform to and select `[Camera Actions] => [Copy Transform from Selected]`
- Activate the camera you want to copy the transform from and orient and translate it as desired. Then right-click on the camera you want to copy the transform to in the *World Inspector* and select `[Camera Actions] => [Copy Transfrom from Active]`. This is especially useful if you want to, e.g., copy the current transform of the `editing_camera` to the `game_camera`.

8.1.6 Working with the console

The console can be used to show the current log output during runtime and to interact with the settings. Its available while the game is running and in the editor. To open it up, press `[F2]`. To execute a command, enter it into the console window and press `[Enter]`.

The following commands are available:

ls
: Lists all settings and their values

load_settings
: Loads and applies all settings from disk (engine and user settings)

save_user_settings
: Writes all settings to the user's settings file. Only settings which differ from their default value are written

<setting_name>
: Prints the current value of the setting with the given name to the console

<setting_name> <value>
: Sets the setting with the given name to the provided value. Settings support values as unsigned integers, floats, boolean values, and strings

8.2 Importing assets

This section briefly the describes the process of importing various asset types.

8.2.1 Importing textures

IOLITE requires textures to be in the DDS file format. The command line tool `texconv.exe` from DirectXTex is an excellent option for converting various image formats to DDS.

Note: If you're using Linux, it's possible to run the tool using Wine, using `wine texconv.exe ...`

All textures are sourced from `media/images` in each data source directory and can be accessed during runtime via the filename (**without** the extension).

Important: When generating textures for use in IOLITE's UI system, it is important to export them with pre-multiplied alpha enabled. Here's an example of how to correctly convert a UI texture using DirectXTex to the BC7 texture format:

```
texconv.exe -pmalpha -f BC7_UNORM_SRGB my_texture.png
```

8.2.2 Importing VOX files

If you want to add a new voxel asset to, e.g., the default data source, place the file in `default/media/voxels`. Please make sure to use the correct `.vox` file extension. If IOLITE is running while adding new voxel assets, you can trigger a hot-reload by executing `[Tools] => [Reload Voxel Shapes]` in the editor. This makes the new assets available without restarting the engine.

8.2.3 Importing scenes from VOX files

By default, IOLITE scans all the VOX files available in the data sources and makes each found model available as a separate asset. In addition, VOX files also support storing scene information which can then be imported. This is useful for quickly importing scenes that have been authored in external tools like MagicaVoxel or Avoyd.

To import a scene from an available VOX file, open the `[Import and Export]` menu, hover `[Import Scene]`, and finally select the VOX file to import the scene from. The scene hierarchy is added to the world as a new entity named according to the source asset.

8.2.4 Importing palettes

Palettes can either be imported by adding VOX files to your project or by importing them separately from image files downloaded from sites such as Lospec or authored in image editing tools like *Photoshop*. This is especially useful if you want to author your voxel assets directly in IOLITE, e.g., using the *Voxel Editing* plugin.

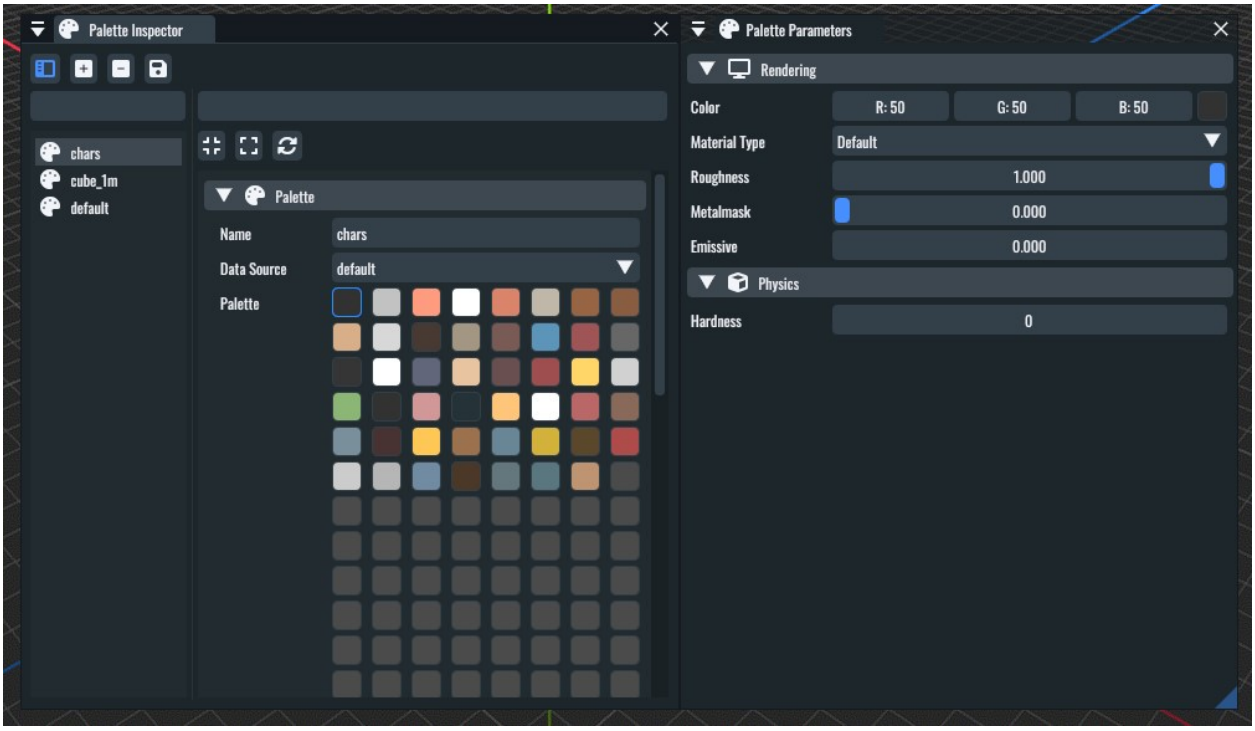
Important: Each pixel in the palette image is interpreted as a color of the palette. The pixels are read row by row starting from the top left corner until the maximum of 255 palette colors is reached.

To import a palette from an image, select `[Import and Export] => [Import Palette]` and choose the desired image file from your file system. Palettes can be inspected and edited via the *Palette Inspector* available via the `[View]` menu. See *Palette Inspector* for more details.

8.3 Editing resources via Inspectors

IOLITE's exposes some of its resources for editing and inspection via so-called *Inspectors*.

8.3.1 Palette Inspector



The palette inspector can be used to create, inspect and modify palettes. Besides the palettes you have created manually, either via the editor or code, it also shows all the palettes that have been sourced from your voxel assets.

The general functionality of palettes is described in detail in *Palettes*.

8.3.2 Post Effect Inspector

> **Warning:** Coming soon.

8.3.3 Particle Inspector

> **Warning:** Coming soon.

8.4 IOLITE PRO specific features

This section is dedicated to the additional features IOLITE PRO provides.

8.4.1 Disabling the splash screen

To disable the splash screen, add the following member to your `app_metadata.json` file:

```
"show_splash_screen": false
```

8.4.2 Creating voxel assets from meshes (Mesh Voxelizer)

The mesh voxelizer allows you to create voxel assets from complex 3D meshes.

To voxelize a mesh, export a mesh from your favorite 3D authoring software, like *Blender* or *3ds Max*, in the *glTF* format. After exporting, the meshes, head over to IOLITE's editor and open up the `[Import and Export]` menu in the toolbar and hover and click on `[Voxelize Mesh]`.

8.4.3 Exporting path-traced renders

To export your path-traced renders, head over to the `[Render]` menu in the menu bar and select `[Export Render]`. With the PRO version, you also have the option to enable *raw color output*, which turns off the display transform, and to select the lossless *EXR* file format which is great for handling post-processing in external tools like *Blender* or *Darktable*.

CHAPTER

NINE

IOLITE PLUGINS

This section serves as a reference for IOLITE's plugin system. In this section, you will learn how to load and work with the available open-source plugins and how to start writing your own native plugins using the C/C++ API.

9.1 Working with plugins

This section serves as a brief introduction how plugins are handled in IOLITE and how to install or compile IOLITE's open-source plugins.

9.1.1 Loading plugins

The compiled plugin libraries (DLL or SO files) have to be placed in IOLITE's root directory, right next to the executable.

In addition, the engine sources the plugins which should be loaded from the optional `plugins.json` file in the root directory. The file requires the following structure:

```
[
  {
    "name": "my_plugin",
    "filename": "MyPlugin"
  },
  {
    "name": "another_plugin",
    // ...
  }
]
```

The following members for configuring a plugin are available:

name (String)
: A descriptive name for your plugin.

filename (String)
: The filename of the library *without* the extension.

load_on_startup (Boolean, optional)
: Set this to "false" to require loading the plugin manually.

checksum (Unsigned integer, optional, internal)
: Used by the free version of IOLITE to whitelist certain plugins.

9.1.2 Installing the open-source plugins

The plugins documented in this chapter are available as open-source plugins in our public GitHub repository. The C/C++ sources of the plugins can be found here:

https://github.com/MissingDeadlines/iolite/tree/main/iolite_plugins

Even though compiling the plugins is straightforward, the plugins are also available as binaries for Windows and Linux, made available using GitHub releases. You can download a release matching your version of IOLITE here:

https://github.com/MissingDeadlines/iolite/releases

To load all the available, simply extract the contents of the `windows` or `linux` subdirectories in the `plugins.zip` file in the root directory of IOLITE, and you are good to go.

Important: There is no need to create the `plugins.json` file manually. Each release ships with a `plugins.json` file containing all the available open-source plugins.

Warning: The plugin libraries have to be placed **right next** to the executable. Placing them in a subdirectory **will not** work.

9.1.3 Compiling the sample and open-source plugins

Building the plugins requires a C++ compiler and a recent version of CMake.

The plugins can be built on any supported platform by either running *build_plugins.bat/sh* or by manually executing the following commands on the command line:

```
$ cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
$ cmake --build build --config Release
```

9.2 Writing plugins using the native C/C++ API

IOLITE's C/C++ API offers the possibility to create highly modular engine and editor plugins in C, C++, or any other language that supports C bindings.

Important: The API documentation is provided via the header file itself.

Important: Native plugins are a feature of IOLITE PRO. You're free to evaluate plugins in the free version, but loading non-default plugins will add a watermark to the editor and game.

9.2.1 Writing a simple plugin

To write your first plugin, head over to our GitHub repository and download the latest version of the API header file.

The header file is completely self-contained and should compile with any C or C++ compiler on any of the supported platforms. Simply set up a project that outputs a DLL and you are good to go. If you're unsure how to proceed, take a look at the minimal sample plugin which shows how to use CMake to set up a plugin project for multiple platforms.

All a plugin has to do is provide the following three functions, as shown in the minimal plugin example in the repository:

```
#include "iolite_api.h"

const struct io_api_manager_i* io_api_manager = 0;

IO_API_EXPORT io_uint32_t IO_API_CALL get_api_version()
{
  // Inform IOLITE which version of the API you are using
  return IO_API_VERSION;
}

IO_API_EXPORT io_int32_t IO_API_CALL load_plugin(const void* api_manager)
{
  // Ensure we can keep accessing the API manager after loading the plugin
  io_api_manager = (const struct io_api_manager_i*)api_manager;

  // Do something with the API manager, set up your plugin, etc.

  return 0; // Return a value < 0 to indicate that the loading of your plugin
            // has failed (depedency not available, etc.)
}

IO_API_EXPORT void IO_API_CALL unload_plugin()
{
  // Clean up here
}
```

After compiling your plugin, create/modify the `plugins.json` file in the root directory of the application. The structure of the file is depicted in the section *Loading plugins*.

If everything worked out okay, you can fire up IOLITE and check the results via the console log output (`[F2]` opens up the console) or the `[Plugin Manager]` available via the `[Plugins]` menu in the menu bar.

9.2.2 Writing a custom scripting backend

IOLITE's Lua integration is provided via a native plugin using the C API, and the full implementation is available via our public GitHub repository. The Lua implementation can be a great starting point if you plan to roll your own custom scripting backend.

9.3 Lua Scripting

IOLITE uses Lua to provide its scripting functionality. Lua is an excellent language for scripting due to its minimal footprint, efficiency, and ease of use.

If you have not worked with Lua before, make sure to check out the following resources to learn the language:

Programming in Lua (First Edition)
https://www.lua.org/pil/contents.html

The Lua 5.1 Reference Manual
https://www.lua.org/manual/5.1/

Important: This section serves as an introduction to the Lua scripting integration in IOLITE. For further details, make sure to check out the *Lua API* documentation to quickly locate the functionality you are looking for.

9.3.1 Lua runtime and libraries

IOLITE uses `LuaJIT-2.1.0-beta3` for maximum performance, which supports the language features of Lua 5.1. The following standard Lua libraries are available:

- base, coroutine, string, and table

Check out the following resources if you're interested in the features and implications induced by using the LuaJIT runtime:

LuaJIT Homepage
https://luajit.org

LuaJIT Overview
https://luajit.org/luajit.html

LuaJIT Benchmarks
https://luajit.org/performance_x86.html

9.3.2 The basic structure of scripts

When using scripts, IOLITE expects you to provide a particular set of callback functions with the correct naming and parameters, which it can call in different scenarios.

Important: Please note that you do not need to provide all callback functions, even not providing any callback function is fine. IOLITE checks for the availability of each upfront.

The `minimal.lua` script available in the `default` data source contains stubs for all the available callback functions and can serve as a template for creating new scripts.

Let's have a closer look at each of the available callback functions.

9.3.3 Available callback functions

```
function OnActivate(entity)
end
```

Called precisely once during a script's lifetime.

This function is called once when the script becomes active. Scripts become active when, e.g., a world is loaded or if an entity with an attached script component gets spawned. This is the right place to set up additional resources and variables for your script.

```
function OnDeactivate(entity)
end
```

Called precisely once during a script's lifetime.

This is the counterpart to `onActivate` and is called once when the script becomes active, either by unloading a world or by destroying a script component during runtime. Use this to tear down additional resources created by your script.

```
function Tick(entity, delta_t)
end
```

Called precisely once each rendered frame.

Use this function for functionality that has a visual effect, like updating the final position of a character or a projectile, for example. It's also the right spot to react to the user's input as quickly as possible.

In general, it's wise to keep the workload in this function to a minimum and, e.g., implement actual gameplay and AI logic in the `OnUpdate` callback function at a lower frequency. The results computed at the lower frequency can then be interpolated in this function to achieve visually pleasing results.

```
function Update(entity, delta_t)
end
```

Called exactly once at the interval specified in the script component.

Use this callback for implementing logic that has no imminent visual effect. This is the perfect spot for implementing AI and gameplay logic.

Important: Don't use this function for reacting on input or for updating data that has a visual effect!

```
function OnEvent(entity, events)
end
```

Called as soon as one or multiple events are available.

All the different types of available events are described in a later section. But the grasp the general concept, here's an example of handling contact events that occur when voxel shapes, and their rigid bodies, interact with each other:

```
function OnEvent(entity, events)
    -- Iterate over all the available events
    for i = 1, #events do
      local e = events[i]
      -- Handle contact events
      if e.type == "Contact" then
```

(continues on next page)

(continued from previous page)

```
        -- Provides the position of the contact
        -- "e.data.pos", the resulting impulse "e.data.impulse",
        -- and the interacting entities "e.data.entity0"
        -- and "e.data.entity1"
      end
    end
```

Last but not least, a variation of the `Tick` callback function:

```
function TickAsync(entity, delta_t)
end
```

Called precisely once during each rendered frame but executed asynchronously till the next call to this function.

Use this function to optimize scripts that need to do some complex and costly calculations. Check out the heightmap sample in our GitHub repository, which uses this functionality.

Important: It's only safe to do some basic calculations here and to modify the internal state of the current script. Accessing entities and components via the scripting API will most certainly lead to crashes or very hard to reproduce bugs. **Use with absolute caution!**

9.3.4 Loading API interfaces

IOLITE provides a lot of different API interfaces for all the available subsystems. To ensure that scripts have a minimal footprint, you have to explicitly state which interfaces you want to use at the beginning of your script.

As an example, if you want to work with nodes and print some text to the log/console, you'll have to load the `Log` and `Node` interface tables like this:

```
Node.load()
Log.load()
```

In this example, the calls to `load()` populate the functions provided by the interfaces `Node` and `Log` via the according global tables.

Please note that not loading the API interfaces will lead to errors stating that the requested function is unavailable.

9.3.5 Hot reloading and error logging

Scripts are hot-reloaded on every change you make. Potential errors and your log calls end up in IOLITE's console and log file. To toggle the console, press `[F2]`.

If executing the script throws an error, go ahead and adjust the faulty line of code, save the file, and directly check back in IOLITE if the error is gone. It's as easy as that.

9.3.6 Date structures and refs

When interacting with IOLITE via the scripting interface, you'll encounter three different types of data structures:

PODs (Plain Old Data)
: Vectors provided by the math interface, etc.

Refs
: Used to reference entities, components, and resources on engine-side

Handles
: Like refs, but specific to certain subsystems, like, e.g., the particle or sound system

Refs, compared to handles, are agnostic of the underlying subsystems. A ref can reference any component, entity, or resource, providing interfaces for checking the underlying type and whether the referenced resource is still alive.

Let's look at some examples of how refs can be utilized in detail. Here we're searching for a specific entity in the scene and checking whether it's available:

```
Entity.load()

-- Try to find the "goose" entity in the world
local goose = Entity.find_first_entity_with_name("goose")
if Ref.is_valid(goose) then
  -- Do something to the goose...
end
```

Now we're dealing with a ref of unknown origin, and we want to make sure it is (A) a node and (B) still alive:

```
Node.load()

-- Check if a given ref is referencing a node component
-- and whether the component is still alive
if Node.get_type_id() == Ref.get_type_id(my_potential_node)
    and Node.is_alive(my_potential_node) then
  -- Retrieve the position when we're safe
  local pos = Node.get_world_position(my_potential_node)
  -- Do something with the position...
end
```

9.3.7 Error handling and scripts

IOLITE strives for a good mixture of error handling and performance.

While a lot of user errors won't make the engine crash, like, e.g., passing the wrong amount of parameters to a function, there are certain cases where this behavior is expected, mostly related to interacting with resources and refs:

- Using the ref on an entity, component, or resource which is no longer alive. Make sure to only interact with alive resources using the `is_alive` function of the corresponding interface table
- Using an invalid ref to execute functions. Ensure you're always using valid refs using `Ref.is_valid(ref_in_requestion)`

9.3.8 Going further

Our GitHub repository houses a couple of Lua-based samples which serve as an excellent reference and starting point. Otherwise, header over to the *Lua API* documentation to quickly locate the functionality you are looking for.

9.4 Voxel Editing

This section serves as the reference guide for the *Voxel Editing Plugin* which can be used to create voxel assets directly in IOLITE's editor.

Note: Coming soon.

9.5 Terrain Generator

This section serves as a reference guide for the *Terrain Generator Plugin* which is a simple helper to generate voxel shapes from heightmap images.

Note: Coming soon.

9.6 Other Plugins

This section serves as the reference guide for the *Voxel Editing Plugin* which can be used to create voxel assets directly in IOLITE's editor.

9.6.1 OIDN Denoiser

The *OIDN Denoiser* plugin provides a powerful denoiser for IOLITE's offline GPU path tracer based on Intel's Open Image Denoise library.

CHAPTER

TEN

API DOCUMENTATION

This section provides detailed reference guides for the different APIs provided by IOLITE.

10.1 Lua API

This section is a reference guide for the Lua scripting interface in IOLITE.

Note: The Lua API in IOLITE is provided via a native C++ plugin and is freely available in our GitHub repository. Have a look at the plugin to inspect the underlying implementation of the exposed functions or to customize/extend the implementation.

10.1.1 IOLITE Lua API header file

The IOLITE Lua API header file can be used to add auto-completion to the code editor of your choice. This has been tested in Visual Studio Code using the latest version of the Lua extension.

Click the following link to download the latest version of the header file and drop it somewhere close to your scripts:

- `iolite_api.lua`

10.1.2 Utils

`Utils.`**`execute`**`(`*`script`*`)`

Params

- **script** (`string`) - The Lua script source to execute.

Returns

- **value** (`any`) - The result of the executed script.

Executes the provided string as a Lua script.

`Utils.`**`load`**`(`*`script_name`*`)`

Params

- **script_name** (`string`) - The name of the script to load as a module (without the file extension).

Returns

- **value** (any) - The result of the executed script.

Executes the script with the given name.

`Utils.require(script_name)`

Params

- **script_name** (string) - The name of the script to execute (without the file extension).

Returns

- **value** (any) - The result of the executed script.

Executes the script with the given name. Executed once for each script. Successive calls return the cached script result.

10.1.3 Globals

`InvalidRef()`

Returns

- **value** (*Ref*) - The invalid ref.

Creates an invalid ref.

`Vec2(x)`

Params

- **x** (number) - The scalar value to set the components to.

Returns

- **value** (*Vec2*) - The new vector.

Initializes a new Vec2.

`Vec2(vec)`

Params

- **vec** (*Vec2*) - The vector to copy from.

Returns

- **value** (*Vec2*) - The new vector.

Initializes a new Vec2.

`Vec2(x, y)`

Params

- **x** (number) - First component.
- **y** (number) - Second component.

Returns

- **value** (*Vec2*) - The new vector.

Initializes a new Vec2.

UVec2(*x*)

Params

- **x** (number) - The scalar value to set the components to.

Returns

- **value** (*UVec2*) - The new vector.

Initializes a new UVec2.

UVec2(*vec*)

Params

- **vec** (*UVec2*) - The vector to copy from.

Returns

- **value** (*UVec2*) - The new vector.

Initializes a new UVec2.

UVec2(*x*, *y*)

Params

- **x** (number) - First component.
- **y** (number) - Second component.

Returns

- **value** (*UVec2*) - The new vector.

Initializes a new UVec2.

IVec2(*x*)

Params

- **x** (number) - The scalar value to set the components to.

Returns

- **value** (*IVec2*) - The new vector.

Initializes a new IVec2.

IVec2(*vec*)

Params

- **vec** (*IVec2*) - The vector to copy from.

Returns

- **value** (*IVec2*) - The new vector.

Initializes a new IVec2.

IVec2(*x*, *y*)

Params

- **x** (number) - First component.
- **y** (number) - Second component.

Returns

- **value** (*IVec2*) - The new vector.

Initializes a new IVec2.

`Vec3`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*Vec3*) - The new vector.

Initializes a new Vec3.

`Vec3`(*vec*)

Params

- **vec** (*Vec3*) - The vector to copy from.

Returns

- **value** (*Vec3*) - The new vector.

Initializes a new Vec3.

`Vec3`(*x*, *y*, *z*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.

Returns

- **value** (*Vec3*) - The new vector.

Initializes a new Vec3.

`UVec3`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*UVec3*) - The new vector.

Initializes a new UVec3.

`UVec3`(*vec*)

Params

- **vec** (*UVec3*) - The vector to copy from.

Returns

- **value** (*UVec3*) - The new vector.

Initializes a new UVec3.

`UVec3`(*x*, *y*, *z*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.

Returns

- **value** (*UVec3*) - The new vector.

Initializes a new UVec3.

`U16Vec3`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*U16Vec3*) - The new vector.

Initializes a new U16Vec3.

`U16Vec3`(*vec*)

Params

- **vec** (*U16Vec3*) - The vector to copy from.

Returns

- **value** (*U16Vec3*) - The new vector.

Initializes a new U16Vec3.

`U16Vec3`(*x*, *y*, *z*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.

Returns

- **value** (*U16Vec3*) - The new vector.

Initializes a new U16Vec3.

`U8Vec3`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*U8Vec3*) - The new vector.

Initializes a new U8Vec3.

`U8Vec3`(*vec*)

Params

- **vec** (*U8Vec3*) - The vector to copy from.

Returns

- **value** (*U8Vec3*) - The new vector.

Initializes a new U8Vec3.

`U8Vec3`(*x*, *y*, *z*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.

Returns

- **value** (*U8Vec3*) - The new vector.

Initializes a new U8Vec3.

`IVec3`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*IVec3*) - The new vector.

Initializes a new IVec3.

`IVec3`(*vec*)

Params

- **vec** (*IVec3*) - The vector to copy from.

Returns

- **value** (*IVec3*) - The new vector.

Initializes a new IVec3.

`IVec3`(*x*, *y*, *z*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.

Returns

- **value** (*IVec3*) - The new vector.

Initializes a new IVec3.

`Vec4`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*Vec4*) - The new vector.

Initializes a new Vec4.

`Vec4`(*vec*)

Params

- **vec** (*Vec4*) - The vector to copy from.

Returns

- **value** (*Vec4*) - The new vector.

Initializes a new Vec4.

`Vec4`(*x*, *y*, *z*, *w*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.
- **w** (`number`) - Fourth component.

Returns

- **value** (*Vec4*) - The new vector.

Initializes a new Vec4.

`UVec4`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*UVec4*) - The new vector.

Initializes a new UVec4.

`UVec4`(*vec*)

Params

- **vec** (*UVec4*) - The vector to copy from.

Returns

- **value** (*UVec4*) - The new vector.

Initializes a new UVec4.

`UVec4`(*x*, *y*, *z*, *w*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.
- **w** (`number`) - Fourth component.

Returns

- **value** (*UVec4*) - The new vector.

Initializes a new UVec4.

`IVec4`(*x*)

Params

- **x** (`number`) - The scalar value to set the components to.

Returns

- **value** (*IVec4*) - The new vector.

Initializes a new IVec4.

`IVec4`(*vec*)

Params

- **vec** (*IVec4*) - The vector to copy from.

Returns

- **value** (*IVec4*) - The new vector.

Initializes a new IVec4.

`IVec4`(*x*, *y*, *z*, *w*)

Params

- **x** (`number`) - First component.
- **y** (`number`) - Second component.
- **z** (`number`) - Third component.
- **w** (`number`) - Fourth component.

Returns

- **value** (*IVec4*) - The new vector.

Initializes a new IVec4.

`Quat`(*x*)

Params

- **x** (`number`) - The scalar value to set the first component to.

Returns

- **value** (*Quat*) - The new quaternion.

Initializes a new Quat.

`Quat`(*quat*)

Params

- **quat** (*Quat*) - The quaternion to copy from.

Returns

- **value** (*Quat*) - The new quaternion.

Initializes a new Quat.

`Quat`(*w*, *x*, *y*, *z*)

Params

- **w** (`number`) - First component.
- **x** (`number`) - Second component.
- **y** (`number`) - Third component.
- **z** (`number`) - Fourth component.

Returns

- **value** (*Quat*) - The new quaternion.

Initializes a new Quat.

`UIAnchor`(*anchor*, *offset*)

Params

- **anchor** (`number`) - The position in [0, 1] relative to the parent transform.
- **offset** (`number`) - The absolute offset (in px).

Returns

- **value** (*UIAnchor*) - The new anchor.

Initializes a new UI anchor.

`UIAnchorOffsets`(*left*, *right*, *top*, *bottom*)

Params

- **left** (`number`) - Absolute offset (in px) to the left anchor.
- **right** (`number`) - Absolute offset (in px) to the right anchor.
- **top** (`number`) - Absolute offset (in px) to the top anchor.
- **bottom** (`number`) - Absolute offset (in px) to the bottom anchor.

Returns

- **value** (*UIAnchor*) - The new set of anchor offsets.

Initializes a new set of UI anchor offsets.

`class Handle`

Handles are used to reference elements of the various subsystems.

`class PropertyDesc`

Variables

- **name** (*string*) –
- **type** (*string*) –

Description of a property.

`class Ref`

Reference to entities, components and resources.

`class Variant`

Special data type that can contain different types of data.

`class Vec2`

Variables

- **x** (*number*) –
- **y** (*number*) –

A vector storing two floating point components.

`class UVec2`

Variables

- **x** (*number*) –
- **y** (*number*) –

A vector storing two unsigned integer components.

`class IVec2`

Variables

- **x** (*number*) –
- **y** (*number*) –

A vector storing two signed integer components.

`class Vec3`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –

A vector storing three floating point components.

`class UVec3`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –

A vector storing three unsigned integer components.

`class U16Vec3`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –

A vector storing three unsigned integer components.

`class U8Vec3`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –

A vector storing three unsigned integer components.

`class IVec3`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –

A vector storing three signed integer components.

`class Vec4`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –
- **w** (*number*) –

A vector storing four floating point components.

`class UVec4`

Variables

- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –
- **w** (*number*) –

A vector storing four unsigned integer components.

`class IVec4`

Variables

- **x** (*number*) –

- **y** (*number*) –
- **z** (*number*) –
- **w** (*number*) –

A vector storing four signed integer components.

class Quat

Variables

- **w** (*number*) –
- **x** (*number*) –
- **y** (*number*) –
- **z** (*number*) –

A quaternion with four floating point components.

class HeightmapPixel

A single pixel used for generating heightmaps.

class PathSettings

Variables

- **capsule_radius** (*number*) – The radius of the agent's capsule.
- **capsule_half_height** (*number*) – The half height of the agent's capsule.
- **step_height** (*number*) – The maximum step height an agent can take.
- **cell_size** (*number*) – The size of the cells used for the voxelization.

Settings used when calculating paths via the Pathfinding related functions.

class PhysicsContactEvent

Variables

- **type** (*string*) – The type name.
- **data** (PhysicsContactEventData) – The data of the contact event.

Physics event fired when contacts between two shapes are detected.

class PhysicsContactEventData

Variables

- **entity0** (Ref) – The first entity in contact.
- **entity1** (Ref) – The second entity in contact.
- **pos** (Vec3) – The position of the contact.
- **impulse** (Vec3) – The impulse of the contact.

The data for a single physics contact event.

class UIAnchor

Defines an anchor used for creating (rectangle) transforms in the UI system.

class UIAnchorOffsets

Defines a set of anchor offsets used for creating (rectangle) transforms in the UI system.

class UIRect

Variables

- **pos** (Vec2) –
- **extent** (Vec2) –

A rectangle defined by a position and extent.

class KeyState

Variables

- **kReleased** (*number*) – The key is released.
- **kPressed** (*number*) – The key is pressed.
- **kClicked** (*number*) – The key has been clicked (pressed and released).

The different states a key can have.

class Key

Variables

- **kUp** (*number*) –
- **kDown** (*number*) –
- **kLeft** (*number*) –
- **kRight** (*number*) –
- **kA** (*number*) –
- **kB** (*number*) –
- **kC** (*number*) –
- **kD** (*number*) –
- **kE** (*number*) –
- **kF** (*number*) –
- **kG** (*number*) –
- **kH** (*number*) –
- **kI** (*number*) –
- **kJ** (*number*) –
- **kK** (*number*) –
- **kL** (*number*) –
- **kM** (*number*) –
- **kN** (*number*) –
- **kO** (*number*) –
- **kP** (*number*) –
- **kQ** (*number*) –
- **kR** (*number*) –
- **kS** (*number*) –

- **kT** (*number*) –
- **kU** (*number*) –
- **kV** (*number*) –
- **kW** (*number*) –
- **kX** (*number*) –
- **kY** (*number*) –
- **kZ** (*number*) –
- **k0** (*number*) –
- **k1** (*number*) –
- **k2** (*number*) –
- **k3** (*number*) –
- **k4** (*number*) –
- **k5** (*number*) –
- **k6** (*number*) –
- **k7** (*number*) –
- **k8** (*number*) –
- **k9** (*number*) –
- **kF1** (*number*) –
- **kF2** (*number*) –
- **kF3** (*number*) –
- **kF4** (*number*) –
- **kF5** (*number*) –
- **kF6** (*number*) –
- **kF7** (*number*) –
- **kF8** (*number*) –
- **kF9** (*number*) –
- **kF10** (*number*) –
- **kF11** (*number*) –
- **kF12** (*number*) –
- **kDel** (*number*) –
- **kBackspace** (*number*) –
- **kTab** (*number*) –
- **kMouseLeft** (*number*) –
- **kMouseRight** (*number*) –
- **kMouseMiddle** (*number*) –
- **kShift** (*number*) –

- **kAlt** (*number*) –
- **kCtrl** (*number*) –
- **kSpace** (*number*) –
- **kEscape** (*number*) –
- **kReturn** (*number*) –
- **kNumPlus** (*number*) –
- **kNumMinus** (*number*) –
- **kNum0** (*number*) –
- **kNum1** (*number*) –
- **kNum2** (*number*) –
- **kNum3** (*number*) –
- **kNum4** (*number*) –
- **kNum5** (*number*) –
- **kNum6** (*number*) –
- **kNum7** (*number*) –
- **kNum8** (*number*) –
- **kNum9** (*number*) –
- **kControllerButtonA** (*number*) –
- **kControllerButtonY** (*number*) –
- **kControllerButtonB** (*number*) –
- **kControllerButtonX** (*number*) –
- **kAny** (*number*) –

The different keys available.

`class Axis`

Variables

- **kLeftX** (*number*) – Left stick x-axis.
- **kLeftY** (*number*) – Left stick y-axis.
- **kRightX** (*number*) – Right stick x-axis.
- **kRightY** (*number*) – Right stick y-axis.
- **kTriggerLeft** (*number*) – Left trigger.
- **kTriggerRight** (*number*) – Right trigger.

The different axes for controller input handling.

10.1.4 Ref

`Ref.get_type_id(ref)`

Params

- **ref** (*Ref*) - The ref.

Returns

- **value** (number) - The ID of the type referenced.

Returns the ID of the type referenced.

`Ref.is_valid(ref)`

Params

- **ref** (*Ref*) - The ref.

Returns

- **value** (boolean) - True if the given ref is valid.

Returns the ID of the type referenced.

`Ref.get_id(ref)`

Params

- **ref** (*Ref*) - The ref.

Returns

- **value** (number) - The ID of the referenced resource.

Returns the unique ID of the referenced resource.

10.1.5 Variant

`Variant.from_float(value)`

Params

- **value** (number) - The floating point value to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a single floating point value.

`Variant.from_int(value)`

Params

- **value** (number) - The integer value to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a single integer value.

`Variant.`**`from_uint`**`(`*value*`)`

Params

- **value** (`number`) - The unsigned integer value to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a single unsigned integer value.

`Variant.`**`from_string`**`(`*value*`)`

Params

- **value** (`string`) - The string to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a string.

`Variant.`**`from_vec2`**`(`*value*`)`

Params

- **value** (*Vec2*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a vector with two components.

`Variant.`**`from_vec3`**`(`*value*`)`

Params

- **value** (*Vec3*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a vector with three components.

`Variant.`**`from_vec4`**`(`*value*`)`

Params

- **value** (*Vec4*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a vector with four components.

`Variant.`**`from_quat`**`(`*value*`)`

Params

- **value** (*Quat*) - The quternion to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing a quaternion with four components.

`Variant.`**`from_ivec2`**`(`*value*`)`

Params

- **value** (*IVec2*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an integer vector with two components.

`Variant.`**`from_ivec3`**`(`*value*`)`

Params

- **value** (*IVec3*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an integer vector with three components.

`Variant.`**`from_ivec4`**`(`*value*`)`

Params

- **value** (*IVec4*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an integer vector with fours components.

`Variant.`**`from_uvec2`**`(`*value*`)`

Params

- **value** (*UVec2*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an unsigned integer vector with two components.

`Variant.`**`from_uvec3`**`(`*value*`)`

Params

- **value** (*UVec3*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an unsigned integer vector with three components.

`Variant.`**`from_u8vec3`**`(`*value*`)`

Params

- **value** (*U8Vec3*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an 8-bit unsigned integer vector with three components.

`Variant.`**`from_u16vec3`**`(`*value*`)`

Params

- **value** (*U16Vec3*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an 16-bit unsigned integer vector with three components.

`Variant.`**`from_uvec4`**`(`*value*`)`

Params

- **value** (*UVec4*) - The vector to set.

Returns

- **value** (*Variant*) - The new variant.

Creates a new variant storing an unsigned integer vector with fours components.

`Variant.`**`get_float`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (`number`) - The floating point value.

Returns the underlying floating point value.

`Variant.`**`get_int`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (`number`) - The integer value.

Returns the underlying integer value.

`Variant.`**`get_uint`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (`number`) - The unsigned integer value.

Returns the underlying unsigned integer value.

`Variant.`**`get_string`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (number) - The string.

Returns the underlying string.

`Variant.`**`get_vec2`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*Vec2*) - The vector.

Returns the underlying two-component vector.

`Variant.`**`get_vec3`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*Vec3*) - The vector.

Returns the underlying three-component vector.

`Variant.`**`get_vec4`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*Vec4*) - The vector.

Returns the underlying four-component vector.

`Variant.`**`get_quat`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*Quat*) - The quaternion.

Returns the underlying four-component quaternion.

`Variant.`**`get_ivec2`**`(`*variant*`)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*IVec2*) - The vector.

Returns the underlying two-component integer vector.

`Variant.get_ivec3(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*IVec3*) - The vector.

Returns the underlying three-component integer vector.

`Variant.get_ivec4(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*IVec4*) - The vector.

Returns the underlying four-component integer vector.

`Variant.get_uvec2(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*UVec2*) - The vector.

Returns the underlying two-component unsigned integer vector.

`Variant.get_uvec3(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*UVec3*) - The vector.

Returns the underlying three-component unsigned integer vector.

`Variant.get_u8vec3(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*U8Vec3*) - The vector.

Returns the underlying three-component 8-bit unsigned integer vector.

`Variant.get_u16vec3(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*U16Vec3*) - The vector.

Returns the underlying three-component 16-bit unsigned integer vector.

`Variant.get_uvec4(variant)`

Params

- **variant** (*Variant*) - The variant to retrieve from.

Returns

- **value** (*UVec4*) - The vector.

Returns the underlying four-component unsigned integer vector.

10.1.6 Math Constants

class Math

Variables

- **pi** (*number*) – Pi
- **half_pi** (*number*) – Pi / 2
- **two_pi** (*number*) – Pi * 2

Various math related constants.

10.1.7 Math General

`Math.load()`

Loads all functions and types for this scripting interface.

`Math.pow(x, y)`

Params

- **x** (number) - The base.
- **y** (number) - The exponent.

Returns

- **value** (number) - The result of x raised to power y.

Calculates x raised to the power y.

`Math.sqrt(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The square root of x.

Calculates the square root of x.

`Math.exp(x)`

Params

- **x** (`number`) - The exponent.

Returns

- **value** (`number`) - The result of e raised to the power x.

Calculates e (Euler's number) raised to the power x.

`Math.abs(x)`

Params

- **x** (`number`) - The input value.

Returns

- **value** (`number`) - The absolute value of x.

Calculates the absolute value of x.

`Math.clamp(x, min, max)`

Params

- **x** (`number`) - The value to clamp.
- **min** (`number`) - The minimum.
- **max** (`number`) - The maximum.

Returns

- **value** (`number`) - The result of x clamped to [min, max].

Clamps x to the provided minimum and maximum.

`Math.min(x, y)`

Params

- **x** (`number`) - The first input value.
- **y** (`number`) - The second input value.

Returns

- **value** (`number`) - The minimum of x and y.

Calculates the minimum of x and y.

`Math.max(x, y)`

Params

- **x** (`number`) - The first input value.
- **y** (`number`) - The second input value.

Returns

- **value** (`number`) - The maximum of x and y.

Calculates the maximum of x and y.

`Math.floor(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The largest integer value not greater than x.

Calculates the largest integer number to greater than x.

`Math.ceil(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The smallest integer value not less than x.

Calculates the smallest integer number not less than x.

`Math.round(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The rounded value.

Calculates the nearest integer value to x, rounding halfway cases away from zero.

`Math.fract(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The fractional part of x.

Returns the fractional part of x, calculated as x - floor(x).

`Math.trunc(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The truncated value.

Calculates a value equal to the nearest integer to x whose absolute value is not larger than the absolute value of x.

10.1.8 Math Trigonometry

`Math.radians(x)`

Params

- **x** (`number`) - Input value in degrees.

Returns

- **value** (`number`) - Input value converted to radians.

Convers the provided value in degrees to radians.

`Math.degrees(angle)`

Params

- **angle** (`number`) - Input value in radians.

Returns

- **value** (`number`) - Input value converted to degrees.

Convers the provided value in radians to degrees.

`Math.sin(angle)`

Params

- **angle** (`number`) - The input angle (in radians).

Returns

- **value** (`number`) - The sine of angle.

Calculates the sine of angle.

`Math.asin(x)`

Params

- **x** (`number`) - The input value.

Returns

- **value** (`number`) - The arc sine of x (in radians).

Calculates the arc sine of x.

`Math.cos(angle)`

Params

- **angle** (`number`) - The input angle.

Returns

- **value** (`number`) - The cosine of angle.

Calculates the cosine of angle.

`Math.acos(x)`

Params

- **x** (`number`) - The input value.

Returns

- **value** (number) - The arc cosine of x (in radians).

Calculates the arc cosine of x.

`Math.tan(angle)`

Params

- **angle** (number) - The input angle.

Returns

- **value** (number) - The tangent of angle.

Calculates the tangent of angle.

`Math.atan(x)`

Params

- **x** (number) - The input value.

Returns

- **value** (number) - The arc tangent of x.

Calculates the arc tangent of x.

`Math.atan(y, x)`

Params

- **y** (number) - The first input value.
- **x** (number) - The second input value.

Returns

- **value** (number) - The arc tangent of y over x.

Calculates the arc tangent of y over x.

10.1.9 Math Interpolation

`Math.lerp(x, y, a)`

Params

- **x** (number or *Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (number or *Vec2* or *Vec3* or *Vec4*) - The second input value.
- **a** (number) - The value to use to interpolate between x and y.

Returns

- **value** (number or *Vec2* or *Vec3* or *Vec4*) - The interpolated value.

Linearly interpolates between x and y.

`Math.slerp(x, y, a)`

Params

- **x** (*Quat*) - The first input value.
- **y** (*Quat*) - The second input value.

- **a** (number) - The value to use to interpolate between x and y.

Returns

- **value** (*Quat*) - The interpolated value.

Spherically linearly interpolates between x and y.

10.1.10 Math Vectors

Math.**vec_get_component**(*x*, *idx*)

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The input value.
- **idx** (number) - The index of the component to retrieve.

Returns

- **value** (number) - The select4ed component of x.

Selects the component of the vector using the provided index.

Math.**vec_mul**(*x*, *y*)

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (*Vec2* or *Vec3* or *Vec4*) - The result of x * y.

Multiplies the provided vectors.

Math.**vec_scale**(*s*, *x*)

Params

- **s** (number) - The scalar value.
- **x** (*Vec2* or *Vec3* or *Vec4*) - The vector to scale.

Returns

- **value** (*Vec2* or *Vec3* or *Vec4*) - The result of s*x.

Scales the vector by the given scalar

Math.**vec_div**(*x*, *y*)

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (*Vec2* or *Vec3* or *Vec4*) - The result of x/y.

Divides the first vector by the second.

`Math.vec_add(x, y)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (*Vec2* or *Vec3* or *Vec4*) - The result of x+y.

Adds the provided vectors.

`Math.vec_sub(x, y)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (*Vec2* or *Vec3* or *Vec4*) - The result of x-y.

Subtracts the provided vectors.

`Math.vec_length(x)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The input value.

Returns

- **value** (`number`) - The length of the vector, calculated as sqrt(x.x*x.x + x.y*x.y + x.z*x.z).

Calculates the length of the provided vector

`Math.vec_length2(x)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The input value.

Returns

- **value** (`number`) - The length of the vector, calculated as x.x*x.x + x.y*x.y + x.z*x.z.

Calculates the squared length of the provided vector

`Math.vec_distance(x, y)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (`number`) - The distance between the vectors, calculated as vec_length(vec_sub(y, x)).

Calculates the distance between the provided vectors.

`Math.vec_distance2(x, y)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (`number`) - The distance between the vectors, calculated as vec_length2(vec_sub(y, x)).

Calculates the squared distance between the provided vectors.

`Math.vec_normalize(x)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The input value.

Returns

- **value** (*Vec2* or *Vec3* or *Vec4*) - The normalized vector of x, calculate as vec_scale(1.0 / vec_length(x), x).

Normalizes the provided vector.

`Math.vec_dot(x, y)`

Params

- **x** (*Vec2* or *Vec3* or *Vec4*) - The first input value.
- **y** (*Vec2* or *Vec3* or *Vec4*) - The second input value.

Returns

- **value** (`number`) - The dot product of x and y.

Calculates the dot product of the provided vectors.

`Math.vec_cross(x, y)`

Params

- **x** (*Vec3*) - The first input value.
- **y** (*Vec3*) - The second input value.

Returns

- **value** (*Vec3*) - The cross product of x and y.

Calculates the cross product of the provided vectors.

10.1.11 Math Quaternions

`Math.quat_mul(x, y)`

Params

- **x** (*Quat*) - The first input value.
- **y** (*Quat*) - The second input value.

Returns

- **value** (*Quat*) - The result of x*y.

Multiplies the provided quaternions

`Math.quat_rotate(x, y)`

Params

- **x** (*Quat*) - The quaternion to rotate by.
- **y** (*Vec3*) - The vector to rotate.

Returns

- **value** (*Vec3*) - The rotated vector.

Rotates the provided vector by the given quaternion.

`Math.quat_inverse(x)`

Params

- **x** (*Quat*) - The quaternion to invert.

Returns

- **value** (*Quat*) - The inverse of the quaternion.

Calculate the inverse of the provided quaternion.

`Math.quat_normalize(x)`

Params

- **x** (*Quat*) - The quaternion to normalize.

Returns

- **value** (*Quat*) - The normalized quaternion.

Normalizes the provided quaternion.

`Math.quat_look_at(dir, up)`

Params

- **dir** (*Vec3*) - The direction to look into.
- **up** (*Vec3*) - The up vector to use.

Returns

- **value** (*Quat*) - Quaternion “looking” into the provided direction.

Calculates a quaternion “looking” into a specific direction.

`Math.quat_from_angle_axis(angle, axis)`

Params

- **angle** (`number`) - The angle.
- **axis** (*Vec3*) - The axis of the rotation.

Returns

- **value** (*Quat*) - Quaternion rotating angle around axis.

Calculates a quaternion from the angle and axis.

`Math.quat_to_euler_angles(x)`

Params

- **x** (*Quat*) - The quaternion to convert.

Returns

- **value** (*Vec3*) - The Euler angles, pitch, yaw, and roll derived from the given Quaternion in radians.

Converts the provided quaternion to Euler angles.

`Math.quat_from_euler_angles(angles)`

Params

- **angles** (*Vec3*) - The Euler angles (in radians) to convert.

Returns

- **value** (*Quat*) - The quaternion constructed from the provided Euler angles.

Converts the provided Euler angles to a quaternion.

`Math.quat_rotation(x, y)`

Params

- **x** (*Vec3*) - The first vector.
- **y** (*Vec3*) - The second vector.

Returns

- **value** (*Quat*) - Quaternion transforming vector to vector.y

Calculates a quaternion that transforms vector x to vector y.

10.1.12 Settings

`Settings.load()`

Loads all functions and types for this scripting interface.

`Settings.set_bool(name, value)`

Params

- **name** (`string`) - The name of the setting.
- **value** (`boolean`) - The value to set.

Sets the given setting to the provided boolean value.

`Settings.set_uint(name, value)`

Params

- **name** (`string`) - The name of the setting.
- **value** (`number`) - The value to set.

Sets the given setting to the provided unsigned integer value.

`Settings.set_float(name, value)`

Params

- **name** (`string`) - The name of the setting.
- **value** (`number`) - The value to set.

Sets the given setting to the provided floating point value.

`Settings.get_bool(name)`

Params

- **name** (`string`) - The name of the setting to retrieve.

Returns

- **value** (`boolean`) - The value of the setting.

Retrieves the given boolean setting.

`Settings.get_uint(name)`

Params

- **name** (`string`) - The name of the setting to retrieve.

Returns

- **value** (`number`) - The value of the setting.

Retrieves the given (unsigned) integer setting.

`Settings.get_float(name)`

Params

- **name** (`string`) - The name of the setting to retrieve.

Returns

- **value** (`number`) - The value of the setting.

Retrieves the given floating point setting.

10.1.13 Logging

`Log.load()`

Loads all functions and types for this scripting interface.

`Log.log_info(s)`

Params

- **s** (`string`) - The string to log.

Logs the given string as info.

`Log.log_warning(s)`

Params

- **s** (`string`) - The string to log.

Logs the given string as a warning.

`Log.log_error(s)`

Params

- **s** (`string`) - The string to log.

Logs the given string as an error.

10.1.14 UI

`UI.load()`

Loads all functions and types for this scripting interface.

`UI.draw_rect(color)`

Params

- **color** (*Vec4*) - The color of the rectangle.

Draws a rectangle.

`UI.draw_direct(color)`

Params

- **color** (*Vec4*) - The color of the circle.

Draws a circle.

`UI.draw_ngon(color, num_sides)`

Params

- **color** (*Vec4*) - The color of the n-sided polygon.
- **num_sides** (`number`) - The number of sided.

Draws a n-sided polygon.

`UI.draw_image(name, tint)`

Params

- **name** (`string`) - The name of the image to draw.
- **tint** (*Vec4*) - The tint of the image.

Draws the image with the given name.

`UI.get_image_size(name)`

Params

- **name** (`string`) - The name of the image.

Returns

- **value** (*Vec2*) - The size of the image (in px).

Gets the size of the image (in px).

`UI.draw_text(text, align_horizontal, align_vertical, flags)`

Params

- **text** (`string`) - The text to draw.
- **align_horizontal** (`number`) - The horizontal alignment.
- **align_vertical** (`number`) - The vertical alignment.
- **flags** (`number`) - The text flags.

Draws the text with the given options.

`UI.calc_text_bounds(text, align_horizontal, align_vertical, flags)`

Params

- **text** (`string`) - The text to compute the bounds for.
- **align_horizontal** (`number`) - The horizontal alignment.
- **align_vertical** (`number`) - The vertical alignment.
- **flags** (`number`) - The text flags.

Returns

- **value** (*Vec2*) - The bounds for the text.

Calculates the bounds for the given text and options.

`UI.get_last_text_bounds()`

Returns

- **value** (*Vec2*) - The bounds for the text.

Returns the bounds of the text that was drawn last.

`UI.push_transform(left, right, top, bottom, rotation)`

Params

- **left** (*UIAnchor*) - Left anchor.
- **right** (*UIAnchor*) - Right anchor.

- **top** (*UIAnchor*) - Top anchor.
- **bottom** (*UIAnchor*) - Bottom anchor.
- **rotation** (number) - The rotation to apply.

Pushes the previous transform to the stack and activates the given one.

UI.**push_transform_preset**(*preset*, *offsets*, *rotation*)

Params

- **preset** (number) - The transform preset to use.
- **offsets** (*UIAnchorOffsets*) - The offsets for each of the anchors.
- **rotation** (number) - The rotation to apply.

Pushes the previous transform to the stack and activates the given one.

UI.**pop_transform**()

Pops the last transform off the stack and activates it.

UI.**push_scale_offset_for_base_size**(*base_size*, *aspect_mode*)

Params

- **base_size** (*Vec2*) - The base size.
- **aspect_mode** (number) - The aspect mode to use.

Calculates the scale and offset for the given base size and according to the aspect mode.

UI.**push_scale_offset**(*scale*, *offset*)

Params

- **scale** (number) - The uniform scaling factor.
- **offset** (*Vec2*) - The offset.

Pushes the scale and offset to the stack and activates the given parameters.

UI.**pop_scale_offset**()

Pops the last scale and offset from the stack and activates it.

UI.**push_style_var_float**(*var*, *value*)

Params

- **var** (number) - The style variation.
- **value** (number) - The value to set.

Pushes the current style variation float value to the stack and sets the given one.

UI.**push_style_var_vec4**(*var*, *value*)

Params

- **var** (number) - The style variation.
- **value** (*Vec4*) - The value to set.

Pushes the current style variation Vec4 value to the stack and sets the given one.

UI.**pop_style_var**()

Pops the last style variation from the stack and activates it.

UI.**clip_children**()

Clips the children of the current transform

UI.**push_font_size**(*size*)

Params

- **size** (number) - The font size (in px).

Pushes the current font size to the stack and sets the given one.

UI.**pop_font_size**()

Pops the last font size from the stack and activates it.

UI.**intersects**()

Returns true if the given position (in px) intersects the current transform.

10.1.15 Random

Random.**load**()

Loads all functions and types for this scripting interface.

Random.**set_seed**(*seed*)

Params

- **seed** (number) - The seed to set.

Sets the seed used for the underlying random number generator (RNG).

Random.**rand_uint**()

Returns

- **value** (number) - The random value.

Calculates a random (unsigned) integer value.

Random.**rand_uint_min_max**(*min*, *max*)

Params

- **min** (number) - The minimum random value to generate.
- **max** (number) - The maximum random value to generate.

Returns

- **value** (number) - The random value.

Calculates a random (unsigned) integer value in the given interval.

Random.**rand_float**()

Returns

- **value** (number) - The random value.

Calculates a random floating point value in [0.0, 1.0].

Random.**rand_float_min_max**(*min*, *max*)

Params

- **min** (number) - The minimum random value to generate.
- **max** (number) - The maximum random value to generate.

Returns

- **value** (number) - The random value.

Calculates a random floating point value in the given interval.

10.1.16 Entity

Entity.**load**()

Loads all functions and types for this scripting interface.

Entity.**get_type_id**()

Returns

- **value** (number) - The type id for all entities.

Gets the type id for all entities.

Entity.**is_alive**(*entity*)

Params

- **entity** (*Ref*) - The entity to check.

Returns

- **value** (boolean) - True if the given entity is alive.

Returns whether the given entity is alive.

Entity.**get_name**(*entity*)

Params

- **entity** (*Ref*) - The entity to check.

Returns

- **value** (string) - The name of the given entity.

Returns the name of the given entity.

Entity.**find_first_entity_with_name**(*name*)

Params

- **name** (string) - The name of the entity to search for.

Returns

- **value** (*Ref*) - The entity with the given name. Ref is invalid if none was found.

Finds the first entity with the given name.

Entity.**find_entities_with_name**(*name*)

Params

- **name** (string) - The name of the entities to search for.

Returns

- **value** (table) - Table containing the entities with the given name.

Finds all entities with the given name and returns as table containg refs as result.

10.1.17 Save Data

SaveData.**load**()

Loads all functions and types for this scripting interface.

SaveData.**save_to_user_data**(*filename*, *node*)

Params

- **filename** (string) - The filename to use (without the file extension).
- **node** (*Ref*) - The node to save.

Stores the given node hierarchy using the given filename in the user data directory.

SaveData.**load_from_user_data**(*filename*)

Params

- **filename** (string) - The filename to load from (without the file extension).

Returns

- **value** (*Ref*) - The root node of the loaded hierarchy.

Loads the node hierarchy from the given file in the user data directory.

10.1.18 World

World.**load**()

Loads all functions and types for this scripting interface.

World.**get_root_node**()

Returns

- **value** (*Ref*) - The root node of the world.

Returns the world's root node.

World.**get_world_name**()

Returns

- **value** (string) - The name of the currently loaded world.

Returns the name of the currently loaded world.

World.**load_world**(*name*)

Params

- **name** (string) - The name of the world to load.

Loads the world with the given name.

World.**save_world**(*name*)

Params

- **name** (string) - The name of the world to save.

Saves the current world under the given name.

World.**spawn_prefab**(*name*)

Params

- **name** (string) - The name of the prefab to load.

Returns

- **value** (*Ref*) - The root node of the loaded prefab.

Loads the prefab with the given name.

World.**get_current_time_factor**()

Returns

- **value** (*Ref*) - The current time factor.

Returns the current time factor.

World.**push_time_factor**(*factor*)

Params

- **factor** (number) - The time factor to apply.

Pushes the current time factor to the stack and applies the given one. Use this to speed up or slow down the game time.

World.**pop_time_factor**()

Pops the last time factor from the stack and activates it.

World.**radius_damage**(*position*, *radius*, *flags*, *max_hardness*)

Params

- **position** (*Vec3*) - The position.
- **radius** (number) - The radius.
- **flags** (number) - The flags to use. Bit 1: Shade voxels around the impact area, Bit 2: Apply as fracturing.
- **max_hardness** (number) - Limits which voxels this damage operation can affect based on the hardness material parameter.

Applies damage to all voxel shapes in the given radius in the world.

World.**calc_mouse_ray**()

Returns

- **origin** (*Vec3*) - The origin of the ray (in world coordinates).
- **direction** (*Vec3*) - The direction of the ray.

Calculates a ray that points at the current mouse position in the world.

World.**highlight_node**(*node*, *color*, *outline*)

Params

- **node** (*Ref*) - The node to highlight.
- **color** (*Vec4*) - The color of the highlight.
- **outline** (boolean) - Set to true to only highlight using an outline.

Highlights the given node using an outline shader.

10.1.19 Particle System

ParticleSystem.**load**()

Loads all functions and types for this scripting interface.

ParticleSystem.**spawn_particle_emitter**(*effect*, *position*, *lifetime*, *adjust_spawn_rate*)

Params

- **effect** (string) - The name of the effect.
- **position** (*Vec3*) - The spawn position of the emitter (in world coordinates).
- **lifetime** (number) - The lifetime of the emitter in seconds.
- **adjust_spawn_rate** (boolean) - Set to true to adjust the spawn rate based on the lifetime.

Returns

- **value** (*Handle*) - The handle for the emitter.

Spawns a new particle emitter.

ParticleSystem.**despawn_particle_emitter**(*emitter*)

Params

- **emitter** (*Handle*) - The emitter to despawn.

Despawns the given particle emitter.

ParticleSystem.**attach_to_node**(*emitter*, *node*)

Params

- **emitter** (*Handle*) - The emitter to attach.
- **node** (*Ref*) - The node to attach the emitter to.

Attaches the given particle emitter to the provided node.

`ParticleSystem.set_spawn_rate(emitter, spawn_rate)`

Params

- **emitter** (*Handle*) - The emitter to adjust.
- **spawn_rate** (number) - The spawn rate to set.

Sets the spawn rate for the given emitter.

`ParticleSystem.set_position(emitter, position)`

Params

- **emitter** (*Handle*) - The emitter to adjust.
- **position** (*Vec3*) - The position to set.

Sets the position for the given emitter.

`ParticleSystem.set_scale(emitter, position)`

Params

- **emitter** (*Handle*) - The emitter to adjust.
- **position** (number) - The scale to set.

Sets the scale for the given emitter.

`ParticleSystem.set_emission_direction(emitter, direction)`

Params

- **emitter** (*Handle*) - The emitter to adjust.
- **direction** (*Vec3*) - The direction to set.

Sets the emission direction for the given emitter.

10.1.20 Input

`Input.load()`

Loads all functions and types for this scripting interface.

`Input.get_key_state(key, player)`

Params

- **key** (number) - The key to retrieve the state from (see Key table).
- **player** (number) - The index of the player to retrieve the state from.

Returns

- **value** (number) - The current state of the key (see KeyState table).

Gets the current state of the given key.

`Input.get_axis_state(axis, player)`

Params

- **axis** (number) - The axis to retrieve the state from (see Axis table).
- **player** (number) - The index of the player to retrieve the state from.

Returns

- **value** (number) - The current state of the axis in [0.0, 1.0].

Gets the current state of the given axis.

`Input.get_mouse_pos()`

Returns

- **value** (*Vec2*) - The position of the mouse.

Gets the current position of the mouse as pixel coordinates.

`Input.get_mouse_pos_viewport()`

Returns

- **value** (*Vec2*) - The position of the mouse in [0.0, 1.0].

Gets the current position of the mouse in the viewport.

`Input.get_mouse_pos_relative()`

Returns

- **value** (*Vec2*) - The delta movement of the mouse.

Gets the delta movement of the mouse in pixels.

`Input.request_mouse_cursor()`

Request the mouse cursor to show. Has to be called every frame to keep the mouse cursor visible.

10.1.21 Terrain

`Terrain.load()`

Loads all functions and types for this scripting interface.

`Terrain.generate_from_image(`*filename*, *palette*, *max_height*, *voxel_size*`)`

Params

- **filename** (string) - The filename of the image.
- **palette** (string) - The name of the palette to use.
- **max_height** (number) - The maximum height of the terrain in world coordinates.
- **voxel_size** (number) - The size of a single voxel.

Returns

- **value** (*Ref*) - The root node of the terrain.

Generates a new heightmap-based terrain from the given image.

`Terrain.generate_from_data(`*data*, *size*, *palette*, *max_height*, *voxel_size*`)`

Params

- **data** (table) - The heightmap data as a table made up of pixels (see Terrain.Pixel)
- **size** (number) - The width/height of the terrain in pixels.
- **palette** (string) - The name of the palette to use.

- **max_height** (number) - The maximum height of the terrain in world coordinates.
- **voxel_size** (number) - The size of a single voxel.

Returns

- **value** (*Ref*) - The root node of the terrain.

Generates a new heightmap-based terrain from the table.

`Terrain.HeightmapPixel(`*height*, *grass_height*, *palette_index*`)`

Params

- **height** (number) - The height in [0.0, 1.0].
- **grass_height** (number) - The grass height in [0.0, 1.0].
- **palette_index** (number) - The palette index.

Returns

- **value** (*HeightmapPixel*) - The new heightmap pixel.

Initializes a new pixel for the terrain generator.

10.1.22 Noise

`Noise.load()`

Loads all functions and types for this scripting interface.

`Noise.simplex(`*x*`)`

Params

- **x** (*Vec4*) - The coordinate to calculate the noise value at.

Returns

- **value** (number) - The noise value at the given coordinate.

Calculates perlin noise.

`Noise.simplex(`*x*`)`

Params

- **x** (*Vec4*) - The coordinate to calculate the noise value at.

Returns

- **value** (number) - The noise value at the given coordinate.

Calculates simplex noise.

10.1.23 Physics

`Physics.load()`

Loads all functions and types for this scripting interface.

`Physics.set_gravity(gravity)`

Params

- **gravity** (*Vec3*) - The gravity to set.

Sets the global gravity (in meters per second).

`Physics.get_gravity()`

Returns

- **value** (*Vec3*) - The current global gravity.

Gets the global gravity (in meters per second).

`Physics.overlap_sphere(position, radius)`

Params

- **position** (*Vec3*) - The position of the sphere in world coordinates.
- **radius** (`number`) - The radius of the sphere.

Returns

- **hit** (`boolean`) - True if a blocking hit is detected.
- **hit_entity** (*Ref*) - The blocking entity. Invalid ref if no hit was detected.

Performs a sphere overlap test against the physics geometry.

`Physics.sweep_sphere(position, radius, direction, distance)`

Params

- **position** (*Vec3*) - The position of the sphere in world coordinates.
- **radius** (`number`) - The radius of the sphere.
- **direction** (*Vec3*) - The direction to perform the sweep test in.
- **distance** (`number`) - The distance to sweep.

Returns

- **hit** (`boolean`) - True if a blocking hit is detected.
- **hit_distance** (`number`) - The distance to the hit.
- **hit_position** (*Vec3*) - The position of the hit.
- **hit_normal** (*Vec3*) - The normal of the hit.
- **hit_entity** (*Ref*) - The blocking entity. Invalid ref if no hit was detected.

Performs a sweeped sphere test against the physics geometry.

`Physics.raycast(ray_origin, ray_direction, ray_distance)`

Params

- **ray_origin** (*Vec3*) - The origin of the ray in world coordinates.

- **ray_direction** (*Vec3*) - The direction of ray.
- **ray_distance** (number) - The maximum distance the ray should travel.

Returns

- **hit** (boolean) - True if a blocking hit is detected.
- **hit_distance** (number) - The distance to the hit.
- **hit_position** (*Vec3*) - The position of the hit.
- **hit_normal** (*Vec3*) - The normal of the hit.
- **hit_entity** (*Ref*) - The blocking entity. Invalid ref if no hit was detected.

Performs a raycast again the physics geometry.

10.1.24 Debug Geometry

DebugGeometry.**load**()

Loads all functions and types for this scripting interface.

DebugGeometry.**draw_line**(*start_pos*, *end_pos*, *color*, *always_in_front*)

Params

- **start_pos** (*Vec3*) - Line start position.
- **end_pos** (*Vec3*) - Line end position.
- **color** (*Vec4*) - The color of the line.
- **always_in_front** (boolean) - Set to true to render the line in front of all the other geometry.

Draws a line.

DebugGeometry.**draw_sphere**(*position*, *radius*, *color*, *always_in_front*)

Params

- **position** (*Vec3*) - The center position of the sphere.
- **radius** (number) - The radius of the sphere.
- **color** (*Vec4*) - The color of the sphere.
- **always_in_front** (boolean) - Set to true to render the sphere in front of all the other geometry.

Draws a sphere.

10.1.25 Sound

Sound.**load**()

Loads all functions and types for this scripting interface.

Sound.**play_sound_effect**(*name*)

Params

- **name** (string) - The name of the sound effect to play.

Returns

- **handle** (*Handle*) - The handle of the created sound effect instance.

Starts playing the sound effect with the given name.

`Sound.stop_sound_effect(handle)`

Params

- **handle** (*Handle*) - The handle of the sound effect instance.

Stops playing the given sound effect instance.

`Sound.set_position(handle, pos)`

Params

- **handle** (*Handle*) - The handle of the sound effect instance.
- **pos** (*Vec3*) - The position to set.

Sets the position of the given sound effect instance in the world. Used for spatial audio effects.

`Sound.get_spectrum()`

Returns

- **spectrum** (`table`) - Table containing the bins of the sound spectrum.

Retrieves the current snapshot of the sound spectrum.

10.1.26 Pathfinding

`Pathfinding.load()`

Loads all functions and types for this scripting interface.

`Pathfinding.find_path(start_pos, end_pos, settings)`

Params

- **start_pos** (*Vec3*) - The start position in the world.
- **end_pos** (*Vec3*) - The end position in the world.
- **settings** (*PathSettings*) - The path settings to use for finding the path.

Returns

- **handle** (*Handle*) - The handle of the path.

Starts finding a path.

`Pathfinding.is_valid(handle)`

Params

- **handle** (*Handle*) - The handle of the path.

Returns

- **value** (`boolean`) - True if the path is valid.

Checks if the given path is valid.

Pathfinding.**reset_path**(*handle*)

Params

- **handle** (*Handle*) - The handle of the path.

Resets the given path.

Pathfinding.**destroy_path**(*handle*)

Params

- **handle** (*Handle*) - The handle of the path.

Destroys the given path.

Pathfinding.**is_path_found**(*handle*)

Params

- **handle** (*Handle*) - The handle of the path.

Returns

- **value** (boolean) - True if a path has been found.

Check if a valid path has been found.

Pathfinding.**get_next_position_on_path**(*handle*)

Params

- **handle** (*Handle*) - The handle of the path.

Returns

- **position** (*Vec3*) - The next position on the path.
- **success** (boolean) - True if a new position was retrieved.

Returns the next position on the path.

Pathfinding.**draw_debug_geometry**(*handle*, *in_front*)

Params

- **handle** (*Handle*) - The handle of the path.
- **in_front** (boolean) - Set to true to draw the debug geometry in front of all other geometry.

Draws the internal debug geometry for the given path.

Pathfinding.**draw_path**(*handle*, *color*, *in_front*)

Params

- **handle** (*Handle*) - The handle of the path.
- **color** (*Vec4*) - The color used for drawing.
- **in_front** (boolean) - Set to true to draw the path in front of all other geometry.

Draws the path using lines.

Pathfinding.**PathSettings**()

Returns

- **value** (*PathSettings*) - The path settings.

Creates and initializes the settings for calculating a path to the default values.

10.1.27 Custom Data Component

`CustomData.load()`

Loads all functions and types for this scripting interface.

`CustomData.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

`CustomData.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`CustomData.destroy(component)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

`CustomData.commit_changes(component)`

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

`CustomData.get_num_active_components()`

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

`CustomData.get_component_for_entity(entity)`

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

`CustomData.is_alive(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`boolean`) - True if the component is alive.

Returns true if the referenced component is alive.

`CustomData.get_entity(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

`CustomData.get_property(component, property_name)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`CustomData.set_property(component, property_name, value)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`CustomData.list_properties()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

`CustomData.get(component, index)`

Params

- **component** (*Ref*) - The custom data component.
- **index** (`number`) - The index of the value to retrieve.

Returns

- **value** (*Variant*) - The value for the given index.

Gets the value for the given index.

`CustomData.set(component, index, value)`

Params

- **component** (*Ref*) - The custom data component.
- **index** (number) - The index of the value to retrieve.
- **value** (*Variant*) - The value to set.

Sets the value for the given index.

`CustomData.add(component, value)`

Params

- **component** (*Ref*) - The custom data component.
- **value** (*Variant*) - The value to add.

Adds a new value.

`CustomData.remove(component, index)`

Params

- **component** (*Ref*) - The custom data component.
- **index** (number) - The index of the value to remove.

Adds a new value.

10.1.28 Tag Component

`Tag.load()`

Loads all functions and types for this scripting interface.

`Tag.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

`Tag.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`Tag.destroy(component)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

Tag.**commit_changes**(*component*)

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

Tag.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

Tag.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

Tag.**is_alive**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

Tag.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

Tag.**get_property**(*component*, *property_name*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`Tag.set_property(component, property_name, value)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`Tag.list_properties()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

`Tag.find_entities_with_tag(tag)`

Params

- **tag** (`string`) - The tag to search for.

Returns

- **value** (`table`) - Table containing all entities with the given tag.

Finds and returns all entities with the given tag.

10.1.29 Flipbook Animation Component

`FlipbookAnimation.load()`

Loads all functions and types for this scripting interface.

`FlipbookAnimation.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`number`) - The type ID of the component.

Returns the type ID of the component.

`FlipbookAnimation.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`FlipbookAnimation.destroy(component)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

FlipbookAnimation.**commit_changes**(*component*)

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

FlipbookAnimation.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

FlipbookAnimation.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

FlipbookAnimation.**is_alive**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

FlipbookAnimation.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

FlipbookAnimation.**get_property**(*component*, *property_name*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`FlipbookAnimation.set_property(component, property_name, value)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`FlipbookAnimation.list_properties()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

`FlipbookAnimation.play(component)`

Params

- **component** (*Ref*) - The component in question.

Starts playing the flipbook animation.

`FlipbookAnimation.stop(component)`

Params

- **component** (*Ref*) - The component in question.

Stops playing the flipbook animation.

10.1.30 Post Effect Volume Component

`PostEffectVolume.load()`

Loads all functions and types for this scripting interface.

`PostEffectVolume.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`number`) - The type ID of the component.

Returns the type ID of the component.

`PostEffectVolume.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

PostEffectVolume.**destroy**(*component*)

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

PostEffectVolume.**commit_changes**(*component*)

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

PostEffectVolume.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

PostEffectVolume.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

PostEffectVolume.**is_alive**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

PostEffectVolume.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

PostEffectVolume.**get_property**(*component*, *property_name*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`PostEffectVolume.`**`set_property`**`(`*component*, *property_name*, *value*`)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`PostEffectVolume.`**`list_properties`**`()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

10.1.31 Camera Component

`Camera.`**`load`**`()`

Loads all functions and types for this scripting interface.

`Camera.`**`get_type_id`**`(`*component*`)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`number`) - The type ID of the component.

Returns the type ID of the component.

`Camera.`**`create`**`(`*parent_entity*`)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`Camera.`**`destroy`**`(`*component*`)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

`Camera.`**`commit_changes`**`(`*component*`)`

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

Camera.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

Camera.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

Camera.**is_alive**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

Camera.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

Camera.**get_property**(*component*, *property_name*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

Camera.**set_property**(*component*, *property_name*, *value*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

Camera.**list_properties**()

Returns

- **value** (table) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

10.1.32 Script Component

Script.**load**()

Loads all functions and types for this scripting interface.

Script.**get_type_id**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

Script.**create**(*parent_entity*)

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

Script.**destroy**(*component*)

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

Script.**commit_changes**(*component*)

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

Script.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

Script.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

`Script.is_alive(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`boolean`) - True if the component is alive.

Returns true if the referenced component is alive.

`Script.get_entity(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

`Script.get_property(component, property_name)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`Script.set_property(component, property_name, value)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`Script.list_properties()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

10.1.33 Voxel Shape Component

`VoxelShape.load()`

Loads all functions and types for this scripting interface.

`VoxelShape.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

`VoxelShape.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`VoxelShape.destroy(component)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

`VoxelShape.commit_changes(component)`

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

`VoxelShape.get_num_active_components()`

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

`VoxelShape.get_component_for_entity(entity)`

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

`VoxelShape.is_alive(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`boolean`) - True if the component is alive.

Returns true if the referenced component is alive.

`VoxelShape.`**`get_entity`**`(`*component*`)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

`VoxelShape.`**`get_property`**`(`*component*, *property_name*`)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`VoxelShape.`**`set_property`**`(`*component*, *property_name*, *value*`)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`VoxelShape.`**`list_properties`**`()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

`VoxelShape.`**`set`**`(`*component*, *coord*, *palette_index*`)`

Params

- **component** (*Ref*) - The voxel shape component.
- **coord** (*U8Vec3*) - The coordinate of the voxel.
- **palette_index** (`number`) - The palette index to set.

Sets the voxel to the given palette index.

`VoxelShape.`**`set_unsafe`**`(`*component*, *coord*, *palette_index*`)`

Params

- **component** (*Ref*) - The voxel shape component.
- **coord** (*U8Vec3*) - The coordinate of the voxel.

- **palette_index** (number) - The palette index to set.

Sets the voxel to the given palette index (without boundary checks).

`VoxelShape.get(component, coord)`

Params

- **component** (*Ref*) - The voxel shape component.
- **coord** (*U8Vec3*) - The coordinate of the voxel.

Returns

- **value** (number) - The palette index of the given voxel.

Gets the palette index for a voxel.

`VoxelShape.fill(component, min, max, palette_index)`

Params

- **component** (*Ref*) - The voxel shape component.
- **min** (*U8Vec3*) - The min voxel coorindate of the area.
- **max** (*U8Vec3*) - The max voxel coordinate of the area.
- **palette_index** (number) - The palette index to set.

Fills all voxels in the range defined by min and max.

`VoxelShape.copy(target, source)`

Params

- **target** (*Ref*) - The target voxel shape component.
- **source** (*Ref*) - The source voxel shape component.

Copies the given source shape into the target shape.

`VoxelShape.get_dim(component)`

Params

- **component** (*Ref*) - The voxel shape component.

Returns

- **value** (*U16Vec3*) - The dimensions of the shape in voxels.

Gets the dimensions of the shape in voxels.

`VoxelShape.voxelize(component)`

Params

- **component** (*Ref*) - The voxel shape component.

Queues this shape for voxelization

`VoxelShape.apply_force_at_world_position(component, force, position)`

Params

- **component** (*Ref*) - The voxel shape component.
- **force** (*Vec3*) - The force vector to apply.

- **position** (*Vec3*) - The position to apply the force at (world coordinates).

Applies a force at the given world position.

`VoxelShape.apply_force_at_local_position(component, force, position)`

Params

- **component** (*Ref*) - The voxel shape component.
- **force** (*Vec3*) - The force vector to apply.
- **position** (*Vec3*) - The position to apply the force at (local coordinates).

Applies a force at the given local position.

`VoxelShape.apply_force(component, force)`

Params

- **component** (*Ref*) - The voxel shape component.
- **force** (*Vec3*) - The force vector to apply.

Applies a force at the center of mass.

`VoxelShape.apply_torque(component, torque)`

Params

- **component** (*Ref*) - The voxel shape component.
- **torque** (*Vec3*) - The torque vector to apply.

Applies a torque at the center of mass.

`VoxelShape.get_linear_velocity(component)`

Params

- **component** (*Ref*) - The voxel shape component.

Returns

- **value** (*Vec3*) - The linear velocity of the shape.

Gets the linear velocity of the shape.

`VoxelShape.set_linear_velocity(component, velocity)`

Params

- **component** (*Ref*) - The voxel shape component.
- **velocity** (*Vec3*) - The linear velocity to set.

Sets the linear velocity of the shape.

`VoxelShape.get_angular_velocity(component)`

Params

- **component** (*Ref*) - The voxel shape component.

Returns

- **value** (*Vec3*) - The angular velocity of the shape.

Gets the angular velocity of the shape.

VoxelShape.**set_angular_velocity**(*component*, *velocity*)

Params

- **component** (*Ref*) - The voxel shape component.
- **velocity** (*Vec3*) - The angular velocity to set.

Sets the angular velocity of the shape.

10.1.34 Light Component

Light.**load**()

Loads all functions and types for this scripting interface.

Light.**get_type_id**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

Light.**create**(*parent_entity*)

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

Light.**destroy**(*component*)

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

Light.**commit_changes**(*component*)

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

Light.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

Light.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

`Light.`**`is_alive`**`(`*component*`)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`boolean`) - True if the component is alive.

Returns true if the referenced component is alive.

`Light.`**`get_entity`**`(`*component*`)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

`Light.`**`get_property`**`(`*component*, *property_name*`)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`Light.`**`set_property`**`(`*component*, *property_name*, *value*`)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`Light.`**`list_properties`**`()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

10.1.35 Node Component

`Node.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

`Node.get_num_active_components()`

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

`Node.get_component_for_entity(entity)`

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

`Node.is_alive(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

`Node.get_entity(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

`Node.create(name)`

Params

- **name** (string) - The name of the new node component.

Returns

- **value** (*Ref*) - The new node component.

Creates a new node component with the given name and attaches it to the root node of the world.

`Node.create(parent_node, name)`

Params

- **parent_node** (*Ref*) - The node to attach the new node to.
- **name** (`string`) - The name of the new node component.

Returns

- **value** (*Ref*) - The new node component.

Creates a new node component with the given name and attaches it to the provided node.

`Node.destroy(node)`

Params

- **node** (*Ref*) - The node to destroy.

Destroys the given node component.

`Node.get_parent(node)`

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Ref*) - The parent of the node (if any).

Gets the parent node (if any).

`Node.get_next_sibling(node)`

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Ref*) - The next sibling of the node (if any).

Gets the next sibling of the node (if any).

`Node.get_prev_sibling(node)`

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Ref*) - The previous sibling of the node (if any).

Gets the previous sibling of the node (if any).

`Node.set_hidden(node, hidden)`

Params

- **node** (*Ref*) - The node in question.
- **hidden** (`boolean`) - Pass true if the node should be hidden.

Sets the hidden state of the node.

Node.**is_hidden**(*node*)

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (boolean) - True if the node is hidden.

Retrieves the hidden state of the node.

Node.**get_position**(*node*)

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Vec3*) - The (local) position of the node.

Gets the (local) position of the node.

Node.**get_world_position**(*node*)

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Vec3*) - The (world) position of the node.

Gets the (world) position of the node.

Node.**get_orientation**(*node*)

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Quat*) - The (local) orientation of the node.

Gets the (local) orientation of the node.

Node.**get_world_orientation**(*node*)

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Quat*) - The (world) orientation of the node.

Gets the (world) orientation of the node.

Node.**get_size**(*node*)

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Vec3*) - The (local) size of the node.

Gets the (local) size of the node.

`Node.get_world_size(node)`

Params

- **node** (*Ref*) - The node in question.

Returns

- **value** (*Vec3*) - The (world) size of the node.

Gets the (world) size of the node.

`Node.set_position(node, position)`

Params

- **node** (*Ref*) - The node in question.
- **position** (*Vec3*) - The (local) position of the node.

Sets the (local) position of the node.

`Node.set_world_position(node, position)`

Params

- **node** (*Ref*) - The node in question.
- **position** (*Vec3*) - The (world) position of the node.

Sets the (world) position of the node.

`Node.set_orientation(node, orientation)`

Params

- **node** (*Ref*) - The node in question.
- **orientation** (*Quat*) - The (local) orientation of the node.

Sets the (local) orientation of the node.

`Node.set_world_orientation(node, orientation)`

Params

- **node** (*Ref*) - The node in question.
- **orientation** (*Quat*) - The (world) orientation of the node.

Sets the (world) orientation of the node.

`Node.set_size(node, size)`

Params

- **node** (*Ref*) - The node in question.
- **size** (*Vec3*) - The (local) size of the node.

Sets the (local) size of the node.

`Node.set_world_size(node, size)`

Params

- **node** (*Ref*) - The node in question.

- **size** (*Vec3*) - The (world) size of the node.

Sets the (world) size of the node.

`Node.to_local_space(node, position)`

Params

- **node** (*Ref*) - The node used for the transformation.
- **position** (*Vec3*) - The position to transform.

Returns

- **value** (*Vec3*) - The position transformed into the local node space.

Transforms the provided position into the local space of the given node.

`Node.to_local_space_direction(node, position)`

Params

- **node** (*Ref*) - The node used for the transformation.
- **position** (*Vec3*) - The direction to transform.

Returns

- **value** (*Vec3*) - The direction transformed into the local node space.

Transforms the provided direction into the local space of the given node.

`Node.to_world_space(node, position)`

Params

- **node** (*Ref*) - The node used for the transformation.
- **position** (*Vec3*) - The position to transform.

Returns

- **value** (*Vec3*) - The position transformed into world space.

Transforms the provided position in the local space of the given node to world space.

`Node.to_world_space_direction(node, direction)`

Params

- **node** (*Ref*) - The node used for the transformation.
- **direction** (*Vec3*) - The direction to transform.

Returns

- **value** (*Vec3*) - The direction transformed into world space.

Transforms the provided direction in the local space of the given node to world space.

`Node.collect_nodes_depth_first(node)`

Params

- **node** (*Ref*) - The root node to start collecting at.

Returns

- **value** (`table`) - Table containig all nodes in the hierarchy.

Collects all nodes in the hierarchy in depth first ordering starting at the provided node (including the root).

`Node.collect_nodes_breadth_first(node)`

Params

- **node** (*Ref*) - The root node to start at.

Returns

- **value** (`table`) - Table containig all nodes in the hierarchy.

Collects all nodes in the hierarchy in breadth first ordering starting at the provided node (including the root).

`Node.update_transforms(node)`

Params

- **node** (*Ref*) - The root node of the hierarchy.

Updates the transformations of the given node hierarchy.

10.1.36 Character Controller Component

`CharacterController.load()`

Loads all functions and types for this scripting interface.

`CharacterController.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (`number`) - The type ID of the component.

Returns the type ID of the component.

`CharacterController.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`CharacterController.destroy(component)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

`CharacterController.commit_changes(component)`

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

CharacterController.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

CharacterController.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

CharacterController.**is_alive**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

CharacterController.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

CharacterController.**get_property**(*component*, *property_name*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

CharacterController.**set_property**(*component*, *property_name*, *value*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

CharacterController.**list_properties**()

Returns

- **value** (table) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

CharacterController.**move**(*component*, *move*)

Params

- **component** (*Ref*) - The controller in question.
- **move** (*Vec3*) - The movement to apply to the character controller in world space.

Moves the character controller.

CharacterController.**is_grounded**(*component*)

Params

- **component** (*Ref*) - The controller in question.

Returns

- **value** (boolean) - True if the controller is grounded.

Returns true if the character controller is grounded.

CharacterController.**is_colliding_sides**(*component*)

Params

- **component** (*Ref*) - The controller in question.

Returns

- **value** (boolean) - True if the controller is colliding with its sides.

Returns true if the controller is colliding with its sides.

CharacterController.**is_colliding_up**(*component*)

Params

- **component** (*Ref*) - The controller in question.

Returns

- **value** (boolean) - True if the top of the controller is colliding.

Returns true if the upper part of the controller is colliding.

10.1.37 Camera Controller Component

CameraController.**load**()

Loads all functions and types for this scripting interface.

CameraController.**get_type_id**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

CameraController.**create**(*parent_entity*)

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

CameraController.**destroy**(*component*)

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

CameraController.**commit_changes**(*component*)

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

CameraController.**get_num_active_components**()

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

CameraController.**get_component_for_entity**(*entity*)

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

CameraController.**is_alive**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

CameraController.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

`CameraController.get_property(component, property_name)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

`CameraController.set_property(component, property_name, value)`

Params

- **component** (*Ref*) - The component.
- **property_name** (`string`) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

`CameraController.list_properties()`

Returns

- **value** (`table`) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

`CameraController.set_target_node(component, node)`

Params

- **component** (*Ref*) - The controller in question.
- **node** (*Ref*) - The node the controller should track.

Sets the node the controller should track.

`CameraController.set_target_euler_angles(component, angles)`

Params

- **component** (*Ref*) - The controller in question.
- **angles** (*Vec3*) - The target euler angles in radians.

Sets the target Euler angles.

`CameraController.get_target_euler_angles(component)`

Params

- **component** (*Ref*) - The controller in question.

Returns

- **value** (*Vec3*) - The current target euler angles in radians.

Gets the target Euler angles.

10.1.38 Particle Component

`Particle.load()`

Loads all functions and types for this scripting interface.

`Particle.get_type_id(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (number) - The type ID of the component.

Returns the type ID of the component.

`Particle.create(parent_entity)`

Params

- **parent_entity** (*Ref*) - The parent entity.

Creates a new component and attaches it to the provided parent entity.

`Particle.destroy(component)`

Params

- **component** (*Ref*) - The component to destroy.

Destroys the provided component.

`Particle.commit_changes(component)`

Params

- **component** (*Ref*) - The component to update.

Commits all changes to the properties of the provided component.

`Particle.get_num_active_components()`

Returns

- **value** (number) - Total total number of active components of this type.

Returns the total number of active components of this type.

`Particle.get_component_for_entity(entity)`

Params

- **entity** (*Ref*) - The entity.

Returns

- **value** (*Ref*) - The component for the given entity.

Returns the component for the given entity.

`Particle.is_alive(component)`

Params

- **component** (*Ref*) - The component.

Returns

- **value** (boolean) - True if the component is alive.

Returns true if the referenced component is alive.

Particle.**get_entity**(*component*)

Params

- **component** (*Ref*) - The component.

Returns

- **value** (*Ref*) - The entity the component is assigned to.

Returns the entity the component is assigned to.

Particle.**get_property**(*component*, *property_name*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to retrieve.

Returns

- **value** (*Variant*) - The property as a variant.

Returns the requested property as a variant.

Particle.**set_property**(*component*, *property_name*, *value*)

Params

- **component** (*Ref*) - The component.
- **property_name** (string) - The name of the property to set.
- **value** (*Variant*) - The value to set.

Sets the requested property to the provided variant value.

Particle.**list_properties**()

Returns

- **value** (table) - Table containing the names and types of all the exposed properties.

Lists all properties exposed by this component interface.

Particle.**get_emitter_handle**(*component*)

Params

- **component** (*Ref*) - The particle in question.

Returns

- **value** (*Handle*) - The emitter handle of this particle.

Gets the emitter handle.

10.2 Native C API

Note: The C API is available as a single documented header file in our public GitHub repository.

Writing your first native plugin for IOLITE is easy. For further instructions, please refer to the guide *Writing plugins using the native C/C++ API*.

```
// MIT License
//
// Copyright (c) 2023 Missing Deadlines (Benjamin Wrensch)
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

//-----------------------------------------------------------------------------//
// Minimal C/C++ plugin sample
//-----------------------------------------------------------------------------//

/*
#include "iolite_api.h"

const struct io_api_manager_i* io_api_manager = 0;

IO_API_EXPORT io_uint32_t IO_API_CALL get_api_version()
{
  // Inform IOLITE which version of the API you are using
  return IO_API_VERSION;
}

IO_API_EXPORT io_int32_t IO_API_CALL load_plugin(const void* api_manager)
{
  // Ensure we can keep accessing the API manager after loading the plugin
  io_api_manager = (const struct io_api_manager_i*)api_manager;

  // Do something with the API manager, set up your plugin, etc.
```

(continues on next page)

(continued from previous page)

```
  return 0; // Return a value < 0 to indicate that the loading of your plugin
            // has failed (depedency not available, etc.)
}

IO_API_EXPORT void IO_API_CALL unload_plugin()
{
  // Clean up here
}
*/

#ifndef INCLUDE_IO_API
#define INCLUDE_IO_API

//-----------------------------------------------------------------------------//
// IOLITE API version
//-----------------------------------------------------------------------------//

// The major, minor, and bug fix parts of the IOLITE API version
//-----------------------------------------------------------------------------//
#define IO_API_VERSION_MAJOR 0
#define IO_API_VERSION_MINOR 4
#define IO_API_VERSION_BUGFIX 0

// The version of the IOLITE API as a single number
//-----------------------------------------------------------------------------//
#define IO_API_VERSION                                                         \
  ((IO_API_VERSION_MAJOR << 24) + (IO_API_VERSION_MINOR << 16) +               \
   (IO_API_VERSION_BUGFIX << 8) + 0)

// Helpers for retrieving the major, minor, and bug fix part from the version
// number
//-----------------------------------------------------------------------------//
#define IO_API_VERSION_GET_MAJOR(_version) ((_version >> 24) & 0xFFu)
#define IO_API_VERSION_GET_MINOR(_version) ((_version >> 16) & 0xFFu)
#define IO_API_VERSION_GET_BUGFIX(_version) ((_version >> 8) & 0xFFu)

//-----------------------------------------------------------------------------//
// DLL export helpers
//-----------------------------------------------------------------------------//

//-----------------------------------------------------------------------------//
#ifdef _WIN32
//-----------------------------------------------------------------------------//
#define IO_API_CALL __stdcall
//-----------------------------------------------------------------------------//
#ifdef __cplusplus
//-----------------------------------------------------------------------------//
#define IO_API_EXPORT extern "C" __declspec(dllexport)
//-----------------------------------------------------------------------------//
#else // __cplusplus
//-----------------------------------------------------------------------------//
#define IO_API_EXPORT __declspec(dllexport)
```

(continues on next page)

(continued from previous page)

```
//-----------------------------------------------------------------------------//
#endif // __cplusplus
//-----------------------------------------------------------------------------//
#else // _WIN32
//-----------------------------------------------------------------------------//
#ifdef __cplusplus
//-----------------------------------------------------------------------------//
#define IO_API_EXPORT extern "C"
//-----------------------------------------------------------------------------//
#else // __cplusplus
//-----------------------------------------------------------------------------//
#define IO_API_EXPORT
//-----------------------------------------------------------------------------//
#endif
//-----------------------------------------------------------------------------//
#define IO_API_CALL
//-----------------------------------------------------------------------------//
#endif // _WIN32
//-----------------------------------------------------------------------------//

//-----------------------------------------------------------------------------//
// Basic types
//-----------------------------------------------------------------------------//

//-----------------------------------------------------------------------------//
#define IO_TRUE 1
#define IO_FALSE 0
//-----------------------------------------------------------------------------//
#ifndef __cplusplus
//-----------------------------------------------------------------------------//
typedef unsigned char io_bool_t;
#else
typedef bool io_bool_t;
//-----------------------------------------------------------------------------//
#endif
//-----------------------------------------------------------------------------//

//-----------------------------------------------------------------------------//
typedef unsigned char io_uint8_t;
typedef char io_int8_t;

//-----------------------------------------------------------------------------//
typedef unsigned short io_uint16_t;
typedef short io_int16_t;

//-----------------------------------------------------------------------------//
typedef unsigned int io_uint32_t;
typedef int io_int32_t;

//-----------------------------------------------------------------------------//
typedef unsigned long long io_uint64_t;
typedef long long io_int64_t;
```

(continues on next page)

(continued from previous page)

```
//--------------------------------------------------------------------------------//
typedef float io_float32_t;
typedef double io_float64_t;

//--------------------------------------------------------------------------------//
// Math related types
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_float32_t x, y;
    };
    struct
    {
      io_float32_t r, g;
    };
    io_float32_t data[2u];
  };
} io_vec2_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_float32_t x, y, z;
    };
    struct
    {
      io_float32_t r, g, b;
    };
    io_float32_t data[3u];
  };
} io_vec3_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_float32_t x, y, z, w;
    };
```

(continues on next page)

(continued from previous page)

```
    struct
    {
      io_float32_t r, g, b, a;
    };
    io_float32_t data[4u];
  };
} io_vec4_t;

//------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_float32_t w, x, y, z;
    };
    io_float32_t data[4u];
  };
} io_quat_t;

//------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_int32_t x, y;
    };
    io_int32_t data[2u];
  };
} io_ivec2_t;

//------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_int32_t x, y, z;
    };
    io_int32_t data[3u];
  };
} io_ivec3_t;

//------------------------------------------------------------------------------//
typedef struct
{
  union
  {
```

(continues on next page)

(continued from previous page)

```
    struct
    {
      io_int32_t x, y, z, w;
    };
    io_int32_t data[4u];
  };
} io_ivec4_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_uint32_t x, y;
    };
    io_uint32_t data[2u];
  };
} io_uvec2_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_uint32_t x, y, z;
    };
    io_uint32_t data[3u];
  };
} io_uvec3_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_uint8_t x, y, z;
    };
    io_uint8_t data[3u];
  };
} io_u8vec3_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
```

(continues on next page)

(continued from previous page)

```
    struct
    {
      io_uint16_t x, y, z;
    };
    io_uint16_t data[3u];
  };
} io_u16vec3_t;

//--------------------------------------------------------------------------------//
typedef struct
{
  union
  {
    struct
    {
      io_uint32_t x, y, z, w;
    };
    io_uint32_t data[4u];
  };
} io_uvec4_t;

//--------------------------------------------------------------------------------//
// Math type conversion helpers
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_VEC2_TYPE
//--------------------------------------------------------------------------------//
inline io_vec2_t io_cvt(IO_USER_VEC2_TYPE v) { return {v.x, v.y}; }
inline IO_USER_VEC2_TYPE io_cvt(io_vec2_t v) { return {v.x, v.y}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_VEC2_TYPE
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_VEC3_TYPE
//--------------------------------------------------------------------------------//
inline io_vec3_t io_cvt(IO_USER_VEC3_TYPE v) { return {v.x, v.y, v.z}; }
inline IO_USER_VEC3_TYPE io_cvt(io_vec3_t v) { return {v.x, v.y, v.z}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_VEC3_TYPE
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_VEC4_TYPE
//--------------------------------------------------------------------------------//
inline io_vec4_t io_cvt(IO_USER_VEC4_TYPE v) { return {v.x, v.y, v.z, v.w}; }
inline IO_USER_VEC4_TYPE io_cvt(io_vec4_t v) { return {v.x, v.y, v.z, v.w}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_VEC4_TYPE
//--------------------------------------------------------------------------------//
```

(continues on next page)

(continued from previous page)

```
//--------------------------------------------------------------------------------//
#ifdef IO_USER_QUAT_TYPE
//--------------------------------------------------------------------------------//
inline io_quat_t io_cvt(IO_USER_QUAT_TYPE v) { return {v.w, v.x, v.y, v.z}; }
//--------------------------------------------------------------------------------//
#ifndef IO_USER_QUAT_TYPE_W_LAST
//--------------------------------------------------------------------------------//
inline IO_USER_QUAT_TYPE io_cvt(io_quat_t v) { return {v.w, v.x, v.y, v.z}; }
//--------------------------------------------------------------------------------//
#else
//--------------------------------------------------------------------------------//
inline IO_USER_QUAT_TYPE io_cvt(io_quat_t v) { return {v.x, v.y, v.z, v.w}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_QUAT_TYPE_W_LAST
//--------------------------------------------------------------------------------//
#endif // IO_USER_QUAT_TYPE
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_UVEC2_TYPE
//--------------------------------------------------------------------------------//
inline io_uvec2_t io_cvt(IO_USER_UVEC2_TYPE v) { return {v.x, v.y}; }
inline IO_USER_UVEC2_TYPE io_cvt(io_uvec2_t v) { return {v.x, v.y}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_UVEC2_TYPE
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_UVEC3_TYPE
//--------------------------------------------------------------------------------//
inline io_uvec3_t io_cvt(IO_USER_UVEC3_TYPE v) { return {v.x, v.y, v.z}; }
inline IO_USER_UVEC3_TYPE io_cvt(io_uvec3_t v) { return {v.x, v.y, v.z}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_UVEC3_TYPE
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_U8VEC3_TYPE
//--------------------------------------------------------------------------------//
inline io_u8vec3_t io_cvt(IO_USER_U8VEC3_TYPE v) { return {v.x, v.y, v.z}; }
inline IO_USER_U8VEC3_TYPE io_cvt(io_u8vec3_t v) { return {v.x, v.y, v.z}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_U8VEC3_TYPE
//--------------------------------------------------------------------------------//

//--------------------------------------------------------------------------------//
#ifdef IO_USER_U16VEC3_TYPE
//--------------------------------------------------------------------------------//
inline io_u16vec3_t io_cvt(IO_USER_U16VEC3_TYPE v) { return {v.x, v.y, v.z}; }
inline IO_USER_U16VEC3_TYPE io_cvt(io_u16vec3_t v) { return {v.x, v.y, v.z}; }
//--------------------------------------------------------------------------------//
#endif // IO_USER_U16VEC3_TYPE
```

(continues on next page)

(continued from previous page)

```
//------------------------------------------------------------------------------//

//------------------------------------------------------------------------------//
#ifdef IO_USER_UVEC4_TYPE
//------------------------------------------------------------------------------//
inline io_uvec4_t io_cvt(IO_USER_UVEC4_TYPE v) { return {v.x, v.y, v.z, v.w}; }
inline IO_USER_UVEC4_TYPE io_cvt(io_uvec4_t v) { return {v.x, v.y, v.z, v.w}; }
//------------------------------------------------------------------------------//
#endif // IO_USER_UVEC4_TYPE
//------------------------------------------------------------------------------//

//------------------------------------------------------------------------------//
#ifdef IO_USER_IVEC2_TYPE
//------------------------------------------------------------------------------//
inline io_ivec2_t io_cvt(IO_USER_IVEC2_TYPE v) { return {v.x, v.y}; }
inline IO_USER_IVEC2_TYPE io_cvt(io_ivec2_t v) { return {v.x, v.y}; }
//------------------------------------------------------------------------------//
#endif // IO_USER_IVEC2_TYPE
//------------------------------------------------------------------------------//

//------------------------------------------------------------------------------//
#ifdef IO_USER_IVEC3_TYPE
//------------------------------------------------------------------------------//
inline io_ivec3_t io_cvt(IO_USER_IVEC3_TYPE v) { return {v.x, v.y, v.z}; }
inline IO_USER_IVEC3_TYPE io_cvt(io_ivec3_t v) { return {v.x, v.y, v.z}; }
//------------------------------------------------------------------------------//
#endif // IO_USER_IVEC3_TYPE
//------------------------------------------------------------------------------//

//------------------------------------------------------------------------------//
#ifdef IO_USER_IVEC4_TYPE
//------------------------------------------------------------------------------//
inline io_ivec4_t io_cvt(IO_USER_IVEC4_TYPE v) { return {v.x, v.y, v.z, v.w}; }
inline IO_USER_IVEC4_TYPE io_cvt(io_ivec4_t v) { return {v.x, v.y, v.z, v.w}; }
//------------------------------------------------------------------------------//
#endif // IO_USER_IVEC4_TYPE
//------------------------------------------------------------------------------//

//------------------------------------------------------------------------------//
// Loosely typed enums and flags
//------------------------------------------------------------------------------//

// Flags defining the faces of a box
//------------------------------------------------------------------------------//
enum io_box_face_flags_
{
  io_box_face_flags_front = 0x01u,
  io_box_face_flags_back = 0x02u,
  io_box_face_flags_top = 0x04u,
  io_box_face_flags_bottom = 0x08u,
  io_box_face_flags_left = 0x10u,
  io_box_face_flags_right = 0x20u,
```

(continues on next page)

(continued from previous page)

```
  io_box_face_flags_all = 0x7Fu
};
typedef io_uint8_t io_box_face_flags;

// Indices for the faces of a box matching the flags above
//------------------------------------------------------------------------------//
enum io_box_face_index_
{
  io_box_face_index_front,  // Z + 1
  io_box_face_index_back,   // Z - 1
  io_box_face_index_top,    // Y + 1
  io_box_face_index_bottom, // Y - 1
  io_box_face_index_left,   // X - 1
  io_box_face_index_right   // X + 1
};
typedef io_uint8_t io_box_face_index;

// Flags for configuring properties for custom components
//------------------------------------------------------------------------------//
enum io_property_flags_
{
  io_property_flags_runtime_only =
      0x01u // Indicates that the given property should neither be serialized
            // nor exposed in the editor
};
typedef io_uint8_t io_property_flags;

// Vertical text alignment
//------------------------------------------------------------------------------//
enum io_ui_text_align_vertical_
{
  io_ui_text_align_vertical_top,    // Position text at top
  io_ui_text_align_vertical_center, // Position text at vertical center
  io_ui_text_align_vertical_bottom  // Position text at bottom
};
typedef io_uint32_t io_ui_text_align_vertical;

// Horizontal text alignment
//------------------------------------------------------------------------------//
enum io_ui_text_align_horizontal_
{
  io_ui_text_align_horizontal_left,   // Position text left
  io_ui_text_align_horizontal_center, // Position text at horizontal center
  io_ui_text_align_horizontal_right   // Position text right
};
typedef io_uint32_t io_ui_text_align_horizontal;

// Various presets for anchors
//------------------------------------------------------------------------------//
enum io_ui_anchor_preset_
{
```

(continues on next page)

(continued from previous page)

```
  io_ui_anchor_preset_full_rect,     // A full rectangle
  io_ui_anchor_preset_top_left,      // Anchor at top left
  io_ui_anchor_preset_top_right,     // Anchor at top right
  io_ui_anchor_preset_bottom_right,  // Anchor at bottom right
  io_ui_anchor_preset_bottom_left,   // Anchor at bottom left
  io_ui_anchor_preset_center_left,   // Anchor at the left center
  io_ui_anchor_preset_center_top,    // Anchor at the top center
  io_ui_anchor_preset_center_right,  // Anchor at the right center
  io_ui_anchor_preset_center_bottom, // Anchor at the bottom center
  io_ui_anchor_preset_center,        // Anchor at the center

  io_ui_anchor_preset_num
};
typedef io_uint32_t io_ui_anchor_preset;

//------------------------------------------------------------------------------//
enum io_ui_aspect_mode_
{
  io_ui_aspect_mode_keep // Keep the aspect ratio (using letter and pillar
                         // boxing)
};
typedef io_uint32_t io_ui_aspect_mode;

//------------------------------------------------------------------------------//
enum io_ui_text_flag_
{
  io_ui_text_flag_wrap = 0x01u // Wrap text
};
typedef io_uint32_t io_ui_text_flag;

//------------------------------------------------------------------------------//
enum io_ui_style_var_
{
  io_ui_style_var_text_color,         // Color for text (vec4)
  io_ui_style_var_text_outline_color, // Color for text outlines (vec4)
  io_ui_style_var_text_outline, // The width (in px) of the text outline (float)
                                // Set to > 0 to add outlines to text
  io_ui_style_var_rect_rounding, // Value in [0, 1] that defines the rounding
                                 // radius. Set to > 0 to draw rectangles with
                                 // rounded corners (float)
  io_ui_style_var_draw_outline,  // Draw outlines with the given width (in px)
                                 // instead of filled shapes
  io_ui_style_var_alpha // Global alpha value applied to all draw operations
                        // (float)
};
typedef io_uint32_t io_ui_style_var;

// Flags used to configure the radius damage applied to the world
//------------------------------------------------------------------------------//
enum io_world_radius_damage_flags_
{
  io_radius_damage_flags_shade_crater =
```

(continues on next page)

(continued from previous page)

```
      0x01u, // Shade the crater around the area affected by the radius damage
  io_radius_damage_flags_fracture =
      0x02u // Apply radius damage as fracture damage
};
typedef io_uint32_t io_world_radius_damage_flags;

// Input key state
//--------------------------------------------------------------------------------//
enum io_input_key_state_
{
  io_input_key_state_released,
  io_input_key_state_pressed,
  io_input_key_state_clicked
};
typedef io_uint8_t io_input_key_state;

// Input axis
//--------------------------------------------------------------------------------//
enum io_input_axis_
{
  io_input_axis_left_x,
  io_input_axis_left_y,
  io_input_axis_right_x,
  io_input_axis_right_y,

  io_input_axis_trigger_left,
  io_input_axis_trigger_right,

  io_input_axis_invalid
};
typedef io_uint8_t io_input_axis;

// Input key
//--------------------------------------------------------------------------------//
enum io_input_key_
{
  io_input_key_up,
  io_input_key_down,
  io_input_key_left,
  io_input_key_right,

  io_input_key_a,
  io_input_key_b,
  io_input_key_c,
  io_input_key_d,
  io_input_key_e,
  io_input_key_f,
  io_input_key_g,
  io_input_key_h,
  io_input_key_i,
  io_input_key_j,
  io_input_key_k,
```

(continues on next page)

(continued from previous page)

```
    io_input_key_l,
    io_input_key_m,
    io_input_key_n,
    io_input_key_o,
    io_input_key_p,
    io_input_key_q,
    io_input_key_r,
    io_input_key_s,
    io_input_key_t,
    io_input_key_u,
    io_input_key_v,
    io_input_key_w,
    io_input_key_x,
    io_input_key_y,
    io_input_key_z,

    io_input_key_0,
    io_input_key_1,
    io_input_key_2,
    io_input_key_3,
    io_input_key_4,
    io_input_key_5,
    io_input_key_6,
    io_input_key_7,
    io_input_key_8,
    io_input_key_9,

    io_input_key_f1,
    io_input_key_f2,
    io_input_key_f3,
    io_input_key_f4,
    io_input_key_f5,
    io_input_key_f6,
    io_input_key_f7,
    io_input_key_f8,
    io_input_key_f9,
    io_input_key_f10,
    io_input_key_f11,
    io_input_key_f12,

    io_input_key_del,
    io_input_key_backspace,
    io_input_key_tab,

    io_input_key_mouse_left,
    io_input_key_mouse_right,
    io_input_key_mouse_middle,

    io_input_key_shift,
    io_input_key_alt,
    io_input_key_ctrl,
```

(continues on next page)

(continued from previous page)

```
  io_input_key_space,
  io_input_key_escape,
  io_input_key_return,

  io_input_key_num_plus,
  io_input_key_num_minus,
  io_input_key_num_0,
  io_input_key_num_1,
  io_input_key_num_2,
  io_input_key_num_3,
  io_input_key_num_4,
  io_input_key_num_5,
  io_input_key_num_6,
  io_input_key_num_7,
  io_input_key_num_8,
  io_input_key_num_9,

  io_input_key_controller_button_a,
  io_input_key_controller_button_y,
  io_input_key_controller_button_b,
  io_input_key_controller_button_x,

  io_input_key_any,
  io_input_key_invalid,

  io_input_key_num_keys
};
typedef io_uint8_t io_input_key;

//------------------------------------------------------------------------------//
// Custom types
//------------------------------------------------------------------------------//

// Ref type
//------------------------------------------------------------------------------//
typedef struct
{
  io_uint32_t internal;
} io_ref_t;

// Name type
//------------------------------------------------------------------------------//
typedef struct
{
  io_uint32_t hash;
} io_name_t;

// 16-bit handle type
//------------------------------------------------------------------------------//
typedef struct
{
  io_uint16_t internal;
```

(continues on next page)

(continued from previous page)

```
} io_handle16_t;

// 32-bit handle type
//-----------------------------------------------------------------------------//
typedef struct
{
  io_uint32_t internal;
} io_handle32_t;

// 64-bit handle type
//-----------------------------------------------------------------------------//
typedef struct
{
  io_uint64_t internal;
} io_handle64_t;

// Variant type
//-----------------------------------------------------------------------------//
typedef struct
{
  io_name_t type;
  io_uint32_t data[4u];
} io_variant_t;

// Property description type
//-----------------------------------------------------------------------------//
typedef struct
{
  const char* name;
  const char* type;
  io_property_flags flags;
} io_property_desc_t;

// Callback function for scheduler tasks
//-----------------------------------------------------------------------------//
typedef void (*io_scheduler_callback_t)(io_uvec2_t range, io_uint32_t thread_id,
                                        io_uint32_t sub_task_index, void* task);

// Scheduler tasks provide the possibility to (evenly) spread workloads to a set
// of worker threads provided by the internal task scheduler.
//   Example:
//     If you provide a task with the number of workloads set to 16 and there
//     are 4 available hardware threads, the workload gets divided into 4
//     sub-tasks with a range in [0, 4]
//-----------------------------------------------------------------------------//
typedef struct
{
  // The number of workloads this task should handle
  io_uint32_t num_workloads;

  // The minimum amount of sub-tasks to spawn per worker thread
  io_uint32_t min_sub_tasks_per_worker;
```

(continues on next page)

(continued from previous page)

```
  // The target of sub-tasks that should be created per worker thread
  io_uint32_t target_sub_tasks_per_worker;

  // The number of currently active sub-tasks. Initialize to zero
  io_uint64_t running_sub_task_count;

  // Callback function called for each of the internal sub-tasks.
  io_scheduler_callback_t callback;
} io_scheduler_task_t;

//-----------------------------------------------------------------------------//
// Global helper functions and types
//-----------------------------------------------------------------------------//

// Initializes a task with the given callback for the given amount of tasks
//-----------------------------------------------------------------------------//
inline void io_init_scheduler_task(io_scheduler_task_t* task,
                                   io_uint32_t num_tasks,
                                   io_scheduler_callback_t callback)
{
  // Defaults
  task->target_sub_tasks_per_worker = 4u;
  task->min_sub_tasks_per_worker = 0u;
  task->running_sub_task_count = 0u;

  task->num_workloads = num_tasks;
  task->callback = callback;
}

// Fixed time step accumulator
//
// Example usage:
//
//  // Do this once to initialize the accumulator
//  static io_fixed_step_accumulator accum;
//  io_init(60.0f, &accum);
//
//  // Do this every frame
//  io_accumulator_add(delta_t, &accum);
//
//  while (io_accumulator_step(&accum))
//    pos += vel * accum.delta_t;
//-----------------------------------------------------------------------------//
typedef struct
{
  io_float32_t update_frequency_in_hz; // The fixed update frequency in Hz
                                       // (steps per second).
  io_float32_t delta_t;                // The delta time (in seconds).
  io_float32_t
      interpolator; // The interpolation factor that needs to be applied to,
                    // e.g., smoothen the visual representation.
  io_float32_t accumulator; // The accumulated time.
```

(continues on next page)

(continued from previous page)

```
} io_fixed_step_accumulator_t;

// Initializes the provided accumulator. Call this function once when creating a
// new accumulator.
//-----------------------------------------------------------------------------//
inline void
io_init_fixed_step_accumulator(io_fixed_step_accumulator_t* accumulator,
                               io_float32_t update_frequency_in_hz)
{
  accumulator->update_frequency_in_hz = update_frequency_in_hz;
  accumulator->delta_t = 1.0f / accumulator->update_frequency_in_hz;
  accumulator->accumulator = 0.0f;
  accumulator->interpolator = 0.0f;
}

// Call this function once per frame for each accumulator providing the variable
// frame delta time.
//-----------------------------------------------------------------------------//
inline void io_accumulator_add(io_float32_t delta_t,
                               io_fixed_step_accumulator_t* accumulator)
{
  accumulator->accumulator += delta_t;
}

// Returns true if another fixed step should be executed.
//-----------------------------------------------------------------------------//
inline io_bool_t io_accumulator_step(io_fixed_step_accumulator_t* accumulator)
{
  io_bool_t stepped = IO_FALSE;
  if (accumulator->accumulator >= accumulator->delta_t)
  {
    accumulator->accumulator -= accumulator->delta_t;
    stepped = IO_TRUE;
  }

  accumulator->interpolator = accumulator->accumulator / accumulator->delta_t;
  return stepped;
}

// Returns the minimum of x and y.
//-----------------------------------------------------------------------------//
inline io_uint32_t io_min(io_uint32_t x, io_uint32_t y)
{
  if (x < y)
    return x;
  return y;
}

// Returns the maximum of x and y.
//-----------------------------------------------------------------------------//
inline io_uint32_t io_max(io_uint32_t x, io_uint32_t y)
```

(continues on next page)

(continued from previous page)

```
{
  if (x > y)
    return x;
  return y;
}

// Simple and fast hash function
//-----------------------------------------------------------------------------//
inline io_uint32_t io_hash(const char* data)
{
  if (0 == data || '\0' == *data)
    return 0u;

  io_uint32_t hash = 5381u;
  while (*data)
    hash = ((hash << 5) + hash) + (io_uint8_t)(*data++);

  return hash;
}

// Converts the given string to a name.
//   Please note that the given string is not being internalized when using
//   this function, hence it won't be possible to request the string for this
//   name. Please use the name related functions provided by the "io_base_t"
//   interface to ensure string internalization.
//-----------------------------------------------------------------------------//
inline io_name_t io_to_name(const char* string)
{
  io_name_t name;
  name.hash = io_hash(string);
  return name;
}

//-----------------------------------------------------------------------------//
// Interface types and typedefs
//-----------------------------------------------------------------------------//

// Called when a change to a file was detected.
//-----------------------------------------------------------------------------//
typedef void (*io_filesystem_on_file_changed_function)(const char* filename);

// Settings for the pathfinding system
//-----------------------------------------------------------------------------//
typedef struct
{
  io_uint32_t find_walkable_cell_range; // The range in voxels we're going to
                                        // search for a walkable cell (at the
                                        // start position of the path).
  io_float32_t capsule_radius;          // The capsule radius of the agent.
  io_float32_t capsule_half_height;     // The capsule half height of the agent.
  io_float32_t step_height; // The maximum step the agent is going to take.
  io_float32_t cell_size;   // The size of the cells/voxels we're using for
```

(continues on next page)

(continued from previous page)

```
                            // calculating the path.
  io_uint32_t
      num_max_steps; // The maximum number of iterations to compute per frame.
} io_pathfinding_path_settings_t;

// Physics overlap result data
//-----------------------------------------------------------------------------//
typedef struct
{
  io_bool_t hit; // True if an overlap was detected.

  io_ref_t entity; // The first entity that is overlapping.
} io_physics_overlap_result_t;

// Physics raycast result data
//-----------------------------------------------------------------------------//
typedef struct
{
  io_bool_t hit; // True if a hit was detected.

  io_float32_t distance; // The distance to the hit.
  io_vec3_t position;    // The position of the hit.
  io_vec3_t normal;      // The normal at the hit position.

  io_ref_t entity; // The entity that was hit.
} io_physics_raycast_result_t;

// Voxel shape raycast result data
//-----------------------------------------------------------------------------//
typedef struct
{
  io_float32_t distance; // The distance to the hit.

  io_vec3_t normal;       // The normal of the hit (in world space).
  io_vec3_t normal_local; // The normal of the hit (in local/voxel space).

  io_ref_t shape;    // The shape hit.
  io_u8vec3_t coord; // The local coordinate of the voxel in the shape
                     // we've hit.
} io_component_voxel_shape_raycast_result_t;

// Header for a single event
//-----------------------------------------------------------------------------//
typedef struct
{
  io_name_t type; // The type of the event.
  io_uint32_t
      data_size_in_bytes; // The size of the data blob attached to this event.
} io_events_header_t;

// SOA style batch of script data
//-----------------------------------------------------------------------------//
```

(continues on next page)

(continued from previous page)

```
typedef struct
{
  void** user_datas; // User data ptrs that can be used to, e.g., store the Lua
                     // state.
  const io_ref_t* entities; // The entities of each of the script components.
  const io_uint32_t* update_intervals; // The update intervals of each of the
                                       // script components.
} io_user_script_batch_t;

// Defines an anchor for UI transformations
//-----------------------------------------------------------------------------//
typedef struct
{
  float anchor, offset;
} io_ui_anchor_t;

// Defines a set of anchor offsets for UI transformations
//-----------------------------------------------------------------------------//
typedef struct
{
  float left, right, top, bottom;
} io_ui_anchor_offsets_t;

// Defines a rectangle.
//-----------------------------------------------------------------------------//
typedef struct
{
  io_vec2_t pos;
  io_vec2_t extent;
} io_ui_rect_t;

//-----------------------------------------------------------------------------//
// Event data types
//-----------------------------------------------------------------------------//

// Physics contact event data
//   Event type name:       "physics_contact"
//   Reported by callback:  "on_physics_events"
//-----------------------------------------------------------------------------//
typedef struct
{
  io_ref_t entity0, entity1; // The entities participating in the contact.
  io_vec3_t pos, impulse;    // Position and impulse of the contact.
} io_events_data_physics_contact_t;

// Voxel shape fracture event data
//   Event type name:       "shape_fractured"
//   Reported by callback:  "on_shape_events"
//-----------------------------------------------------------------------------//
typedef struct
{
  io_ref_t base, chunk; // The base and the chunk shape
```

(continues on next page)

(continued from previous page)

```
} io_events_data_shape_fractured_t;

// Voxel shape voxelization completed event data
//   Event type name:       "shape_voxelization_completed"
//   Reported by callback:  "on_shape_events"
//------------------------------------------------------------------------------//
typedef struct
{
  io_ref_t shape; // The shape that has been voxelized
} io_events_data_shape_voxelization_completed_t;

//------------------------------------------------------------------------------//
// Interface function declarations and implementations
//------------------------------------------------------------------------------//

//------------------------------------------------------------------------------//
inline void
io_pathfinding_init_path_settings(io_pathfinding_path_settings_t* settings)
{
  settings->find_walkable_cell_range = 8u;
  settings->capsule_radius = 0.2f;
  settings->capsule_half_height = 0.4f;
  settings->step_height = 0.2f;
  settings->cell_size = 0.2f;
  settings->num_max_steps = 128u;
}

//------------------------------------------------------------------------------//
inline const void* io_events_get_data(const io_events_header_t* current)
{
  if (current->data_size_in_bytes > 0u)
    return (io_uint8_t*)current + sizeof(io_events_header_t);
  return 0;
}

//------------------------------------------------------------------------------//
inline const io_events_header_t*
io_events_get_next(const io_events_header_t* current)
{
  const io_events_header_t* next =
      (io_events_header_t*)((io_uint8_t*)current + sizeof(io_events_header_t) +
                            current->data_size_in_bytes);
  return next;
}

//------------------------------------------------------------------------------//
#ifdef IO_API_IMPLEMENTATION
//------------------------------------------------------------------------------//

// Empty

//------------------------------------------------------------------------------//
```

(continues on next page)

(continued from previous page)

```
#endif // IO_API_IMPLEMENTATION
//-----------------------------------------------------------------------------//

//-----------------------------------------------------------------------------//
// Interfaces that need to be implemented by the plugin providers. Only
// implemented functions are called.
// **Make sure to NULL unused function ptrs before registering the interface!**
//-----------------------------------------------------------------------------//

//-----------------------------------------------------------------------------//
#define IO_USER_TASK_API_NAME "io_user_task_i"
//-----------------------------------------------------------------------------//

// Interface for extending the game mode
//-----------------------------------------------------------------------------//
struct io_user_task_i // NOLINT
{
  // Called once when the game mode becomes active.
  void (*on_activate)();
  // Called once when the game mode becomes inactive.
  void (*on_deactivate)();

  // Called every frame when the game mode is active.
  void (*on_tick)(io_float32_t delta_t);
};

//-----------------------------------------------------------------------------//
#define IO_USER_DEBUG_VIEW_API_NAME "io_user_debug_view_i"
//-----------------------------------------------------------------------------//

// Interface for providing custom debug views via Dear ImGui
//   Debug views can be cycled through via the F1 key by default in the editor
//   and the game mode
//-----------------------------------------------------------------------------//
struct io_user_debug_view_i // NOLINT
{
  // Called when the debug view should be filled using Dear ImGui calls. Called
  // in the context of the current debug view window.
  void (*on_build_debug_view)(io_float32_t delta_t);
};

//-----------------------------------------------------------------------------//
#define IO_USER_EDITOR_API_NAME "io_user_editor_i"
//-----------------------------------------------------------------------------//

// Interface for extending the editor
//-----------------------------------------------------------------------------//
struct io_user_editor_i // NOLINT
{
  // Called when building the "Plugin" menu in the editor's menu bar. Extend the
  // menu here using ImGui::MenuItem() etc.
  void (*on_build_plugin_menu)();
```

(continues on next page)

(continued from previous page)

```
  // Called once when the editor becomes active.
  void (*on_activate)();
  // Called once when the editor becomes inactive.
  void (*on_deactivate)();

  // Called every frame when the editor is active.
  void (*on_tick)(io_float32_t delta_t);
};

//----------------------------------------------------------------------------//
#define IO_USER_EDITOR_TOOL_API_NAME "io_user_editor_tool_i"
//----------------------------------------------------------------------------//

// Interface for implementing custom editor tools
//----------------------------------------------------------------------------//
struct io_user_editor_tool_i // NOLINT
{
  // Called when the tool is active and should be updated.
  void (*on_tick)(io_float32_t delta_t);

  // Return the icon for displaying the tool in the editor.
  const char* (*get_icon)();
  // Return the tooltip shown when hovering the tool.
  const char* (*get_tooltip)();
};

//----------------------------------------------------------------------------//
#define IO_USER_DENOISER_API_NAME "io_user_denoiser_i"
//----------------------------------------------------------------------------//

// Interface for providing denoisers applied when exporting path-traced renders
//----------------------------------------------------------------------------//
struct io_user_denoiser_i // NOLINT
{
  // Requests the name of the denoiser.
  const char* (*get_name)();

  // Request to denoise the input data and write the result to output.
  void (*denoise)(io_uint32_t width, io_uint32_t height, const io_vec4_t* input,
                  io_vec4_t* output);
};

//----------------------------------------------------------------------------//
#define IO_USER_EVENTS_API_NAME "io_user_events_i"
//----------------------------------------------------------------------------//

// Interface for subscribing to events from subsystems
//----------------------------------------------------------------------------//
struct io_user_events_i // NOLINT
{
  // Called when physics related events are ready to be processed.
```

(continues on next page)

(continued from previous page)

```
  void (*on_physics_events)(const io_events_header_t* begin,
                            const io_events_header_t* end);
  // Called when voxel shape related events are ready to be processed.
  void (*on_shape_events)(const io_events_header_t* begin,
                          const io_events_header_t* end);
};

//--------------------------------------------------------------------------//
#define IO_USER_SCRIPT_API_NAME "io_user_script_i"
//--------------------------------------------------------------------------//

// Interface for implementing custom scripting backends
//--------------------------------------------------------------------------//
struct io_user_script_i // NOLINT
{
  // Called when a script component is initialized.
  void (*on_init_script)(const char* script_name, io_ref_t entity,
                         io_uint32_t update_interval, void** user_data);
  // Called when a script component is destroyed.
  void (*on_destroy_script)(io_ref_t entity, void** user_data);
  // Called when the given scripts should be ticked.
  void (*on_tick_scripts)(io_float32_t delta_t,
                          const io_user_script_batch_t* scripts,
                          io_uint32_t scripts_length);
  // Called when the given scripts should be activated.
  void (*on_activate_scripts)(const io_user_script_batch_t* scripts,
                              io_uint32_t scripts_length);
  // Called when the given scripts should be deactivated.
  void (*on_deactivate_scripts)(const io_user_script_batch_t* scripts,
                                io_uint32_t scripts_length);
};

//--------------------------------------------------------------------------//
// Interfaces that are provided by IOLITE.
//--------------------------------------------------------------------------//

//--------------------------------------------------------------------------//
#define IO_API_MANAGER_API_NAME "io_api_manager_i"
//--------------------------------------------------------------------------//

// The central interface for registering and retrieving API interfaces
//--------------------------------------------------------------------------//
struct io_api_manager_i // NOLINT
{
  // Registers a new API interface for the given name. Multiple interfaces for
  // the same name are allowed.
  void (*register_api)(const char* name, const void* interface);
  // Unregisters the given API interface.
  void (*unregister_api)(const void* interface);

  // Finds the first API interface for the given name.
  const void* (*find_first)(const char* name);
```

(continues on next page)

(continued from previous page)

```
  // Returns the next API interface for the given type. NULL if none is found.
  const void* (*get_next)(const void* interface);
};

//------------------------------------------------------------------------------//
#define IO_BASE_API_NAME "io_base_i"
//------------------------------------------------------------------------------//

// Collection of global functions which are not tied to a certain
// subsystem
//------------------------------------------------------------------------------//
struct io_base_i // NOLINT
{
  // Refs

  // Returns an invalid ref.
  io_ref_t (*ref_invalid)();
  // Returns true if this ref is valid.
  io_bool_t (*ref_is_valid)(io_ref_t ref);
  // Returns the type ID for the provided ref.
  io_uint32_t (*ref_get_type_id)(io_ref_t ref);
  // Returns the ID for the given ref.
  io_uint32_t (*ref_get_id)(io_ref_t ref);

  // Names

  // Internalizes the given string and returns a name.
  io_name_t (*name_from_string)(const char* string);
  // Returns the internalized string for the given name.
  const char* (*name_get_string)(io_name_t name);

  // Variants

  // Creates a new variant from a float value.
  io_variant_t (*variant_from_float)(io_float32_t value);
  // Gets the value of the variant as a float.
  io_float32_t (*variant_get_float)(io_variant_t variant);

  // Creates a new variant from a signed integer value.
  io_variant_t (*variant_from_int)(io_int32_t value);
  // Gets the value of the variant as a signed integer.
  io_int32_t (*variant_get_int)(io_variant_t variant);

  // Creates a new variant from an unsigned integer value.
  io_variant_t (*variant_from_uint)(io_uint32_t value);
  // Gets the value of the variant as an unsigned integer.
  io_uint32_t (*variant_get_uint)(io_variant_t variant);

  // Creates a new variant from a string.
  io_variant_t (*variant_from_string)(const char* value);
  // Gets the value of the variant as a string.
  const char* (*variant_get_string)(io_variant_t variant);
```

(continues on next page)

(continued from previous page)

```
    // Creates a new variant from a vec2.
    io_variant_t (*variant_from_vec2)(io_vec2_t value);
    // Gets the value of the variant as a vec2.
    io_vec2_t (*variant_get_vec2)(io_variant_t variant);

    // Creates a new variant from a vec3.
    io_variant_t (*variant_from_vec3)(io_vec3_t value);
    // Gets the value of the variant as a vec3.
    io_vec3_t (*variant_get_vec3)(io_variant_t variant);

    // Creates a new variant from a vec4.
    io_variant_t (*variant_from_vec4)(io_vec4_t value);
    // Gets the value of the variant as a vec4.
    io_vec4_t (*variant_get_vec4)(io_variant_t variant);

    // Creates a new variant from a quaternion.
    io_variant_t (*variant_from_quat)(io_quat_t value);
    // Gets the value of the variant as a quaternion.
    io_quat_t (*variant_get_quat)(io_variant_t variant);

    // Creates a new variant from an ivec2.
    io_variant_t (*variant_from_ivec2)(io_ivec2_t value);
    // Gets the value of the variant as an ivec2.
    io_ivec2_t (*variant_get_ivec2)(io_variant_t variant);

    // Creates a new variant from an ivec3.
    io_variant_t (*variant_from_ivec3)(io_ivec3_t value);
    // Gets the value of the variant as an ivec3.
    io_ivec3_t (*variant_get_ivec3)(io_variant_t variant);

    // Creates a new variant from an ivec4.
    io_variant_t (*variant_from_ivec4)(io_ivec4_t value);
    // Gets the value of the variant as an ivec4.
    io_ivec4_t (*variant_get_ivec4)(io_variant_t variant);

    // Creates a new variant from an uvec2.
    io_variant_t (*variant_from_uvec2)(io_uvec2_t value);
    // Gets the value of the variant as an uvec2.
    io_uvec2_t (*variant_get_uvec2)(io_variant_t variant);

    // Creates a new variant from an uvec3.
    io_variant_t (*variant_from_uvec3)(io_uvec3_t value);
    // Gets the value of the variant as an uvec3.
    io_uvec3_t (*variant_get_uvec3)(io_variant_t variant);

    // Creates a new variant from an 8-bit uvec3.
    io_variant_t (*variant_from_u8vec3)(io_u8vec3_t value);
    // Gets the value of the variant as an 8-bit uvec3.
    io_u8vec3_t (*variant_get_u8vec3)(io_variant_t variant);

    // Creates a new variant from a 16-bit uvec3.
```

(continues on next page)

(continued from previous page)

```
  io_variant_t (*variant_from_u16vec3)(io_u16vec3_t value);
  // Gets the value of the variant as a 16-bit uvec3.
  io_u16vec3_t (*variant_get_u16vec3)(io_variant_t variant);

  // Creates a new variant from an uvec4.
  io_variant_t (*variant_from_uvec4)(io_uvec4_t value);
  // Gets the value of the variant as an uvec4.
  io_uvec4_t (*variant_get_uvec4)(io_variant_t variant);

  // Dear ImGui

  // Needed to use Dear ImGui in a plugin; use in combination with
  // ImGui::SetCurrentContext().
  void* (*imgui_get_context)();
  // Needed to use Dear ImGui in a plugin; use in combination with
  // ImGui::SetAllocatorFunctions().
  void (*imgui_get_allocator_functions)(void** alloc_func, void** free_func);

  // Memory management. Provides a TLSF-backed, thread-safe allocator which
  // features allocation tracking.

  // Allocates a memory area with the given size.
  void* (*mem_allocate)(io_uint32_t size_in_bytes);
  // Allocates a memory area with the given size and alignment.
  void* (*mem_allocate_aligned)(io_uint32_t size_in_bytes,
                                io_uint32_t alignment_in_bytes);
  // Frees the provided memory area.
  void (*mem_free)(void* ptr);

  // Task scheduler

  // Enqueues and kicks the given task
  void (*scheduler_enqueue_task)(io_scheduler_task_t* task);
  // Waits till the given task is completed
  void (*scheduler_wait_for_task)(const io_scheduler_task_t* task);
  // Returns true if the given task is completed
  io_bool_t (*scheduler_is_task_completed)(const io_scheduler_task_t* task);
};

//-----------------------------------------------------------------------------//
#define IO_LOGGING_API_NAME "io_logging_i"
//-----------------------------------------------------------------------------//

// Provides access to the logging subsystem
//-----------------------------------------------------------------------------//
struct io_logging_i // NOLINT
{
  // Logs the given message as information.
  void (*log_info)(const char* msg);
  // Logs the given message as a warning.
  void (*log_warning)(const char* msg);
  // Logs the given message as a error.
```

(continues on next page)

(continued from previous page)

```
  void (*log_error)(const char* msg);
};

//-----------------------------------------------------------------------------//
#define IO_EDITOR_API_NAME "io_editor_i"
//-----------------------------------------------------------------------------//

// Provides access to the editor
//-----------------------------------------------------------------------------//
struct io_editor_i // NOLINT
{
  // Selects the provided node.
  void (*select_node)(io_ref_t node);

  // Returns the first selected node.
  io_ref_t (*get_first_selected_node)();
  // Returns the first selected entity.
  io_ref_t (*get_first_selected_entity)();
};

//-----------------------------------------------------------------------------//
#define IO_CUSTOM_COMPONENTS_API_NAME "io_custom_components_i"
//-----------------------------------------------------------------------------//

// Interface for managing custom components
//-----------------------------------------------------------------------------//
struct io_custom_components_i // NOLINT
{
  // Registering and configuring custom components

  // Requests a new custom component manager to configure.
  io_handle16_t (*request_manager)();
  // Releases and destroys the given custom component manager.
  void (*release_and_destroy_manager)(io_handle16_t manager);

  // Registers a new property with the given name, default value, optional
  // accessor, and flags.
  //   - Registering properties is only allowed *before* calling "init_manager".
  //   - The type of the property is derived from the type of the
  //     "default_value" variant.
  //   - The "accessor" parameter is optional (set to NULL if not needed) and
  //     can point to a *pointer* of the property's type. The accessor needs to
  //     be available while the manager is registered and gets dynamically
  //     updated in case the property memory changes (when the manager runs over
  //     its capacity). Alternatively, the property memory can be queried using
  //     the "get_property_memory" function.
  //   - The accessors will be initialized *after* calling "init_manager".
  void (*register_property)(io_handle16_t manager, const char* name,
                            io_variant_t default_value, void** accessor,
                            io_property_flags flags);

  // Initializes the manager and makes it available under the given (type) name.
```

(continues on next page)

(continued from previous page)

```
  //   Call this *once* after all properties have been registered.
  void (*init_manager)(io_handle16_t manager, const char* type);

  // Returns the type ID for this type of component.
  io_uint32_t (*get_type_id)(io_handle16_t manager);

  // Functions to interact with components and their properties

  // Creates a new custom component and attaches it to the provided parent
  // entity.
  io_ref_t (*create_custom_component)(io_handle16_t manager,
                                      io_ref_t parent_entity);
  // Destroys the given custom component.
  void (*destroy_custom_component)(io_handle16_t manager, io_ref_t component);

  // Gets the linear memory for the property with the given name.
  //  Don't cache the returned pointer! The property memory will change when
  //  the manager runs over its current capacity.
  void* (*get_property_memory)(io_handle16_t manager, const char* name);
  // Gets the linear memory storing the entities of the active components.
  io_ref_t* (*get_entity_memory)(io_handle16_t manager);

  // Returns the number of active components.
  io_uint32_t (*get_num_active_components)(io_handle16_t manager);

  // Gets the entity the given component is attached to.
  io_ref_t (*get_entity)(io_handle16_t manager, io_ref_t component);
  // Gets the component for the given entity (if any).
  io_ref_t (*get_component_for_entity)(io_handle16_t manager, io_ref_t entity);
  // Returns true if the provided component is alive.
  io_bool_t (*is_alive)(io_handle16_t manager, io_ref_t component);

  // Converts the given index to a ref.
  io_ref_t (*make_ref)(io_handle16_t manager, io_uint32_t component_index);
  // Converts the given ref to an index.
  io_uint32_t (*make_index)(io_handle16_t manager, io_ref_t component);
};

//-----------------------------------------------------------------------------//
#define IO_SETTINGS_API_NAME "io_settings_i"
//-----------------------------------------------------------------------------//

// Provides access to the settings subsystem
//-----------------------------------------------------------------------------//
struct io_settings_i // NOLINT
{
  // Sets the given boolean setting.
  void (*set_bool)(const char* name, io_bool_t value);
  // Gets the given boolean setting.
  io_bool_t (*get_bool)(const char* name);

  // Sets the given unsigned integer setting.
```

(continues on next page)

(continued from previous page)

```
  void (*set_uint)(const char* name, io_uint32_t value);
  // Gets the given unsigned integer setting.
  io_uint32_t (*get_uint)(const char* name);

  // Sets the given float setting.
  void (*set_float)(const char* name, io_float32_t value);
  // Gets the given float setting.
  io_float32_t (*get_float)(const char* name);

  // Sets the given string setting.
  void (*set_string)(const char* name, const char* value);
  // Gets the given string setting.
  const char* (*get_string)(const char* name);
};

//--------------------------------------------------------------------------------//
#define IO_UI_API_NAME "io_ui_i"
//--------------------------------------------------------------------------------//

// Provides access to the UI subsystem
//--------------------------------------------------------------------------------//
struct io_ui_i // NOLINT
{
  // Draws a rectangle.
  void (*draw_rect)(io_vec4_t color);
  // Draws a circle.
  void (*draw_circle)(io_vec4_t color);
  // Draws a n-sided polygon.
  void (*draw_ngon)(io_vec4_t color, io_uint32_t num_sides);

  // Draws the given image.
  void (*draw_image)(const char* name, io_vec4_t tint);
  // Gets the size of the given image (in px).
  io_vec2_t (*get_image_size)(const char* name);

  // Draws the given text.
  void (*draw_text)(const char* text,
                    io_ui_text_align_horizontal align_horizontal,
                    io_ui_text_align_vertical align_vertical,
                    io_ui_text_flag flags);
  // Calculates the bounding rectangle for the given text and settings.
  io_ui_rect_t (*calc_text_bounds)(const char* text,
                                   io_ui_text_align_horizontal align_horizontal,
                                   io_ui_text_align_vertical align_vertical,
                                   io_ui_text_flag flags);
  // Returns the bounding rectangle for the last text that has been drawn.
  io_ui_rect_t (*get_last_text_bounds)();

  // Pushes the transform defined by the anchors and rotation (in rad).
  void (*push_transform)(io_ui_anchor_t left, io_ui_anchor_t right,
                         io_ui_anchor_t top, io_ui_anchor_t bottom,
                         io_float32_t rotation);
```

(continues on next page)

(continued from previous page)

```
  // Pushes the transform defined by the anchor preset, offsets, and rotation
  // (in rad).
  void (*push_transform_preset)(io_ui_anchor_preset preset,
                                io_ui_anchor_offsets_t offsets,
                                io_float32_t rotation);
  // Pops the last transform from the stack and sets it.
  void (*pop_transform)();

  // Calculates the scale and offset for the given base size and according
  // to the aspect mode.
  void (*push_scale_offset_for_base_size)(io_vec2_t base_size,
                                          io_ui_aspect_mode aspect_mode);
  // Pushes the current scale and offset to the stack and activates the given
  // parameters.
  void (*push_scale_offset)(io_float32_t scale, io_vec2_t offset);
  // Pops the last scale and offset from the stack and sets it.
  void (*pop_scale_offset)();

  // Pushes the current style variation float value to the stack and sets the
  // given one.
  void (*push_style_var_float)(io_ui_style_var var, io_float32_t value);
  // Pushes the current style variation vec4 value to the stack and sets the
  // given one.
  void (*push_style_var_vec4)(io_ui_style_var var, io_vec4_t value);
  void (*pop_style_var)();

  // Clips the children of the current transform.
  void (*clip_children)();

  // Pushes the current font size to the stack and sets the given one.
  void (*push_font_size)(io_float32_t size);
  // Pops the last font size from the stack and sets it.
  void (*pop_font_size)();

  // Returns true if the given position (in px) intersects the current
  // transform.
  io_bool_t (*intersects)(io_vec2_t position);
};

//------------------------------------------------------------------------------//
#define IO_WORLD_API_NAME "io_world_i"
//------------------------------------------------------------------------------//

// Provides access to the world subsystem
//------------------------------------------------------------------------------//
struct io_world_i // NOLINT
{
  // Gets the root node of the world.
  io_ref_t (*get_root_node)();
  // Gets the name of the currently loaded world
  const char* (*get_world_name)();
```

(continues on next page)

(continued from previous page)

```
  // Loads the world with the given name.
  void (*load_world)(const char* name);
  // Saves the current world under the given name.
  void (*save_world)(const char* name);

  // Spawns the prefab with the given name.
  io_ref_t (*spawn_prefab)(const char* name);

  // Retrieves the currently active camera.
  io_ref_t (*get_active_camera)();

  // Gets the current time factor.
  io_float32_t (*get_current_time_factor)();
  // Pushes the current time factor to the stack and applies the given one.
  //   Use this to speed up or slow down the game time.
  void (*push_time_factor)(io_float32_t factor);
  // Pops the last time factor from the stack and activates it.
  void (*pop_time_factor)();

  // Applies radius damage with the given parameters at the given position.
  void (*radius_damage)(io_vec3_t pos, io_float32_t radius,
                        io_world_radius_damage_flags flags, float max_hardness);

  // Calculates a ray from the current camera position to the position of the
  // mouse.
  void (*calc_mouse_ray)(io_vec3_t* origin, io_vec3_t* dir);

  // Highlights the given node.
  void (*highlight_node)(io_ref_t node, io_vec4_t color, io_bool_t outline);
};

//-----------------------------------------------------------------------------//
#define IO_SAVE_DATA_API_NAME "io_save_data_i"
//-----------------------------------------------------------------------------//

// Provides access to the save data subsystem
//-----------------------------------------------------------------------------//
struct io_save_data_i // NOLINT
{
  // Saves the provided node hierarchy to the user data directory.
  void (*save_to_user_data)(const char* filename, io_ref_t node);
  // Loads the node hierarchy from the provided file in the user data directory.
  io_ref_t (*load_from_user_data)(const char* filename);
};

//-----------------------------------------------------------------------------//
#define IO_PARTICLE_SYSTEM_API_NAME "io_particle_system_i"
//-----------------------------------------------------------------------------//

// Provides access to the particle subsystem
//-----------------------------------------------------------------------------//
struct io_particle_system_i // NOLINT
```

(continues on next page)

(continued from previous page)

```
{
  // Spawns a particle emitter for the given effect.
  io_handle16_t (*spawn_particle_emitter)(const char* effect_name,
                                          io_vec3_t position,
                                          io_float32_t lifetime_in_seconds,
                                          io_bool_t adjust_spawn_rate);
  // Despawns the given emitter.
  void (*despawn_particle_emitter)(io_handle16_t emitter);

  // Attaches the given emitter to the given node. Pass an invalid ref to detach
  // it.
  void (*attach_to_node)(io_handle16_t emitter, io_ref_t node);

  // Sets the spawn rate for the given emitter.
  void (*set_spawn_rate)(io_handle16_t emitter, io_float32_t spawn_rate);
  // Sets the position for the given emitter.
  void (*set_position)(io_handle16_t emitter, io_vec3_t position);
  // Sets the scale for the given emitter.
  void (*set_scale)(io_handle16_t emitter, float scale);
  // Sets the scale for the given emitter.
  void (*set_emission_direction)(io_handle16_t emitter, io_vec3_t direction);
};

//--------------------------------------------------------------------------//
#define IO_INPUT_SYSTEM_API_NAME "io_input_system_i"
//--------------------------------------------------------------------------//

// Provides access to the input subsystem
//--------------------------------------------------------------------------//
struct io_input_system_i // NOLINT
{
  // Gets the state of the given key.
  io_input_key_state (*get_key_state)(io_input_key key, io_uint8_t player_id);
  // Gets the state of the given axis.
  io_float32_t (*get_axis_state)(io_input_axis axis, io_uint8_t player_id);

  // Gets the mouse position in pixels.
  io_vec2_t (*get_mouse_pos)();
  // Gets the position of the mouse in the viewport.
  io_vec2_t (*get_mouse_pos_viewport)();
  // Gets the mouse position relative to the last one (mouse movement).
  io_vec2_t (*get_mouse_pos_relative)();

  // Call this every frame to show the mouse cursor.
  void (*request_mouse_cursor)();
};

//--------------------------------------------------------------------------//
#define IO_PHYSICS_API_NAME "io_physics_i"
//--------------------------------------------------------------------------//

// Provides access to the physics subsystem
```

(continues on next page)

(continued from previous page)

```c
//-----------------------------------------------------------------------------//
struct io_physics_i // NOLINT
{
  // Sets the global gravity.
  void (*set_gravity)(io_vec3_t gravity);
  // Gets the global gravity
  io_vec3_t (*get_gravity)();

  // Performs a sphere overlap test.
  io_physics_overlap_result_t (*overlap_sphere)(io_vec3_t position,
                                                io_float32_t radius);
  // Performs a sphere sweep test in the given direction.
  io_physics_raycast_result_t (*sweep_sphere)(io_vec3_t position,
                                              io_float32_t radius,
                                              io_vec3_t direction,
                                              io_float32_t distance);
  // Performs a raycast.
  io_physics_raycast_result_t (*raycast)(io_vec3_t origin, io_vec3_t direction,
                                         io_float32_t distance);
};

//-----------------------------------------------------------------------------//
#define IO_PHYSX_API_NAME "io_physx_i"
//-----------------------------------------------------------------------------//

// Provides direct access to the internal low-level PhysX data structures
//    Use this if you want to directly utilize PhysX in your plugin to add
//    custom behavior and functionality
//-----------------------------------------------------------------------------//
struct io_physx_i // NOLINT
{
  // Returns the ptr to the global physx::PxPhysics instance.
  void* (*get_px_physics)();
  // Returns the ptr to the global physx::PxScene instance.
  void* (*get_px_scene)();

  // Returns the ptr to the physx::PxRigidActor instance for the given shape.
  // Please note the following:
  //     1. The actor can be *NULL* for shapes with pending voxelization or
  //     disabled collision.
  //     2. The actor is replaced after the voxelization for a shape finishes
  //     and the previous one becomes *invalid*.
  void* (*get_px_rigid_actor_for_shape)(io_ref_t shape);
};

//-----------------------------------------------------------------------------//
#define IO_DEBUG_GEOMETRY_API_NAME "io_debug_geometry_i"
//-----------------------------------------------------------------------------//

// Provides access to the debug geometry subsystem
//-----------------------------------------------------------------------------//
struct io_debug_geometry_i // NOLINT
```

(continues on next page)

(continued from previous page)

```
{
  // Simple draw functions

  // Draws a line.
  void (*draw_line)(io_vec3_t start, io_vec3_t end, io_vec4_t color,
                    io_bool_t always_in_front);

  // Draws a sphere.
  void (*draw_sphere)(io_vec3_t center, io_float32_t radius, io_vec4_t color,
                      io_bool_t always_in_front);

  // Draws a box.
  void (*draw_box)(io_vec3_t center, io_quat_t orientation, io_vec3_t extent,
                   io_vec4_t color, io_bool_t always_in_front,
                   io_box_face_flags face_flags);
  // Draws a solid box.
  void (*draw_solid_box)(io_vec3_t center, io_quat_t orientation,
                         io_vec3_t extent, io_vec4_t color,
                         io_bool_t always_in_front,
                         io_box_face_flags face_flags);

  // Batched draw functions

  // Draws the given lines.
  //   Line 1: positions[0], positions[1]
  //   Line 2: positions[2], positions[3]
  //   ...
  void (*draw_lines)(io_vec3_t* positions, io_uint32_t num_positions,
                     io_vec4_t color, io_bool_t always_in_front);
  // Draws the given line strip.
  //   Line 1: positions[0], positions[1]
  //   Line 2: positions[1], positions[2]
  //  ...
  void (*draw_line_strip)(io_vec3_t* positions, io_uint32_t num_positions,
                          io_vec4_t color, io_bool_t always_in_front);

  // Draws the given triangles.
  //   Triangle 1: positions[0], positions[1], positions[2]
  //   Triangle 2: positions[3], positions[4], positions[5]
  //   ...
  void (*draw_solid_triangles)(io_vec3_t* positions, io_uint32_t num_positions,
                               io_vec4_t color, io_bool_t always_in_front);

  // Enables software back face culling between the begin/end calls.
  void (*backface_culling_begin)();
  void (*backface_culling_end)();
};

//-----------------------------------------------------------------------------//
#define IO_SOUND_API_NAME "io_sound_i"
//-----------------------------------------------------------------------------//
```

(continues on next page)

(continued from previous page)

```c
// Provides access to the sound subsystem
//--------------------------------------------------------------------------//
struct io_sound_i // NOLINT
{
  // Plays the sound effect with the given name.
  io_handle64_t (*play_sound_effect)(const char* effect_name);
  // Stops the given sound effect.
  void (*stop_sound_effect)(io_handle64_t effect_handle);

  // Sets the position of the given sound effect.
  void (*set_position)(io_handle64_t effect_handle, io_vec3_t position);

  // Gets the current audio spectrum.
  void (*get_spectrum)(const io_float32_t** spectrum,
                       io_uint32_t* spectrum_length);
};

//--------------------------------------------------------------------------//
#define IO_PATHFINDING_API_NAME "io_pathfinding_i"
//--------------------------------------------------------------------------//

// Provides access to the pathfinding subsystem
//--------------------------------------------------------------------------//
struct io_pathfinding_i // NOLINT
{
  // Starts finding a path from "start" to "end".
  io_handle16_t (*find_path)(io_vec3_t start, io_vec3_t end,
                             const io_pathfinding_path_settings_t* settings);

  // Returns true if the given path handle is valid.
  io_bool_t (*is_valid)(io_handle16_t path_handle);

  // Resets the path.
  void (*reset_path)(io_handle16_t path_handle);
  // Destroys the path.
  void (*destroy_path)(io_handle16_t path_handle);

  // Returns true if a valid path has been found.
  io_bool_t (*is_path_found)(io_handle16_t path_handle);
  // Returns the next position on the path.
  io_bool_t (*get_next_position_on_path)(io_handle16_t path_handle,
                                         io_vec3_t* next_position);

  // Draws the given path.
  void (*draw_path)(io_handle16_t path_handle, io_vec4_t color,
                    io_bool_t always_in_front);
  // Draws the debug geometry for the given path.
  void (*draw_debug_geometry)(io_handle16_t path_handle,
                              io_bool_t always_in_front);
};

//--------------------------------------------------------------------------//
```

(continues on next page)

(continued from previous page)

```c
#define IO_FILESYSTEM_API_NAME "io_filesystem_i"
//-------------------------------------------------------------------------//

// Provides various file system related functions
//-------------------------------------------------------------------------//
struct io_filesystem_i // NOLINT
{
  // Accessing files in data sources

  // Tries to load the given file from the first data source (directory or
  // package) it is available in.
  io_bool_t (*load_file_from_data_source)(const char* filepath,
                                          io_uint8_t* buffer,
                                          io_uint32_t* buffer_length);

  // User data access

  // Creates or retrieves either the user directory or a subdirectory in the
  // user directory. "Subdirectory" is optional and can be NULL.
  void (*create_or_retrieve_user_directory)(const char* subdirectory,
                                            char* buffer,
                                            io_uint32_t* buffer_length);

  // File system watches

  // Watches the given directory and calls "callback" when a file changes.
  void (*watch_directory)(const char* directory_path,
                          io_filesystem_on_file_changed_function callback);
  // Watches the given directory in *all* data sources and calls "callback" when
  // a file changes.
  void (*watch_data_source_directory)(
      const char* directory_path,
      io_filesystem_on_file_changed_function callback);
  // Removes all watches using the given callback function.
  void (*remove_directory_watch)(io_filesystem_on_file_changed_function);
};

//-------------------------------------------------------------------------//
#define IO_ENTITY_API_NAME "io_entity_i"
//-------------------------------------------------------------------------//

// Provides access to entities
//-------------------------------------------------------------------------//
struct io_entity_i // NOLINT
{
  // Gets the type ID for entities.
  io_uint32_t (*get_type_id)();
  // Returns true if the given entity is alive.
  io_bool_t (*is_alive)(io_ref_t entity);

  // Gets the name of the given entity.
  const char* (*get_name)(io_ref_t entity);
```

(continues on next page)

(continued from previous page)

```
  // Returns the linear memory containing all the names for all active entities.
  io_name_t* (*get_name_memory)();

  // Finds the first entity with the given name.
  io_ref_t (*find_first_entity_with_name)(const char* name);
  // Finds all entities with the given name.
  void (*find_entities_with_name)(const char* name, io_ref_t* entities,
                                  io_uint32_t* entities_length);
};

// Base interface all components provide
//   *Not all functions are provided by all components*
//------------------------------------------------------------------------------//
typedef struct
{
  // Returns the type ID for this type of component.
  io_uint32_t (*get_type_id)();

  // Creates a new component and attaches it to the provided parent entity.
  io_ref_t (*create)(io_ref_t parent_entity);
  // Destroys the given component.
  void (*destroy)(io_ref_t component);

  // Commits any changes and reloads the internals of the component.
  void (*commit_changes)(io_ref_t component);

  // Returns the number of active components.
  io_uint32_t (*get_num_active_components)();
  // Gets the component for the given entity (if any).
  io_ref_t (*get_component_for_entity)(io_ref_t entity);

  // Returns a ref for the given linear component index.
  io_ref_t (*make_ref)(io_uint32_t component_index);
  // Returns the linear component index for the given ref
  io_uint32_t (*make_index)(io_ref_t component);

  // Returns true if the given component is alive.
  io_bool_t (*is_alive)(io_ref_t component);
  // Gets the entity the given component is attached to.
  io_ref_t (*get_entity)(io_ref_t component);

  // Sets the property of the given component to the provided value.
  void (*set_property)(io_ref_t component, const char* name,
                       io_variant_t value);
  // Gets the property of the given component as a variant.
  io_variant_t (*get_property)(io_ref_t component, const char* name);

  // Returns the linear property memory for the given property for all active
  // components.
  void* (*get_property_memory)(const char* name);
  // Returns the linear memory containing the entities for all active
  // components.
```

(continues on next page)

(continued from previous page)

```
  io_ref_t* (*get_entity_memory)();

  // Returns a list of property descriptions for all the properties the
  // component provides.
  void (*list_properties)(io_property_desc_t* property_descs,
                          io_uint32_t* property_descs_length);
} io_component_base_i;

//----------------------------------------------------------------------------//
#define IO_COMPONENT_NODE_API_NAME "io_component_node_i"
//----------------------------------------------------------------------------//

// Provides access to node components
//----------------------------------------------------------------------------//
struct io_component_node_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;

  // Creates a new node and attaches it to the current world's root node.
  io_ref_t (*create)(const char* name);
  // Creates a new node and attaches it to the provided parent node.
  io_ref_t (*create_with_parent)(const char* name, io_ref_t parent);

  // Retrieves the parent node (if any).
  io_ref_t (*get_parent)(io_ref_t node);
  // Retrieves the next sibling node (if any).
  io_ref_t (*get_next_sibling)(io_ref_t node);
  // Retrieves the previous sibling node (if any).
  io_ref_t (*get_prev_sibling)(io_ref_t node);

  // Sets the hidden state of the node.
  void (*set_hidden)(io_ref_t node, io_bool_t hidden);
  // Returns true if the node is hidden, false otherwise.
  io_bool_t (*is_hidden)(io_ref_t node);

  // Sets the position for the given node.
  void (*set_position)(io_ref_t node, io_vec3_t pos);
  // Gets the position of the given node.
  io_vec3_t (*get_position)(io_ref_t node);
  // Gets the linear memory containing the positions of all active nodes
  io_vec3_t* (*get_position_memory)();

  // Sets the world position for the given node.
  void (*set_world_position)(io_ref_t node, io_vec3_t pos);
  // Gets the world position of the given node.
  io_vec3_t (*get_world_position)(io_ref_t node);

  // Sets the size for the given node.
  void (*set_size)(io_ref_t node, io_vec3_t size);
  // Gets the size of the given node.
  io_vec3_t (*get_size)(io_ref_t node);
```

(continues on next page)

(continued from previous page)

```
  // Gets the linear memory containing the sizes of all active nodes
  io_vec3_t* (*get_size_memory)();

  // Sets the world size for the given node.
  void (*set_world_size)(io_ref_t node, io_vec3_t size);
  // Gets the world size of the given node.
  io_vec3_t (*get_world_size)(io_ref_t node);

  // Sets the orientation for the given node.
  void (*set_orientation)(io_ref_t node, io_quat_t orient);
  // Gets the orientation of the given node.
  io_quat_t (*get_orientation)(io_ref_t node);
  // Gets the linear memory containing the orientations of all active nodes
  io_quat_t* (*get_orientation_memory)();

  // Sets the world orientation for the given node.
  void (*set_world_orientation)(io_ref_t node, io_quat_t orient);
  // Gets the world orientation of the given node.
  io_quat_t (*get_world_orientation)(io_ref_t node);

  // Transform the given world space position to the local space of the node.
  io_vec3_t (*to_local_space)(io_ref_t node, io_vec3_t pos);
  // Transform the given world space direction to the local space of the node.
  io_vec3_t (*to_local_space_direction)(io_ref_t node, io_vec3_t dir);
  // Transform the given local space position to world space.
  io_vec3_t (*to_world_space)(io_ref_t node, io_vec3_t pos);
  // Transform the given local space direction to world space.
  io_vec3_t (*to_world_space_direction)(io_ref_t node, io_vec3_t dir);

  // Collects all nodes in the hierarchy (depth first ordering).
  void (*collect_nodes_depth_first)(io_ref_t root_node, io_ref_t* nodes,
                                    io_uint32_t* nodes_length);
  // Collects all nodes in the hierarchy (breadth first ordering).
  void (*collect_nodes_breadth_first)(io_ref_t root_node, io_ref_t* nodes,
                                      io_uint32_t* nodes_length);

  // Updates the transforms of the node hierarchy.
  void (*update_transforms)(io_ref_t node);
  // Updates the transforms of the node hierarchy in parallel (if possible).
  void (*update_transforms_jobified)(const io_ref_t* node,
                                     io_uint32_t nodes_length);
};

//-----------------------------------------------------------------------------//
#define IO_COMPONENT_CUSTOM_DATA_API_NAME "io_component_custom_data_i"
//-----------------------------------------------------------------------------//

// Provides access to custom data components
//-----------------------------------------------------------------------------//
struct io_component_custom_data_i // NOLINT
{
  // Base interface functions.
```

(continues on next page)

(continued from previous page)

```
  io_component_base_i base;

  // Gets the value for the given index.
  io_variant_t (*get)(io_ref_t custom_data, io_uint32_t index);
  // Sets the value for the given index.
  void (*set)(io_ref_t custom_data, io_uint32_t index, io_variant_t value);
  // Sets the value for the given index.
  void (*add)(io_ref_t custom_data, io_variant_t value);
  // Removes the value at the given index.
  void (*remove)(io_ref_t custom_data, io_uint32_t index);
};

//----------------------------------------------------------------------------//
#define IO_COMPONENT_TAG_API_NAME "io_component_tag_api_i"
//----------------------------------------------------------------------------//

// Provides access to tag components
//----------------------------------------------------------------------------//
struct io_component_tag_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;

  // Finds all entities with the given tag.
  void (*find_entities_with_tag)(const char* tag, io_ref_t* entities,
                                 io_uint32_t* entities_length);
};

//----------------------------------------------------------------------------//
#define IO_COMPONENT_FLIPBOOK_ANIMATION_API_NAME                               \
  "io_component_flipbook_animation_i"
//----------------------------------------------------------------------------//

// Provides access to flip book animation components
//----------------------------------------------------------------------------//
struct io_component_flipbook_animation_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;

  // Starts playing the provided flip book animation.
  void (*play)(io_ref_t flipbook_animation);
  // Stops playing the provided flip book animation.
  void (*stop)(io_ref_t flipbook_animation);
};

//----------------------------------------------------------------------------//
#define IO_COMPONENT_POST_EFFECT_VOLUME_API_NAME                               \
  "io_component_post_effect_volume_i"
//----------------------------------------------------------------------------//

// Provides access to post effect volume components
```

(continues on next page)

(continued from previous page)

```
//------------------------------------------------------------------------------//
struct io_component_post_effect_volume_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;
};

//------------------------------------------------------------------------------//
#define IO_COMPONENT_CAMERA_API_NAME "io_component_camera_i"
//------------------------------------------------------------------------------//

// Provides access to camera components
//------------------------------------------------------------------------------//
struct io_component_camera_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;
};

//------------------------------------------------------------------------------//
#define IO_COMPONENT_SCRIPT_API_NAME "io_component_script_i"
//------------------------------------------------------------------------------//

// Provides access to script components
//------------------------------------------------------------------------------//
struct io_component_script_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;
};

//------------------------------------------------------------------------------//
#define IO_COMPONENT_LIGHT_API_NAME "io_component_light_i"
//------------------------------------------------------------------------------//

// Provides access to light components
//------------------------------------------------------------------------------//
struct io_component_light_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;
};

//------------------------------------------------------------------------------//
#define IO_COMPONENT_VOXEL_SHAPE_API_NAME "io_component_voxel_shape_i"
//------------------------------------------------------------------------------//

// Provides access to voxel shape components
//------------------------------------------------------------------------------//
struct io_component_voxel_shape_i // NOLINT
{
  // Base interface functions.
```

(continues on next page)

(continued from previous page)

```
io_component_base_i base;

// Various

// Retrieves the current palette in use for this shape (if any).
io_ref_t (*get_palette)(io_ref_t shape);

// Conversion helpers

// Transforms the given world space position to local/voxel space.
io_vec3_t (*to_local_space)(io_ref_t shape, io_vec3_t position);
// Transforms the given world space position to a coordinate in local/voxel
// space.
io_ivec3_t (*to_local_coord)(io_ref_t shape, io_vec3_t position);
// Transforms the given local/voxel space position to world space.
io_vec3_t (*to_world_space)(io_ref_t shape, io_vec3_t position);

// Voxel data related functions

// Sets the given voxel to the palette index (and clamps coordinate to the
// dimensions of the shape).
void (*set)(io_ref_t shape, io_u8vec3_t coord, io_uint8_t palette_index);
// Sets the given voxel to the palette index (**without clamping** the
// coordinate to the dimensions of the shape).
void (*set_unsafe)(io_ref_t shape, io_u8vec3_t coord,
                   io_uint8_t palette_index);

// Marks the given voxel as fractured. Requires "support structures" to
// be enabled for the shape.
void (*fracture)(io_ref_t shape, io_u8vec3_t coord);
// Disconnects the given face of the voxel. Requires "support structures" to
// be enabled for the shape.
void (*disconnect)(io_ref_t shape, io_u8vec3_t coord, io_box_face_index face);
// Returns true if the face of the given voxel is connected to another voxel.
io_bool_t (*is_connected)(io_ref_t shape, io_u8vec3_t coord,
                          io_box_face_index face);

// Sets the volume defined by the min and max coordinate to the provided
// palette index.
void (*fill)(io_ref_t shape, io_u8vec3_t coord_min, io_u8vec3_t coord_max,
             io_uint8_t palette_index);

// Shrinks the given shape, so it utilizes the least amount of space.
void (*compact)(io_ref_t shape);

// Gets the palette index for the given voxel coordinate.
io_uint8_t (*get)(io_ref_t shape, io_u8vec3_t coord);

// Gets the dimensions of the voxel shape.
io_u16vec3_t (*get_dim)(io_ref_t shape);

// Gets the underlying voxel data.
```

(continues on next page)

(continued from previous page)

```
// Directly retrieving the data is the most efficient solution for modifying
// and reading back large chunks of voxels. The data is laid out in memory as
// follows:
//   idx = x + y * dim.x + z * dim.x * dim.y
io_uint8_t* (*get_voxel_data)(io_ref_t shape);

// Queues the given voxel shape for voxelization. Has to be called after every
// change to the underlying voxel data to commit the changes and make them
// visible.
void (*voxelize)(io_ref_t shape);
// Returns true if one (or multiple) voxelization requests for the given shape
// are pending.
io_bool_t (*is_voxelization_pending)(io_ref_t shape);

// Voxel shape queries

// Performs a raycast against the given shape. The result is optional and can
// be *NULL*.
io_bool_t (*raycast)(io_ref_t shape, io_vec3_t origin, io_vec3_t direction,
                     io_float32_t distance,
                     io_component_voxel_shape_raycast_result_t* result);
// Performs a raycast against the bounds of the given shape. The result is
// optional and can be *NULL*.
io_bool_t (*raycast_bounds)(
    io_ref_t shape, io_vec3_t origin, io_vec3_t direction,
    io_float32_t distance, io_bool_t flip_winding,
    io_component_voxel_shape_raycast_result_t* result);
// Performs a raycast again all shapes in the world. *All parameters* beside
// the origin, direction, and distance are *optional*.
io_bool_t (*raycast_global)(io_vec3_t origin, io_vec3_t direction,
                            io_float32_t distance,
                            io_component_voxel_shape_raycast_result_t* result,
                            const io_ref_t* shapes_to_ignore,
                            io_uint32_t shapes_to_ignore_length);
// Performs a sphere overlap test against all shapes in the world and returns
// all candidates.
void (*overlap_sphere_global)(io_vec3_t position, io_float32_t radius,
                              io_ref_t* overlapping_shapes,
                              io_uint32_t* overlapping_shapes_length);

// Physics related functions

// Applies the given force vector at the center of mass.
void (*apply_force)(io_ref_t shape, io_vec3_t force);
// Applies the given torque vector at the center of mass.
void (*apply_torque)(io_ref_t shape, io_vec3_t torque);

// Applies the given force vector at the given world position.
void (*apply_force_at_world_position)(io_ref_t shape, io_vec3_t force,
                                      io_vec3_t position);
// Applies the given force vector at the given world position.
void (*apply_force_at_local_position)(io_ref_t shape, io_vec3_t force,
```

(continues on next page)

(continued from previous page)

```
                                        io_vec3_t position);

  // Sets the linear velocity of this shape.
  void (*set_linear_velocity)(io_ref_t shape, io_vec3_t velocity);
  // Gets the linear velocity of this shape.
  io_vec3_t (*get_linear_velocity)(io_ref_t shape);

  // Sets the angular velocity of this shape.
  void (*set_angular_velocity)(io_ref_t shape, io_vec3_t velocity);
  // Gets the angular velocity of this shape.
  io_vec3_t (*get_angular_velocity)(io_ref_t shape);
};

//--------------------------------------------------------------------------//
#define IO_COMPONENT_CHARACTER_CONTROLLER_API_NAME                            \
  "io_component_character_controller_i"
//--------------------------------------------------------------------------//

// Provides access to character controller components
//--------------------------------------------------------------------------//
struct io_component_character_controller_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;

  // Moves the character controller.
  void (*move)(io_ref_t controller, io_vec3_t move_vector);
  // Returns true if the character controller is grounded.
  io_bool_t (*is_grounded)(io_ref_t controller);
  // Returns true if the character controller is colliding with its sides.
  io_bool_t (*is_colliding_sides)(io_ref_t controller);
  // Returns true if the upper part of the character controller is colliding.
  io_bool_t (*is_colliding_up)(io_ref_t controller);
};

//--------------------------------------------------------------------------//
#define IO_COMPONENT_CAMERA_CONTROLLER_API_NAME                               \
  "io_component_camera_controller_i"
//--------------------------------------------------------------------------//

// Provides access to camera controller components
//--------------------------------------------------------------------------//
struct io_component_camera_controller_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;

  // Sets the node to target for the given controller.
  void (*set_target_node)(io_ref_t controller, io_ref_t node);
  // Sets the Euler angles to target for the given controller.
  void (*set_target_euler_angles)(io_ref_t controller, io_vec3_t euler_angles);
  // Gets the Euler angles to target for the given controller.
```

(continues on next page)

(continued from previous page)

```
  io_vec3_t (*get_target_euler_angles)(io_ref_t controller);
};

//----------------------------------------------------------------------------//
#define IO_COMPONENT_PARTICLE_API_NAME "io_component_particle_i"
//----------------------------------------------------------------------------//

// Provides access to particle components
//----------------------------------------------------------------------------//
struct io_component_particle_i // NOLINT
{
  // Base interface functions.
  io_component_base_i base;

  // Returns the emitter handle for the given particle.
  io_handle16_t (*get_emitter_handle)(io_ref_t particle);
};

// Base interface all resources provide
//   *Not all functions are provided by all resources*
//----------------------------------------------------------------------------//
typedef struct
{
  // Returns the type ID for this type of resource.
  io_uint32_t (*get_type_id)();

  // Creates a new resource with the given name.
  io_ref_t (*create)(const char* name);
  // Destroys the given resource.
  void (*destroy)(io_ref_t resource);

  // Commits any changes and reloads the internals of the resource.
  void (*commit_changes)(io_ref_t component);

  // Returns the number of active resources.
  io_uint32_t (*get_num_active_resources)();

  // Returns a ref for the given linear resource index.
  io_ref_t (*make_ref)(io_uint32_t resource_index);
  // Returns the linear resource index for the given ref
  io_uint32_t (*make_index)(io_ref_t resource);

  // Returns the name of the resource.
  const char* (*get_name)(io_ref_t resource);
  // Returns the linear memory containing all the names for all active
  // resources.
  io_name_t* (*get_name_memory)();

  // Returns true if the given resource is alive.
  io_bool_t (*is_alive)(io_ref_t resource);

  // Sets the property of the given resource to the provided value.
```

(continues on next page)

(continued from previous page)

```
  void (*set_property)(io_ref_t resource, const char* name, io_variant_t value);
  // Gets the property of the given resource as a variant.
  io_variant_t (*get_property)(io_ref_t resource, const char* name);

  // Returns the linear property memory for the given property for all active
  // resources.
  void* (*get_property_memory)(const char* name);

  // Returns a list of property descriptions for all the properties the
  // resource provides.
  void (*list_properties)(io_property_desc_t* property_descs,
                          io_uint32_t* property_descs_length);
} io_resource_base_i;

//-----------------------------------------------------------------------------//
#define IO_RESOURCE_PALETTE_API_NAME "io_resource_palette_i"
//-----------------------------------------------------------------------------//

// Provides access to palette resources
//-----------------------------------------------------------------------------//
struct io_resource_palette_i // NOLINT
{
  // Base interface functions.
  io_resource_base_i base;

  // Sets the color for the given palette index of the given palette.
  void (*set_color)(io_ref_t palette, io_uint8_t palette_index,
                    io_vec4_t color);
  // Gets the color for the given palette index of the given palette.
  io_vec4_t (*get_color)(io_ref_t palette, io_uint8_t palette_index);

  // Material parameters:
  //   x: Roughness  [0, 1]
  //   y: Metal mask  [0, 1]
  //   z: Hardness   [0, 255]
  //   w: Emissive   [0, FLT_MAX]

  // Sets the material parameters for the given palette index of the given
  // palette.
  void (*set_material_parameters)(io_ref_t palette, io_uint8_t palette_index,
                                  io_vec4_t parameters);

  // Gets the material parameters for the given palette index of the given
  // palette.
  io_vec4_t (*get_material_parameters)(io_ref_t palette,
                                       io_uint8_t palette_index);
};

#endif
```

CHAPTER

ELEVEN

IOLITE 0.4 DOCUMENTATION

Welcome to the documentation of IOLITE, the voxel-powered game engine. If you're new here, we recommend directly heading to the *introduction section*.

INDEX

C

D

E

F

H

I

N

P

Q

R

S

T

U

V

W