Spectral

Generated by Doxygen 1.12.0

Chapter 1

Src Directories

1.1 Directory Structure

- game: game related code and assets, this will eventually be automatically generated code from the project files, but for now it is testing code.
- modules: all the different modules required for the engine to run as well as a few headers that are required.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Topic Documentation

6.1 Window Module

Classes

- struct sWindowFlags
- struct sWindow

  Window structure. More...
- struct WindowModule

  Window module class. More...

Typedefs

- typedef sWindow ∗(∗ **window::WindowLoader**) (const char ∗name, int width, int height, sWindowFlags flags)
- typedef void(∗ **window::WindowDestructor**) (sWindow ∗window)
- typedef void(∗ **window::WindowUpdate**) (sWindow ∗window)
- typedef void(∗ **window::WindowSwapBuffers**) (sWindow window)
- typedef bool(∗ **window::WindowShouldClose**) (sWindow window)
- typedef void(∗ **window::WindowSetShouldClose**) (sWindow window, bool value)
- typedef void ∗(∗ **window::WindowGetHandle**) (sWindow window)
- typedef bool(∗ **window::WindowIsKeyPressed**) (sWindow window, Key key)
- typedef bool(∗ **window::WindowIsMouseButtonPressed**) (sWindow window, int button)
- typedef void(∗ **window::WindowGetMousePosition**) (sWindow window, float ∗x, float ∗y)
- typedef void(∗ **window::WindowSetMousePosition**) (sWindow window, float x, float y)
- typedef void(∗ **window::WindowSetCursorMode**) (sWindow window, CursorMode mode)
- typedef void(∗ **window::WindowSetWindowTitle**) (sWindow window, const char ∗title)
- typedef void(∗ **window::WindowSetResizable**) (sWindow window, bool resizable)

Enumerations

- enum class **Key** {
  A =0 , **B** , **C** , **D** ,
  E , **F** , **G** , **H** ,
  I , **J** , **K** , **L** ,
  M , **N** , **O** , **P** ,
  Q , **R** , **S** , **T** ,
  U , **V** , **W** , **X** ,
  Y , **Z** , **Num0** , **Num1** ,
  Num2 , **Num3** , **Num4** , **Num5** ,
  Num6 , **Num7** , **Num8** , **Num9** ,
  Escape , **LControl** , **LShift** , **LAlt** ,
  LSystem , **RControl** , **RShift** , **RAlt** ,
  RSystem , **Menu** , **LBracket** , **RBracket** ,
  SemiColon , **Comma** , **Period** , **Quote** ,
  Slash , **BackSlash** , **Tilde** , **Equal** ,
  Dash , **Space** , **Return** , **BackSpace** ,
  Tab , **PageUp** , **PageDown** , **End** ,
  Home , **Insert** , **Delete** , **Add** ,
  Subtract , **Multiply** , **Divide** , **Left** ,
  Right , **Up** , **Down** , **Numpad0** ,
  Numpad1 , **Numpad2** , **Numpad3** , **Numpad4** ,
  Numpad5 , **Numpad6** , **Numpad7** , **Numpad8** ,
  Numpad9 , **F1** , **F2** , **F3** ,
  F4 , **F5** , **F6** , **F7** ,
  F8 , **F9** , **F10** , **F11** ,
  F12 , **F13** , **F14** , **F15** ,
  Pause , **KeyCount** }
- enum class CursorMode { CursorMode::Normal = 0 , CursorMode::Hidden , CursorMode::Disabled }

  Cursor mode enumeration.

Functions

- sWindow * **WindowModule::loadWindow** (const char *name, int width, int height, sWindowFlags flags)
- sWindow * **WindowModule::loadWindow** (const char *name, int width, int height, bool vsync=true, bool resizable=false)
- void **WindowModule::updateWindow** (sWindow *window)
- double **WindowModule::getTime** (sWindow window)
- **WindowModule::WindowModule** (const char *dynlib)

Variables

- bool sWindowFlags::vsync

  Whether vertical sync is enabled.
- bool sWindowFlags::resizable

  Whether the window is fullscreen.
- double sWindow::dt

  Delta time since the last frame.
- int sWindow::width

  The width of the window.
- int sWindow::height

  The height of the window.
- sWindowFlags sWindow::flags

The window flags.

- void * **sWindow::internal**
- WindowModule * **sWindow::creator**
- double **sWindow::lastTime**
- std::chrono::high_resolution_clock::time_point **sWindow::startTime**
- bool **sWindow::did_resize**
- window::WindowLoader **WindowModule::internal_loadWindow**
- window::WindowUpdate **WindowModule::internal_updateWindow**
- window::WindowDestructor WindowModule::destroyWindow

  Destructor function pointer for the window module.
- window::WindowSwapBuffers **WindowModule::swapBuffers**
- window::WindowShouldClose **WindowModule::shouldClose**
- window::WindowSetShouldClose **WindowModule::setShouldClose**
- window::WindowGetHandle **WindowModule::getHandle**
- window::WindowIsKeyPressed **WindowModule::isKeyPressed**
- window::WindowIsMouseButtonPressed **WindowModule::isMouseButtonPressed**
- window::WindowGetMousePosition **WindowModule::getMousePosition**
- window::WindowSetMousePosition **WindowModule::setMousePosition**
- window::WindowSetCursorMode **WindowModule::setCursorMode**
- window::WindowSetWindowTitle **WindowModule::setWindowTitle**

6.1.1 Detailed Description

6.1.2 Class Documentation

6.1.2.1 struct sWindowFlags

Class Members

bool	resizable	Whether the window is fullscreen. If true, the window will be displayed in fullscreen mode. **Note** This may not be supported on all platforms or windowing libraries.
bool	vsync	Whether vertical sync is enabled. If true, the window will synchronize its frame rate with the monitor's refresh rate. **Note** This may help reduce screen tearing and improve performance. This may not be supported on all platforms or windowing libraries.

6.1.2.2 struct sWindow

Window structure.

This structure represents a window created by the window module. It contains various properties and methods for interacting with the window.

Note

Certain members of this structure are meant to be not accessed directly, but cannot be made private due to the internal implementation.

Class Members

WindowModule ∗	creator	
bool	did_resize	
double	dt	Delta time since the last frame. This is the time in seconds since the last frame was rendered.
sWindowFlags	flags	The window flags. This is a structure containing various properties of the window. **See also** sWindowFlags
int	height	The height of the window. This is the height of the window in pixels.
void ∗	internal	
double	lastTime	
time_point	startTime	
int	width	The width of the window. This is the width of the window in pixels.

6.1.2.3 struct WindowModule

Window module class.

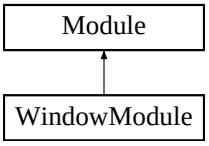
This class represents the window module and provides methods for creating and managing windows. It is responsible for loading the windowing library and providing access to its functions.

Note

This class contains function pointers that will be loaded from the module file (a dynamic library).

These functions will do something different depending on the implementation selected at runtime.

Inheritance diagram for WindowModule:



Public Member Functions

- sWindow ∗ **loadWindow** (const char ∗name, int width, int height, sWindowFlags flags)
- sWindow ∗ **loadWindow** (const char ∗name, int width, int height, bool vsync=true, bool resizable=false)
- void **updateWindow** (sWindow ∗window)
- double **getTime** (sWindow window)
- **WindowModule** (const char ∗dynlib)

Public Member Functions inherited from Module

- **Module** (const char ∗path, const char ∗ident)

Public Attributes

- window::WindowLoader **internal_loadWindow**
- window::WindowUpdate **internal_updateWindow**
- window::WindowDestructor destroyWindow

  Destructor function pointer for the window module.
- window::WindowSwapBuffers **swapBuffers**
- window::WindowShouldClose **shouldClose**
- window::WindowSetShouldClose **setShouldClose**
- window::WindowGetHandle **getHandle**
- window::WindowIsKeyPressed **isKeyPressed**
- window::WindowIsMouseButtonPressed **isMouseButtonPressed**
- window::WindowGetMousePosition **getMousePosition**
- window::WindowSetMousePosition **setMousePosition**
- window::WindowSetCursorMode **setCursorMode**
- window::WindowSetWindowTitle **setWindowTitle**

Public Attributes inherited from Module

- DynamicLibrary **lib**

6.1.3 Enumeration Type Documentation

6.1.3.1 CursorMode

```
enum class CursorMode  [strong]
```

Cursor mode enumeration.

This enumeration defines the different modes for the cursor in the window.

Note

The cursor mode may not be supported on all platforms or windowing libraries.

See also

WindowModule::setCursorMode

Enumerator

Normal	Normal cursor mode. The cursor is visible and can be moved freely within the window. **Note** This is the default mode.
Hidden	Hidden cursor mode. The cursor is hidden, might be movable depending on the windowing library. **Warning** This may not be supported on all platforms or windowing libraries.

Disabled	Locked cursor mode. The cursor is hidden and locked to the window, preventing it from leaving the window area. **Warning** This may not be supported on all platforms or windowing libraries. **Note** This is useful for first-person camera controls or similar applications.

6.1.4 Variable Documentation

6.1.4.1 creator

```
WindowModule* sWindow::creator
```

6.1.4.2 destroyWindow

```
window::WindowDestructor WindowModule::destroyWindow
```

Destructor function pointer for the window module.

This function is used to destroy a window created by the window module.

Note

This function will be loaded from the module file (a dynamic library), and the implementation may vary.

6.1.4.3 did_resize

```
bool sWindow::did_resize
```

6.1.4.4 dt

```
double sWindow::dt
```

Delta time since the last frame.

This is the time in seconds since the last frame was rendered.

6.1.4.5 flags

```
sWindowFlags sWindow::flags
```

The window flags.

This is a structure containing various properties of the window.

See also

sWindowFlags

6.1.4.6 height

```
int sWindow::height
```

The height of the window.

This is the height of the window in pixels.

6.1.4.7 internal

```
void* sWindow::internal
```

6.1.4.8 lastTime

```
double sWindow::lastTime
```

6.1.4.9 resizable

```
bool sWindowFlags::resizable
```

Whether the window is fullscreen.

If true, the window will be displayed in fullscreen mode.

Note

This may not be supported on all platforms or windowing libraries.

6.1.4.10 startTime

```
std::chrono::high_resolution_clock::time_point sWindow::startTime
```

6.1.4.11 vsync

```
bool sWindowFlags::vsync
```

Whether vertical sync is enabled.

If true, the window will synchronize its frame rate with the monitor's refresh rate.

Note

This may help reduce screen tearing and improve performance.
This may not be supported on all platforms or windowing libraries.

6.1.4.12 width

```
int sWindow::width
```

The width of the window.

This is the width of the window in pixels.

Chapter 7

Class Documentation

7.1 AssetBuffer Struct Reference

Public Attributes

- const uint8_t * **data**
- size_t **len**

7.1.1 Member Data Documentation

7.1.1.1 data

```
const uint8_t* AssetBuffer::data
```

7.1.1.2 len

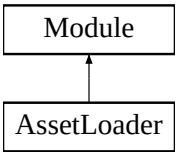
```
size_t AssetBuffer::len
```

The documentation for this struct was generated from the following file:

- src/modules/asset.h

7.2 AssetLoader Struct Reference

Inheritance diagram for AssetLoader:



Public Member Functions

- AssetBuffer **loadAsset** (const char ∗path)
- TextAssetBuffer **loadTextAsset** (const char ∗path)

Public Member Functions inherited from Module

- **Module** (const char ∗path, const char ∗ident)

Additional Inherited Members

Public Attributes inherited from Module

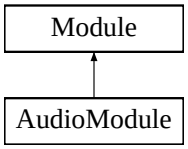
- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/asset.h

7.3 AudioModule Struct Reference

Inheritance diagram for AudioModule:



Public Member Functions

- void **seekAudioSourceSeconds** (sAudioSource source, float seconds)
- void **seekAudioSourcePercent** (sAudioSource source, float percent)
- float **getAudioSourcePercent** (sAudioSource source)
- float **getAudioSourceSeconds** (sAudioSource source)
- **AudioModule** (const char ∗dylib)

Public Member Functions inherited from Module

- **Module** (const char ∗path, const char ∗ident)

Public Attributes

- audio::Init **init**
- audio::LoadAudioClip **loadAudioClip**
- audio::CreateAudioSource **createAudioSource**
- audio::PlayAudioSource **playAudioSource**
- audio::StopAudioSource **stopAudioSource**
- audio::SetAudioSourcePosition **setAudioSourcePosition**
- audio::SetAudioSourceVelocity **setAudioSourceVelocity**
- audio::SetAudioSourcePitch **setAudioSourcePitch**
- audio::SetAudioSourceGain **setAudioSourceGain**
- audio::SetAudioSourceLooping **setAudioSourceLooping**
- audio::SeekAudioSourceSamples **seekAudioSourceSamples**
- audio::GetAudioSourceSamples **getAudioSourceSamples**
- audio::GetAudioSourceSampleRate **getAudioSourceSampleRate**
- audio::DestroyAudioClip **destroyAudioClip**
- audio::DestroyAudioSource **destroyAudioSource**
- audio::Destroy **destroy**

Public Attributes inherited from Module

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/aud/module.h

7.4 Clay__Alignpointer Struct Reference

Public Attributes

- char **c**
- void * **x**

7.4.1 Member Data Documentation

7.4.1.1 c

```
char Clay__Alignpointer::c
```

7.4.1.2 x

```
void* Clay__Alignpointer::x
```

The documentation for this struct was generated from the following file:

- src/modules/iui/clay.h

7.5 Clay_BorderElementConfig Struct Reference

Public Attributes

- Clay_Border **left**
- Clay_Border **right**
- Clay_Border **top**
- Clay_Border **bottom**
- Clay_Border **betweenChildren**
- Clay_CornerRadius **cornerRadius**

7.5.1 Member Data Documentation

7.5.1.1 betweenChildren

```
Clay_Border Clay_BorderElementConfig::betweenChildren
```

7.5.1.2 bottom

```
Clay_Border Clay_BorderElementConfig::bottom
```

7.5.1.3 cornerRadius

```
Clay_CornerRadius Clay_BorderElementConfig::cornerRadius
```

7.5.1.4 left

```
Clay_Border Clay_BorderElementConfig::left
```

7.5.1.5 right

```
Clay_Border Clay_BorderElementConfig::right
```

7.5.1.6 top

```
Clay_Border Clay_BorderElementConfig::top
```

The documentation for this struct was generated from the following file:

- src/modules/iui/clay.h

7.6 Clay_CustomElementConfig Struct Reference

Public Attributes

- void * **customData**

7.6.1 Member Data Documentation

7.6.1.1 customData

```
void* Clay_CustomElementConfig::customData
```

The documentation for this struct was generated from the following file:

- src/modules/iui/clay.h

7.7 Clay_ImageElementConfig Struct Reference

Public Attributes

- void * **imageData**
- Clay_Dimensions **sourceDimensions**

7.7.1 Member Data Documentation

7.7.1.1 imageData

```
void* Clay_ImageElementConfig::imageData
```

7.7.1.2 sourceDimensions

```
Clay_Dimensions Clay_ImageElementConfig::sourceDimensions
```

The documentation for this struct was generated from the following file:

- src/modules/iui/clay.h

7.8 Clay_RectangleElementConfig Struct Reference

Public Attributes

- Clay_Color **color**
- Clay_CornerRadius **cornerRadius**

7.8.1 Member Data Documentation

7.8.1.1 color

```
Clay_Color Clay_RectangleElementConfig::color
```

7.8.1.2 cornerRadius

```
Clay_CornerRadius Clay_RectangleElementConfig::cornerRadius
```

The documentation for this struct was generated from the following file:

- src/modules/iui/clay.h

7.9 Clay_TextElementConfig Struct Reference

Public Attributes

- Clay_Color **textColor**
- uint16_t **fontId**
- uint16_t **fontSize**
- uint16_t **letterSpacing**
- uint16_t **lineHeight**
- Clay_TextElementConfigWrapMode **wrapMode**

7.9.1 Member Data Documentation

7.9.1.1 fontId

```
uint16_t Clay_TextElementConfig::fontId
```

7.9.1.2 fontSize

```
uint16_t Clay_TextElementConfig::fontSize
```

7.9.1.3 letterSpacing

```
uint16_t Clay_TextElementConfig::letterSpacing
```

7.9.1.4 lineHeight

```
uint16_t Clay_TextElementConfig::lineHeight
```

7.9.1.5 textColor

```
Clay_Color Clay_TextElementConfig::textColor
```

7.9.1.6 wrapMode

```
Clay_TextElementConfigWrapMode Clay_TextElementConfig::wrapMode
```

The documentation for this struct was generated from the following file:

- src/modules/iui/clay.h

7.10 Cube Struct Reference

Public Member Functions

- **Cube** (GraphicsModule *gfxm, sShader shader)
- **Cube** (GraphicsModule *gfxm, sShader shader, vec3 pos)
- void **draw** (GraphicsModule *gfxm)

Public Attributes

- sMesh **mesh**
- sModelTransform **transform**

The documentation for this struct was generated from the following file:

- src/game/src/game.cpp

7.11 DynamicLibrary Struct Reference

Public Member Functions

- **DynamicLibrary** (const char *path, const char *ident)
- void * **getSymbol** (const char *name)
- bool **valid** ()

Static Public Member Functions

- static char * **makePath** (const char *path, const char *ident)

Public Attributes

- void * **handle**
- const char * **mod_name**
- const char * **mod_imp**

The documentation for this struct was generated from the following file:

- src/modules/moduleLib.h

7.12 DynamicScript Struct Reference

Public Attributes

- void * **handle**

7.12.1 Member Data Documentation

7.12.1.1 handle

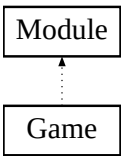
```
void* DynamicScript::handle
```

The documentation for this struct was generated from the following file:

- src/modules/scrld/cppscript.cpp

7.13 Game Class Reference

Inheritance diagram for Game:



Public Attributes

- GameMain **main**

The documentation for this class was generated from the following file:

- src/modules/game.h

7.14 GameContext Struct Reference

Public Attributes

- WindowModule **winm**
- GraphicsModule **gfxm**
- ShaderModule **shdr**
- TextureModule **texm**
- TextModule **textm**
- AssetLoader **assetm**

7.14.1 Member Data Documentation

7.14.1.1 assetm

```
AssetLoader GameContext::assetm
```

7.14.1.2 gfxm

```
GraphicsModule GameContext::gfxm
```

7.14.1.3 shdr

```
ShaderModule GameContext::shdr
```

7.14.1.4 texm

```
TextureModule GameContext::texm
```

7.14.1.5 textm

```
TextModule GameContext::textm
```

7.14.1.6 winm

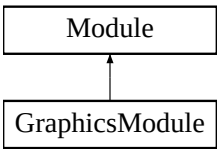
```
WindowModule GameContext::winm
```

The documentation for this struct was generated from the following file:

- src/modules/game.h

7.15 GraphicsModule Struct Reference

Inheritance diagram for GraphicsModule:



Public Member Functions

- void **init** (sWindow *win)
- sMesh **createMesh** (sShader vertexShader, void *vertices, size_t vertexCount, sIndex *indices, size_t indexCount)
- sShaderProgram **createShaderProgram** (sShader *shaders, size_t count)
- sShaderProgram **createShaderProgram** (std::initializer_list< sShader > shaders)
- sVertexDefinition * **createVertexDefinition** (int *elements, size_t count)
- sVertexDefinition * **createVertexDefinition** (std::initializer_list< int > elements)
- void **freeVertexDefinition** (sVertexDefinition *def)
- sShader **createShader** (const char *source, sShaderType type, sVertexDefinition *vertDef)
- sShader **createShader** (const char *source, sShaderType type)
- sShader **loadShader** (const char *path, sShaderType type, sVertexDefinition *vertDef)
- sShader **loadShader** (const char *path, sShaderType type)
- **GraphicsModule** (const char *dynlib)

Public Member Functions inherited from Module

- **Module** (const char *path, const char *ident)

Public Attributes

- graphics::SetClearColor **setClearColor**
- graphics::Clear **clear**
- graphics::Init **internal_init**
- graphics::CreateMesh **internal_createMesh**
- graphics::DrawMesh **drawMesh**
- graphics::UseShaderProgram **useShaderProgram**
- graphics::CreateShaderProgram **internal_createShaderProgram**
- graphics::CreateShader **internal_createShader**
- graphics::Present **present**
- graphics::GetShaderType **getShaderType**
- graphics::CreateUniforms **createUniforms**
- graphics::SetUniforms **setUniforms**
- graphics::CreateTexture **createTexture**
- graphics::UseTexture **useTexture**
- graphics::FreeTexture **freeTexture**
- graphics::FreeShader **freeShader**
- graphics::FreeShaderProgram **freeShaderProgram**
- graphics::FreeMesh **freeMesh**
- graphics::FreeUniforms **freeUniforms**
- graphics::Destroy **destroy**
- graphics::SetScissor **setScissor**
- graphics::EnableScissor **enableScissor**
- graphics::DisableScissor **disableScissor**
- sWindow * **win**

Public Attributes inherited from Module

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.16 mat4 Union Reference

Public Attributes

- float **m** [4][4]
- struct {
  vec4 **x**
  vec4 **y**
  vec4 **z**
  vec4 **w**
  };

7.16.1 Member Data Documentation

7.16.1.1 [struct]

```
struct { ... } mat4
```

7.16.1.2 m

```
float mat4::m[4][4]
```

The documentation for this union was generated from the following file:

- src/modules/math/module.h

7.17 mat4.__unnamed14__ Struct Reference

Public Attributes

- vec4 **x**
- vec4 **y**
- vec4 **z**
- vec4 **w**

7.17.1 Member Data Documentation

7.17.1.1 w

7.17.1.2 x

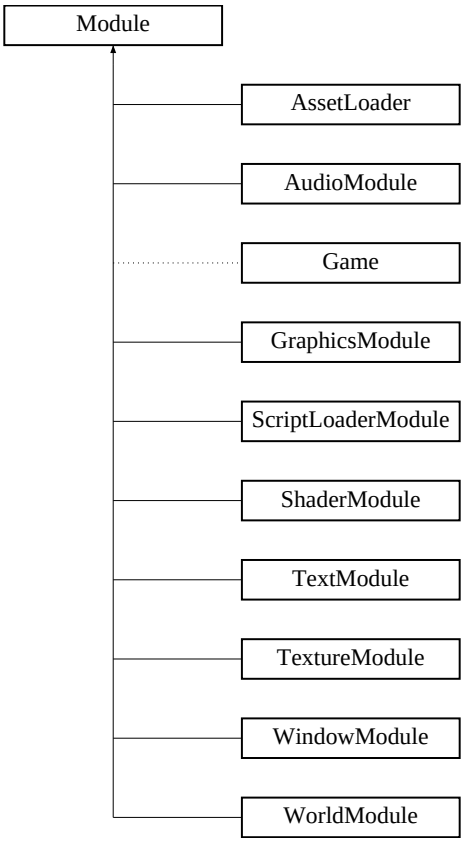
7.17.1.3 y

7.17.1.4 z

The documentation for this struct was generated from the following files:

7.18 Module Struct Reference

Inheritance diagram for Module:



Public Member Functions

- **Module** (const char ∗path, const char ∗ident)

Public Attributes

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/moduleLib.h

7.19 sAudioClip Struct Reference

Public Attributes

- void * **internal**

7.19.1 Member Data Documentation

7.19.1.1 internal

```
void* sAudioClip::internal
```

The documentation for this struct was generated from the following file:

- src/modules/aud/module.h

7.20 sAudioSource Struct Reference

Public Attributes

- void * **internal**
- float **posX**
- float **posY**
- float **posZ**
- float **velX**
- float **velY**
- float **velZ**

7.20.1 Member Data Documentation

7.20.1.1 internal

```
void* sAudioSource::internal
```

7.20.1.2 posX

```
float sAudioSource::posX
```

7.20.1.3 posY

```
float sAudioSource::posY
```

7.20.1.4 posZ

```
float sAudioSource::posZ
```

7.20.1.5 velX

```
float sAudioSource::velX
```

7.20.1.6 velY

```
float sAudioSource::velY
```

7.20.1.7 velZ

```
float sAudioSource::velZ
```

The documentation for this struct was generated from the following file:

- src/modules/aud/module.h

7.21 sCamera Struct Reference

Public Member Functions

- vec3 **right** ()
- vec3 **left** ()
- vec3 **back** ()
- vec3 **down** ()
- vec3 **right** (vec3 forward)
- vec3 **left** (vec3 forward)
- vec3 **back** (vec3 forward)
- vec3 **down** (vec3 up)

Public Attributes

- vec3 **pos** = {0, 0, 0}
- vec3 **up** = {0, 1, 0}
- vec3 **forward** = {0, 0, -1}
- float **yaw** = 0.0f
- float **pitch** = 0.0f

The documentation for this struct was generated from the following file:

- src/modules/math/module.h

7.22 Script Struct Reference

Public Attributes

- void * **internal**
- ScriptInit **init**
- ScriptUpdate **update**

7.22.1 Member Data Documentation

7.22.1.1 init

```
ScriptInit Script::init
```

7.22.1.2 internal

```
void* Script::internal
```

7.22.1.3 update

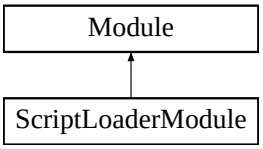
```
ScriptUpdate Script::update
```

The documentation for this struct was generated from the following file:

- src/modules/scrld/module.h

7.23 ScriptLoaderModule Struct Reference

Inheritance diagram for ScriptLoaderModule:



Public Member Functions

- **ScriptLoaderModule** (const char *dynlib)

Public Member Functions inherited from Module

- **Module** (const char *path, const char *ident)

Public Attributes

- scrload::ScriptLoader **loadScript**
- scrload::ScriptCompiler **compileScript**
- scrload::MultiScriptCompiler **compileScripts**
- char * **inputExtension**
- char * **outputExtension**

Public Attributes inherited from Module

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/scrld/module.h

7.24 sD3D11_1Context Struct Reference

Public Attributes

- HWND **hwnd**
- ID3D11Device1 * **device**
- ID3D11DeviceContext1 * **deviceContext**
- IDXGISwapChain1 * **swapChain**
- ID3D11RenderTargetView * **frameBufferView**
- ID3D11DepthStencilView * **depthStencilView**
- ID3D11RasterizerState * **rasterizerState**
- ID3D11DepthStencilState * **depthStencilState**
- ID3D11BlendState * **blendState**
- struct {
  int **x**
  int **y**
  int **width**
  int **height**
  } **scissor**
- sWindow * **win**

7.24.1 Member Data Documentation

7.24.1.1 blendState

```
ID3D11BlendState* sD3D11_1Context::blendState
```

7.24.1.2 depthStencilState

```
ID3D11DepthStencilState* sD3D11_1Context::depthStencilState
```

7.24.1.3 depthStencilView

```
ID3D11DepthStencilView* sD3D11_1Context::depthStencilView
```

7.24.1.4 device

```
ID3D11Device1* sD3D11_1Context::device
```

7.24.1.5 deviceContext

```
ID3D11DeviceContext1* sD3D11_1Context::deviceContext
```

7.24.1.6 frameBufferView

```
ID3D11RenderTargetView* sD3D11_1Context::frameBufferView
```

7.24.1.7 hwnd

```
HWND sD3D11_1Context::hwnd
```

7.24.1.8 rasterizerState

```
ID3D11RasterizerState* sD3D11_1Context::rasterizerState
```

7.24.1.9 [struct]

```
struct { ... } sD3D11_1Context::scissor
```

7.24.1.10 swapChain

```
IDXGISwapChain1* sD3D11_1Context::swapChain
```

7.24.1.11 win

```
sWindow* sD3D11_1Context::win
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/d3d11_1.cpp

7.25 sD3D11_1Context.scissor Struct Reference

Public Attributes

- int **x**
- int **y**
- int **width**
- int **height**

7.25.1 Member Data Documentation

7.25.1.1 height

7.25.1.2 width

7.25.1.3 x

7.25.1.4 y

The documentation for this struct was generated from the following files:

7.26 sFont Struct Reference

Public Attributes

- void * **internal**
- int **size**

7.26.1 Member Data Documentation

7.26.1.1 internal

```
void* sFont::internal
```

7.26.1.2 size

```
int sFont::size
```

The documentation for this struct was generated from the following file:

- src/modules/text/module.h

7.27 sFreeTypeContext Struct Reference

Public Attributes

- FT_Library **ft**
- GraphicsModule * **gfxm**
- ShaderModule * **shdr**
- AssetLoader * **assetm**

7.27.1 Member Data Documentation

7.27.1.1 assetm

```
AssetLoader* sFreeTypeContext::assetm
```

7.27.1.2 ft

```
FT_Library sFreeTypeContext::ft
```

7.27.1.3 gfxm

```
GraphicsModule* sFreeTypeContext::gfxm
```

7.27.1.4 shdr

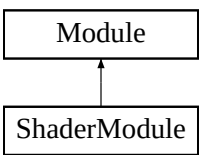
```
ShaderModule* sFreeTypeContext::shdr
```

The documentation for this struct was generated from the following file:

- src/modules/text/freetype.cpp

7.28 ShaderModule Struct Reference

Inheritance diagram for ShaderModule:



Public Member Functions

- sShader **compile** (GraphicsModule *gfxm, const char *path, sShaderType type, sVertexDefinition *vert↩ Def=nullptr)
- sShader **createShader** (GraphicsModule *gfxm, const char *data, size_t len, sShaderType type, sVertexDefinition *vertDef=nullptr)
- **ShaderModule** (const char *dynlib, const char *dynp2)

Public Member Functions inherited from Module

- **Module** (const char *path, const char *ident)

Public Attributes

- shader::Compile **internal_compile**
- shader::CreateShader **internal_createShader**

Public Attributes inherited from Module

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/shdr/module.h

7.29 sInternalFont Struct Reference

Classes

- struct CharacterDef

Public Attributes

- sVertexDefinition * **vertDef**
- sShaderProgram **shader**
- sShader **vertexShader**
- sUniforms **uniforms**
- sTexture **atlas**
- struct sInternalFont::CharacterDef **characters** [128]
- int **atlasWidth**
- int **atlasHeight**
- float **scale**

7.29.1 Class Documentation

7.29.1.1 struct sInternalFont::CharacterDef

Class Members

double	advance	
vec2	bearing	
vec2	offset	
vec2	size	

7.29.2 Member Data Documentation

7.29.2.1 atlas

```
sTexture sInternalFont::atlas
```

7.29.2.2 atlasHeight

```
int sInternalFont::atlasHeight
```

7.29.2.3 atlasWidth

```
int sInternalFont::atlasWidth
```

7.29.2.4 characters

```
struct sInternalFont::CharacterDef sInternalFont::characters[128]
```

7.29.2.5 scale

```
float sInternalFont::scale
```

7.29.2.6 shader

```
sShaderProgram sInternalFont::shader
```

7.29.2.7 uniforms

```
sUniforms sInternalFont::uniforms
```

7.29.2.8 vertDef

```
sVertexDefinition* sInternalFont::vertDef
```

7.29.2.9 vertexShader

```
sShader sInternalFont::vertexShader
```

The documentation for this struct was generated from the following file:

- src/modules/text/freetype.cpp

7.30 sInternalMesh Struct Reference

Public Attributes

- ID3D11Buffer * **vertexBuffer**
- ID3D11Buffer * **indexBuffer**
- UINT **stride**
- UINT **offset**
- size_t **numIndices**
- unsigned int **vao**
- unsigned int **vbo**
- unsigned int **ebo**
- size_t **indexCount**

7.30.1 Member Data Documentation

7.30.1.1 ebo

```
unsigned int sInternalMesh::ebo
```

7.30.1.2 indexBuffer

```
ID3D11Buffer* sInternalMesh::indexBuffer
```

7.30.1.3 indexCount

```
size_t sInternalMesh::indexCount
```

7.30.1.4 numIndices

```
size_t sInternalMesh::numIndices
```

7.30.1.5 offset

```
UINT sInternalMesh::offset
```

7.30.1.6 stride

```
UINT sInternalMesh::stride
```

7.30.1.7 vao

```
unsigned int sInternalMesh::vao
```

7.30.1.8 vbo

```
unsigned int sInternalMesh::vbo
```

7.30.1.9 vertexBuffer

```
ID3D11Buffer* sInternalMesh::vertexBuffer
```

The documentation for this struct was generated from the following files:

- src/modules/gfx/d3d11_1.cpp
- src/modules/gfx/glad.cpp

7.31 sInternalRectUniforms Struct Reference

Public Attributes

- vec4 **color**
- mat4 **proj**
- mat4 **view**
- mat4 **model**
- float **z**

7.31.1 Member Data Documentation

7.31.1.1 color

```
vec4 sInternalRectUniforms::color
```

7.31.1.2 model

```
mat4 sInternalRectUniforms::model
```

7.31.1.3 proj

```
mat4 sInternalRectUniforms::proj
```

7.31.1.4 view

```
mat4 sInternalRectUniforms::view
```

7.31.1.5 z

```
float sInternalRectUniforms::z
```

The documentation for this struct was generated from the following file:

- src/modules/iui/module.h

7.32 sInternalRectVertex Struct Reference

Public Attributes

- vec2 **pos**

7.32.1 Member Data Documentation

7.32.1.1 pos

```
vec2 sInternalRectVertex::pos
```

The documentation for this struct was generated from the following file:

- src/modules/iui/module.h

7.33 sInternalRoundedRectUniforms Struct Reference

Public Attributes

- vec4 **color**
- vec2 **topleft**
- vec2 **widheight**
- float **radius**
- mat4 **proj**
- mat4 **view**
- mat4 **model**
- float **z**

7.33.1 Member Data Documentation

7.33.1.1 color

```
vec4 sInternalRoundedRectUniforms::color
```

7.33.1.2 model

```
mat4 sInternalRoundedRectUniforms::model
```

7.33.1.3 proj

```
mat4 sInternalRoundedRectUniforms::proj
```

7.33.1.4 radius

```
float sInternalRoundedRectUniforms::radius
```

7.33.1.5 topleft

```
vec2 sInternalRoundedRectUniforms::topleft
```

7.33.1.6 view

```
mat4 sInternalRoundedRectUniforms::view
```

7.33.1.7 widheight

```
vec2 sInternalRoundedRectUniforms::widheight
```

7.33.1.8 z

```
float sInternalRoundedRectUniforms::z
```

The documentation for this struct was generated from the following file:

- src/modules/iui/module.h

7.34 sInternalShader Struct Reference

Public Attributes

- enum sShaderType **type**
- ID3D11VertexShader * **vertexShader**
- ID3D11PixelShader * **pixelShader**
- ID3D11GeometryShader * **geometryShader**
- ID3DBlob * **shaderBlob**
- sVertexDefinition * **vertDef**
- unsigned int **shader**

7.34.1 Member Data Documentation

7.34.1.1 geometryShader

```
ID3D11GeometryShader* sInternalShader::geometryShader
```

7.34.1.2 pixelShader

```
ID3D11PixelShader* sInternalShader::pixelShader
```

7.34.1.3 shader

```
unsigned int sInternalShader::shader
```

7.34.1.4 shaderBlob

```
ID3DBlob* sInternalShader::shaderBlob
```

7.34.1.5 type

```
enum sShaderType sInternalShader::type
```

7.34.1.6 vertDef

```
sVertexDefinition * sInternalShader::vertDef
```

7.34.1.7 vertexShader

```
ID3D11VertexShader* sInternalShader::vertexShader
```

The documentation for this struct was generated from the following files:

- src/modules/gfx/d3d11_1.cpp
- src/modules/gfx/eogll.cpp
- src/modules/gfx/glad.cpp

7.35 sInternalShaderProgram Struct Reference

Public Attributes

- sInternalShader **vertexShader**
- sInternalShader **fragmentShader**
- ID3D11InputLayout * **inputLayout**
- size_t **textureCount**
- unsigned int **program**
- int **texcount**

7.35.1 Member Data Documentation

7.35.1.1 fragmentShader

```
sInternalShader sInternalShaderProgram::fragmentShader
```

7.35.1.2 inputLayout

```
ID3D11InputLayout* sInternalShaderProgram::inputLayout
```

7.35.1.3 program

```
unsigned int sInternalShaderProgram::program
```

7.35.1.4 texcount

```
int sInternalShaderProgram::texcount
```

7.35.1.5 textureCount

```
size_t sInternalShaderProgram::textureCount
```

7.35.1.6 vertexShader

```
sInternalShader sInternalShaderProgram::vertexShader
```

The documentation for this struct was generated from the following files:

- src/modules/gfx/d3d11_1.cpp
- src/modules/gfx/eogll.cpp
- src/modules/gfx/glad.cpp

7.36 sInternalText Struct Reference

Public Attributes

- sInternalFont * **font**
- char * **text**
- TextUniforms **uniforms**
- size_t **vertexCount**
- TextVertex * **vertices**
- sMesh **mesh**

7.36.1 Member Data Documentation

7.36.1.1 font

```
sInternalFont* sInternalText::font
```

7.36.1.2 mesh

```
sMesh sInternalText::mesh
```

7.36.1.3 text

```
char* sInternalText::text
```

7.36.1.4 uniforms

```
TextUniforms sInternalText::uniforms
```

7.36.1.5 vertexCount

```
size_t sInternalText::vertexCount
```

7.36.1.6 vertices

```
TextVertex* sInternalText::vertices
```

The documentation for this struct was generated from the following file:

- src/modules/text/freetype.cpp

7.37 sInternalTexture Struct Reference

Public Attributes

- ID3D11ShaderResourceView * **texture**
- ID3D11SamplerState * **sampler**
- unsigned int **texture**

7.37.1 Member Data Documentation

7.37.1.1 sampler

```
ID3D11SamplerState* sInternalTexture::sampler
```

7.37.1.2 texture [1/2]

```
unsigned int sInternalTexture::texture
```

7.37.1.3 texture [2/2]

```
unsigned int sInternalTexture::texture
```

The documentation for this struct was generated from the following files:

- src/modules/gfx/d3d11_1.cpp
- src/modules/gfx/eogll.cpp
- src/modules/gfx/glad.cpp

7.38 sInternalUniforms Struct Reference

Public Attributes

- sUniformDefinition **fragmentPart**
- sUniformDefinition **vertexPart**
- sShaderProgram **program**
- ID3D11Buffer * **fragmentBuffer**
- ID3D11Buffer * **vertexBuffer**
- sUniformDefinition **def**
- int * **locations**

7.38.1 Member Data Documentation

7.38.1.1 def

```
sUniformDefinition sInternalUniforms::def
```

7.38.1.2 fragmentBuffer

```
ID3D11Buffer* sInternalUniforms::fragmentBuffer
```

7.38.1.3 fragmentPart

```
sUniformDefinition sInternalUniforms::fragmentPart
```

7.38.1.4 locations

```
int * sInternalUniforms::locations
```

7.38.1.5 program

```
sShaderProgram sInternalUniforms::program
```

7.38.1.6 vertexBuffer

```
ID3D11Buffer* sInternalUniforms::vertexBuffer
```

7.38.1.7 vertexPart

```
sUniformDefinition sInternalUniforms::vertexPart
```

The documentation for this struct was generated from the following files:

- src/modules/gfx/d3d11_1.cpp
- src/modules/gfx/eogll.cpp
- src/modules/gfx/glad.cpp

7.39 sIUIGlobalState Struct Reference

Public Attributes

- WindowModule * **winm**
- GraphicsModule * **gfxm**
- TextModule * **textm**
- ShaderModule * **shdr**
- sShaderProgram **rect_shader**
- sVertexDefinition * **rect_vert_def**
- sMesh **rect_mesh**
- sUniforms **rect_uniforms**
- sShaderProgram **rounded_rect_shader**
- sUniforms **rounded_rect_uniforms**
- sWindow * **win**
- sFont * **fonts** =nullptr

7.39.1 Member Data Documentation

7.39.1.1 fonts

```
sFont* sIUIGlobalState::fonts =nullptr
```

7.39.1.2 gfxm

```
GraphicsModule* sIUIGlobalState::gfxm
```

7.39.1.3 rect_mesh

```
sMesh sIUIGlobalState::rect_mesh
```

7.39.1.4 rect_shader

```
sShaderProgram sIUIGlobalState::rect_shader
```

7.39.1.5 rect_uniforms

```
sUniforms sIUIGlobalState::rect_uniforms
```

7.39.1.6 rect_vert_def

```
sVertexDefinition* sIUIGlobalState::rect_vert_def
```

7.39.1.7 rounded_rect_shader

```
sShaderProgram sIUIGlobalState::rounded_rect_shader
```

7.39.1.8 rounded_rect_uniforms

```
sUniforms sIUIGlobalState::rounded_rect_uniforms
```

7.39.1.9 shdr

```
ShaderModule* sIUIGlobalState::shdr
```

7.39.1.10 textm

```
TextModule* sIUIGlobalState::textm
```

7.39.1.11 win

```
sWindow* sIUIGlobalState::win
```

7.39.1.12 winm

```
WindowModule* sIUIGlobalState::winm
```

The documentation for this struct was generated from the following file:

- src/modules/iui/module.h

7.40 sMesh Struct Reference

Public Attributes

- void * **internal**
- GraphicsModule * **creator**

7.40.1 Member Data Documentation

7.40.1.1 creator

```
GraphicsModule* sMesh::creator
```

7.40.1.2 internal

```
void* sMesh::internal
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.41 sModelTransform Struct Reference

Public Member Functions

- mat4 **matrix** ()

Public Attributes

- vec3 **pos** = {0, 0, 0}
- vec3 **sca** = {1, 1, 1}
- vec3 **rot** = {0, 0, 0}
- vec3 **lastPos** = {0, 0, 0}
- vec3 **lastSca** = {1, 1, 1}
- vec3 **lastRot** = {0, 0, 0}
- mat4 **internal_matrix** = identity()

The documentation for this struct was generated from the following file:

- src/modules/math/module.h

7.42 sShader Struct Reference

Public Attributes

- void * **internal**

7.42.1 Member Data Documentation

7.42.1.1 internal

```
void* sShader::internal
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.43 sShaderProgram Struct Reference

Public Attributes

- void * **internal**
- GraphicsModule * **creator**
- void * **gfx_internal**

7.43.1 Member Data Documentation

7.43.1.1 creator

```
GraphicsModule* sShaderProgram::creator
```

7.43.1.2 gfx_internal

```
void* sShaderProgram::gfx_internal
```

7.43.1.3 internal

```
void* sShaderProgram::internal
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.44 sText Struct Reference

Public Attributes

- void * **internal**

7.44.1 Member Data Documentation

7.44.1.1 internal

```
void* sText::internal
```

The documentation for this struct was generated from the following file:

- src/modules/text/module.h

7.45 sTexture Struct Reference

Public Attributes

- void * **internal**

7.45.1 Member Data Documentation

7.45.1.1 internal

```
void* sTexture::internal
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.46 sTextureDefinition Struct Reference

Public Attributes

- size_t **width**
- size_t **height**
- size_t **channels**
- unsigned char * **data**

7.46.1 Member Data Documentation

7.46.1.1 channels

```
size_t sTextureDefinition::channels
```

7.46.1.2 data

```
unsigned char* sTextureDefinition::data
```

7.46.1.3 height

```
size_t sTextureDefinition::height
```

7.46.1.4 width

```
size_t sTextureDefinition::width
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.47 sUniformDefinition Struct Reference

Public Member Functions

- **sUniformDefinition** (std::initializer_list< sUniformElement > elements)
- size_t **size** ()
- **sUniformDefinition** (sUniformElement *elements, size_t count)

Public Attributes

- sUniformElement * **elements**
- size_t **count**

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.48 sUniformElement Struct Reference

Public Member Functions

- **sUniformElement** (sShaderType shaderType, const char *name, sUniformType type, size_t countx, size_t county)
- **sUniformElement** (sShaderType shaderType, const char *name, sUniformType type, size_t countx)

Public Attributes

- sShaderType **shaderType**
- const char * **name**
- sUniformType **type**
- size_t **countx**
- size_t **county** =1

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.49 sUniforms Struct Reference

Public Attributes

- void * **internal**

7.49.1 Member Data Documentation

7.49.1.1 internal

```
void* sUniforms::internal
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.50 sVertexDefinition Struct Reference

Public Attributes

- int * **elements**
- size_t **count**

7.50.1 Member Data Documentation

7.50.1.1 count

```
size_t sVertexDefinition::count
```

7.50.1.2 elements

```
int* sVertexDefinition::elements
```

The documentation for this struct was generated from the following file:

- src/modules/gfx/module.h

7.51 swAudio Struct Reference

Public Attributes

- std::string **path**
- std::vector< uint8_t > **data**

7.51.1 Member Data Documentation

7.51.1.1 data

```
std::vector<uint8_t> swAudio::data
```

7.51.1.2 path

```
std::string swAudio::path
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.52 swEtc Struct Reference

Public Types

- enum **swEtcType** { **PRE_LOAD** , **PRE_LOOP** , **POST_LOOP** , **POST_GAME** }

Public Attributes

- enum swEtc::swEtcType **type**

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.53 swGame Struct Reference

Public Attributes

- swWorld **world**
- std::vector< swModel > **models**
- std::vector< swMaterial > **materials**
- std::vector< swLevel > **levels**
- std::vector< swScript > **scripts**
- std::vector< swEtc > **etc**
- std::vector< swTexture > **textures**
- std::vector< swAudio > **audio**

7.53.1 Member Data Documentation

7.53.1.1 audio

```
std::vector<swAudio> swGame::audio
```

7.53.1.2 etc

```
std::vector<swEtc> swGame::etc
```

7.53.1.3 levels

```
std::vector<swLevel> swGame::levels
```

7.53.1.4 materials

```
std::vector<swMaterial> swGame::materials
```

7.53.1.5 models

```
std::vector<swModel> swGame::models
```

7.53.1.6 scripts

```
std::vector<swScript> swGame::scripts
```

7.53.1.7 textures

```
std::vector<swTexture> swGame::textures
```

7.53.1.8 world

```
swWorld swGame::world
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.54 swLevel Struct Reference

Public Attributes

- std::vector< swLevelObject > **objects**

7.54.1 Member Data Documentation

7.54.1.1 objects

```
std::vector<swLevelObject> swLevel::objects
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.55 swLevelObject Struct Reference

Public Attributes

- sModelTransform **transform**
- void * **ecsObject**

7.55.1 Member Data Documentation

7.55.1.1 ecsObject

```
void* swLevelObject::ecsObject
```

7.55.1.2 transform

```
sModelTransform swLevelObject::transform
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.56 swMaterial Struct Reference

Public Attributes

- std::string **shader**
- std::vector< std::string > **samplers**
- std::vector< std::string > **vertexUniforms**
- std::vector< std::string > **fragmentUniforms**

7.56.1 Member Data Documentation

7.56.1.1 fragmentUniforms

```
std::vector<std::string> swMaterial::fragmentUniforms
```

7.56.1.2 samplers

```
std::vector<std::string> swMaterial::samplers
```

7.56.1.3 shader

```
std::string swMaterial::shader
```

7.56.1.4 vertexUniforms

```
std::vector<std::string> swMaterial::vertexUniforms
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.57 swModel Struct Reference

Public Attributes

- std::vector< float > **vertices**
- std::vector< unsigned int > **indices**

7.57.1 Member Data Documentation

7.57.1.1 indices

```
std::vector<unsigned int> swModel::indices
```

7.57.1.2 vertices

```
std::vector<float> swModel::vertices
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.58 swScript Struct Reference

Public Attributes

- std::string **ext**
- std::string **mod**

7.58.1 Member Data Documentation

7.58.1.1 ext

```
std::string swScript::ext
```

7.58.1.2 mod

```
std::string swScript::mod
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.59 swTexture Struct Reference

Public Attributes

- std::string **path**
- std::vector< uint8_t > **data**

7.59.1 Member Data Documentation

7.59.1.1 data

```
std::vector<uint8_t> swTexture::data
```

7.59.1.2 path

```
std::string swTexture::path
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.60 swWorld Struct Reference

Public Attributes

- std::string **name**
- std::string **author**
- std::string **description**

7.60.1 Member Data Documentation

7.60.1.1 author

```
std::string swWorld::author
```

7.60.1.2 description

```
std::string swWorld::description
```

7.60.1.3 name

```
std::string swWorld::name
```

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

7.61 TextAssetBuffer Struct Reference

Public Attributes

- const char * **data**
- size_t **len**

7.61.1 Member Data Documentation

7.61.1.1 data

```
const char* TextAssetBuffer::data
```

7.61.1.2 len

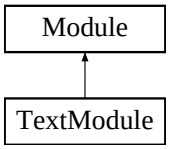
```
size_t TextAssetBuffer::len
```

The documentation for this struct was generated from the following file:

- src/modules/asset.h

7.62 TextModule Struct Reference

Inheritance diagram for TextModule:



Public Member Functions

- sFont **loadFont** (const char ∗path, int size, const char ∗vertpath, const char ∗fragpath)
- sFont **loadFontAsset** (const char ∗path, int size, const char ∗vertpath, const char ∗fragpath)
- **TextModule** (const char ∗path)

Public Member Functions inherited from Module

- **Module** (const char ∗path, const char ∗ident)

Public Attributes

- text::Init **init**
- text::LoadFont **internal_loadFont**
- text::LoadFontAsset **internal_loadFontAsset**
- text::CreateText **createText**
- text::DrawText **drawText**
- text::FreeText **freeText**
- text::FreeFont **freeFont**
- text::SetTextColor **setTextColor**
- text::SetTextModel **setTextModel**
- text::SetTextView **setTextView**
- text::SetTextProj **setTextProj**
- text::MeasureText **measureText**
- text::SetTextZ **setTextZ**

Public Attributes inherited from Module

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/text/module.h

7.63 TextUniforms Struct Reference

Public Attributes

- vec3 **color**
- mat4 **proj**
- mat4 **view**
- mat4 **model**
- float **z**

7.63.1 Member Data Documentation

7.63.1.1 color

```
vec3 TextUniforms::color
```

7.63.1.2 model

```
mat4 TextUniforms::model
```

7.63.1.3 proj

```
mat4 TextUniforms::proj
```

7.63.1.4 view

```
mat4 TextUniforms::view
```

7.63.1.5 z

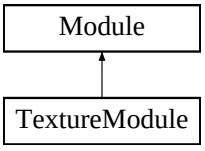
```
float TextUniforms::z
```

The documentation for this struct was generated from the following file:

- src/modules/text/freetype.cpp

7.64 TextureModule Struct Reference

Inheritance diagram for TextureModule:



Public Member Functions

- **TextureModule** (const char ∗path)

Public Member Functions inherited from Module

- **Module** (const char ∗path, const char ∗ident)

Public Attributes

- texload::LoadTexture **loadTexture**
- texload::FreeTexture **freeTexture**
- texload::LoadTextureFromBuffer **loadTextureFromBuffer**

Public Attributes inherited from Module

- DynamicLibrary **lib**

The documentation for this struct was generated from the following file:

- src/modules/tex/module.h

7.65 TextVertex Struct Reference

Public Attributes

- vec2 **pos**
- vec2 **uv**

7.65.1 Member Data Documentation

7.65.1.1 pos

```
vec2 TextVertex::pos
```

7.65.1.2 uv

```
vec2 TextVertex::uv
```

The documentation for this struct was generated from the following file:

- src/modules/text/freetype.cpp

7.66 vec2 Union Reference

Public Member Functions

- **vec2** (float x, float y)
- **vec2** (int x, int y)
- **vec2** (unsigned int x, unsigned int y)

Public Attributes

- struct {
  float **x**
  float **y**
  };
- struct {
  float **u**
  float **v**
  };
- float **f** [2]

The documentation for this union was generated from the following file:

- src/modules/math/module.h

7.67 vec2.__unnamed10__ Struct Reference

Public Attributes

- float **x**
- float **y**

7.67.1 Member Data Documentation

7.67.1.1 x

7.67.1.2 y

The documentation for this struct was generated from the following files:

7.68 vec2.__unnamed12__ Struct Reference

Public Attributes

- float **u**
- float **v**

7.68.1 Member Data Documentation

7.68.1.1 u

7.68.1.2 v

The documentation for this struct was generated from the following files:

7.69 vec3 Union Reference

Public Attributes

- struct {
  float **x**
  float **y**
  float **z**
  };
- struct {
  float **r**
  float **g**
  float **b**
  };
- float **v** [3]

7.69.1 Member Data Documentation

7.69.1.1 [struct]

```
struct { ... } vec3
```

7.69.1.2 [struct]

```
struct { ... } vec3
```

7.69.1.3 v

```
float vec3::v[3]
```

The documentation for this union was generated from the following file:

- src/modules/math/module.h

7.70 vec3.__unnamed2__ Struct Reference

Public Attributes

- float **x**
- float **y**
- float **z**

7.70.1 Member Data Documentation

7.70.1.1 x

7.70.1.2 y

7.70.1.3 z

The documentation for this struct was generated from the following files:

7.71 vec3.__unnamed4__ Struct Reference

Public Attributes

- float **r**
- float **g**
- float **b**

7.71.1 Member Data Documentation

7.71.1.1 b

7.71.1.2 g

7.71.1.3 r

The documentation for this struct was generated from the following files:

7.72 vec4 Union Reference

Public Attributes

- struct {
  float **x**
  float **y**
  float **z**
  float **w**
  };
- struct {
  float **r**
  float **g**
  float **b**
  float **a**
  };
- float **v** [4]

7.72.1 Member Data Documentation

7.72.1.1 [struct]

```
struct { ... } vec4
```

7.72.1.2 [struct]

```
struct { ... } vec4
```

7.72.1.3 v

```
float vec4::v[4]
```

The documentation for this union was generated from the following file:

- src/modules/math/module.h

7.73 vec4.__unnamed6__ Struct Reference

Public Attributes

- float **x**
- float **y**
- float **z**
- float **w**

7.73.1 Member Data Documentation

7.73.1.1 w

7.73.1.2 x

7.73.1.3 y

7.73.1.4 z

The documentation for this struct was generated from the following files:

7.74 vec4.__unnamed8__ Struct Reference

Public Attributes

- float **r**
- float **g**
- float **b**
- float **a**

7.74.1 Member Data Documentation

7.74.1.1 a

7.74.1.2 b

7.74.1.3 g

7.74.1.4 r

The documentation for this struct was generated from the following files:

7.75 Vertex Struct Reference

Public Attributes

- float **position** [3]
- float **normal** [3]
- float **texcoord** [2]

7.75.1 Member Data Documentation

7.75.1.1 normal

```
float Vertex::normal[3]
```

7.75.1.2 position

```
float Vertex::position[3]
```

7.75.1.3 texcoord

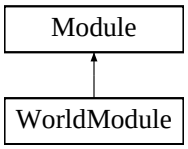
```
float Vertex::texcoord[2]
```

The documentation for this struct was generated from the following file:

- src/game/src/cube.h

7.76 WorldModule Struct Reference

Inheritance diagram for WorldModule:



Public Attributes

- world::LoadGame **loadGame**
- world::FreeGame **freeGame**
- world::SaveGame **saveGame**

Public Attributes inherited from Module

- DynamicLibrary **lib**

Additional Inherited Members

Public Member Functions inherited from Module

- **Module** (const char ∗path, const char ∗ident)

The documentation for this struct was generated from the following file:

- src/modules/wrld/module.h

Chapter 8

File Documentation

8.1 cube.h

```
00001 #include "gfx/module.h"
00002 #pragma pack(1)
00003 struct Vertex {
00004     float position[3];
00005     float normal[3];
00006     float texcoord[2];
00007 } vertices[] = {
00008     {{-1.0f, -1.0f,  1.0f}, { 0.0f,  0.0f,  1.0f}, {0.0f, 0.0f}},
00009     {{ 1.0f, -1.0f,  1.0f}, { 0.0f,  0.0f,  1.0f}, {1.0f, 0.0f}},
00010     {{ 1.0f,  1.0f,  1.0f}, { 0.0f,  0.0f,  1.0f}, {1.0f, 1.0f}},
00011     {{-1.0f,  1.0f,  1.0f}, { 0.0f,  0.0f,  1.0f}, {0.0f, 1.0f}},
00012
00013     {{-1.0f, -1.0f, -1.0f}, { 0.0f,  0.0f, -1.0f}, {0.0f, 0.0f}},
00014     {{ 1.0f, -1.0f, -1.0f}, { 0.0f,  0.0f, -1.0f}, {1.0f, 0.0f}},
00015     {{ 1.0f,  1.0f, -1.0f}, { 0.0f,  0.0f, -1.0f}, {1.0f, 1.0f}},
00016     {{-1.0f,  1.0f, -1.0f}, { 0.0f,  0.0f, -1.0f}, {0.0f, 1.0f}},
00017
00018     {{-1.0f,  1.0f,  1.0f}, { 0.0f,  1.0f,  0.0f}, {0.0f, 0.0f}},
00019     {{ 1.0f,  1.0f,  1.0f}, { 0.0f,  1.0f,  0.0f}, {1.0f, 0.0f}},
00020     {{ 1.0f,  1.0f, -1.0f}, { 0.0f,  1.0f,  0.0f}, {1.0f, 1.0f}},
00021     {{-1.0f,  1.0f, -1.0f}, { 0.0f,  1.0f,  0.0f}, {0.0f, 1.0f}},
00022
00023     {{-1.0f, -1.0f,  1.0f}, { 0.0f, -1.0f,  0.0f}, {0.0f, 0.0f}},
00024     {{ 1.0f, -1.0f,  1.0f}, { 0.0f, -1.0f,  0.0f}, {1.0f, 0.0f}},
00025     {{ 1.0f, -1.0f, -1.0f}, { 0.0f, -1.0f,  0.0f}, {1.0f, 1.0f}},
00026     {{-1.0f, -1.0f, -1.0f}, { 0.0f, -1.0f,  0.0f}, {0.0f, 1.0f}},
00027
00028     {{ 1.0f, -1.0f,  1.0f}, { 1.0f,  0.0f,  0.0f}, {0.0f, 0.0f}},
00029     {{ 1.0f, -1.0f, -1.0f}, { 1.0f,  0.0f,  0.0f}, {1.0f, 0.0f}},
00030     {{ 1.0f,  1.0f, -1.0f}, { 1.0f,  0.0f,  0.0f}, {1.0f, 1.0f}},
00031     {{ 1.0f,  1.0f,  1.0f}, { 1.0f,  0.0f,  0.0f}, {0.0f, 1.0f}},
00032
00033     {{-1.0f, -1.0f,  1.0f}, {-1.0f,  0.0f,  0.0f}, {0.0f, 0.0f}},
00034     {{-1.0f, -1.0f, -1.0f}, {-1.0f,  0.0f,  0.0f}, {1.0f, 0.0f}},
00035     {{-1.0f,  1.0f, -1.0f}, {-1.0f,  0.0f,  0.0f}, {1.0f, 1.0f}},
00036     {{-1.0f,  1.0f,  1.0f}, {-1.0f,  0.0f,  0.0f}, {0.0f, 1.0f}}
00037 };
00038
00039 sIndex indices[] = {
00040         0,  1,  2,   2,  3,  0,
00041         6,  5,  4,   4,  7,  6,
00042         8,  9, 10,  10, 11,  8,
00043     14, 13, 12,  12, 15, 14,
00044     16, 17, 18,  18, 19, 16,
00045     22, 21, 20,  20, 23, 22
00046 };
```

8.2 asset.h

```
00001 #pragma once
00002
00003 #include <cstdint>
00004 #include <string>
00005 #include "moduleLib.h"
```

```
00006
00007 std::string replace(std::string src, char from, char to) {
00008     for (size_t i = 0; i < src.size(); i++) {
00009         if (src[i] == from) {
00010             src[i] = to;
00011         }
00012     }
00013     return src;
00014 }
00015
00016 uint64_t hash(std::string str) {
00017     uint64_t hash = 5381;
00018     for (char c : str) {
00019         hash = ((hash « 5) + hash) + c;
00020     }
00021     return hash;
00022 }
00023
00024 struct AssetBuffer {
00025     const uint8_t* data;
00026     size_t len;
00027 };
00028
00029 struct TextAssetBuffer {
00030     const char* data;
00031     size_t len;
00032 };
00033
00034 struct AssetLoader : Module {
00035
00036     AssetLoader() : Module("assets", "game") {
00037         // assets are stored in a module, basically a big cpp file with a bunch of variables
00038     }
00039
00040     AssetBuffer loadAsset(const char* path) {
00041         // TODO: Should I use a hash instead of a string?
00042         // we will try strings first
00043
00044         std::string assetPath = replace(path, '/', '_');
00045         assetPath = replace(assetPath, '.', '_');
00046         assetPath = replace(assetPath, '-', '_');
00047         assetPath = replace(assetPath, ' ', '_');
00048         assetPath = replace(assetPath, '\\', '_');
00049         assetPath = replace(assetPath, ':', '_');
00050         assetPath = replace(assetPath, '?', '_');
00051         assetPath = replace(assetPath, '&', '_');
00052         assetPath = replace(assetPath, '=', '_');
00053         assetPath = replace(assetPath, '+', '_');
00054         assetPath = replace(assetPath, '%', '_');
00055         assetPath = replace(assetPath, '#', '_');
00056         assetPath = replace(assetPath, '!', '_');
00057         assetPath = replace(assetPath, '@', '_');
00058         assetPath = replace(assetPath, '$', '_');
00059         assetPath = replace(assetPath, '^', '_');
00060         assetPath = replace(assetPath, '*', '_');
00061         assetPath = replace(assetPath, '(', '_');
00062         assetPath = replace(assetPath, ')', '_');
00063         assetPath = replace(assetPath, '[', '_');
00064         assetPath = replace(assetPath, ']', '_');
00065         assetPath = replace(assetPath, '{', '_');
00066         assetPath = replace(assetPath, '}', '_');
00067         assetPath = replace(assetPath, '<', '_');
00068         assetPath = replace(assetPath, '>', '_');
00069         assetPath = replace(assetPath, '|', '_');
00070         assetPath = replace(assetPath, ';', '_');
00071         assetPath = replace(assetPath, ',', '_');
00072
00073         std::string assetVar = "asset_" + assetPath;
00074         std::string assetSizeVar = "asset_" + assetPath + "_size";
00075
00076         // inside of the binary it is declared as extern "C" const uint8_t asset_<path>[];
00077         // and extern "C" const size_t asset_<path>_size;
00078         const uint8_t* data = (const uint8_t*)lib.getSymbol(assetVar.c_str());
00079         size_t len = *(size_t*)lib.getSymbol(assetSizeVar.c_str());
00080
00081         return {data, len};
00082     }
00083
00084     TextAssetBuffer loadTextAsset(const char* path) {
00085         AssetBuffer abuf = loadAsset(path);
00086         return {(const char*)abuf.data, abuf.len};
00087     }
00088 };
```

8.3 module.h

```
00001 #pragma once
00002 #include "../moduleLib.h"
00003
00004 struct sAudioClip {
00005     void* internal;
00006 };
00007
00008 struct sAudioSource {
00009     void* internal;
00010     float posX, posY, posZ;
00011     float velX, velY, velZ;
00012 };
00013
00014 namespace audio {
00015     typedef void (*Init)();
00016     typedef sAudioClip (*LoadAudioClip)(const char* path);
00017     typedef sAudioSource (*CreateAudioSource)(sAudioClip clip);
00018     typedef void (*PlayAudioSource)(sAudioSource source);
00019     typedef void (*StopAudioSource)(sAudioSource source);
00020     typedef void (*SetAudioSourcePosition)(sAudioSource source, float x, float y, float z);
00021     typedef void (*SetAudioSourceVelocity)(sAudioSource source, float x, float y, float z);
00022     typedef void (*SetAudioSourcePitch)(sAudioSource source, float pitch);
00023     typedef void (*SetAudioSourceGain)(sAudioSource source, float gain);
00024     typedef void (*SetAudioSourceLooping)(sAudioSource source, bool looping);
00025     typedef void (*SeekAudioSourceSamples)(sAudioSource source, int samples);
00026     typedef int (*GetAudioSourceSamples)(sAudioSource source);
00027     typedef int (*GetAudioSourceSampleRate)(sAudioSource source);
00028     typedef void (*DestroyAudioClip)(sAudioClip clip);
00029     typedef void (*DestroyAudioSource)(sAudioSource source);
00030     typedef void (*Destroy)();
00031
00032 }
00033
00034 struct AudioModule : public Module {
00035     audio::Init init;
00036     audio::LoadAudioClip loadAudioClip;
00037     audio::CreateAudioSource createAudioSource;
00038     audio::PlayAudioSource playAudioSource;
00039     audio::StopAudioSource stopAudioSource;
00040     audio::SetAudioSourcePosition setAudioSourcePosition;
00041     audio::SetAudioSourceVelocity setAudioSourceVelocity;
00042     audio::SetAudioSourcePitch setAudioSourcePitch;
00043     audio::SetAudioSourceGain setAudioSourceGain;
00044     audio::SetAudioSourceLooping setAudioSourceLooping;
00045     audio::SeekAudioSourceSamples seekAudioSourceSamples;
00046     audio::GetAudioSourceSamples getAudioSourceSamples;
00047     audio::GetAudioSourceSampleRate getAudioSourceSampleRate;
00048     audio::DestroyAudioClip destroyAudioClip;
00049     audio::DestroyAudioSource destroyAudioSource;
00050     audio::Destroy destroy;
00051
00052     void seekAudioSourceSeconds(sAudioSource source, float seconds) {
00053         seekAudioSourceSamples(source, seconds * getAudioSourceSampleRate(source));
00054     }
00055
00056     void seekAudioSourcePercent(sAudioSource source, float percent) {
00057         seekAudioSourceSamples(source, percent * getAudioSourceSamples(source));
00058     }
00059
00060     float getAudioSourcePercent(sAudioSource source) {
00061         return (float)getAudioSourceSamples(source) / getAudioSourceSampleRate(source);
00062     }
00063
00064     float getAudioSourceSeconds(sAudioSource source) {
00065         return (float)getAudioSourceSamples(source) / getAudioSourceSampleRate(source);
00066     }
00067
00068     explicit AudioModule(const char* dylib) : Module(dylib, "aud") {
00069         init = (audio::Init)lib.getSymbol("init");
00070         loadAudioClip = (audio::LoadAudioClip)lib.getSymbol("loadAudioClip");
00071         createAudioSource = (audio::CreateAudioSource)lib.getSymbol("createAudioSource");
00072         playAudioSource = (audio::PlayAudioSource)lib.getSymbol("playAudioSource");
00073         stopAudioSource = (audio::StopAudioSource)lib.getSymbol("stopAudioSource");
00074         setAudioSourcePosition =
      (audio::SetAudioSourcePosition)lib.getSymbol("setAudioSourcePosition");
00075         setAudioSourceVelocity =
      (audio::SetAudioSourceVelocity)lib.getSymbol("setAudioSourceVelocity");
00076         setAudioSourcePitch = (audio::SetAudioSourcePitch)lib.getSymbol("setAudioSourcePitch");
00077         setAudioSourceGain = (audio::SetAudioSourceGain)lib.getSymbol("setAudioSourceGain");
00078         setAudioSourceLooping = (audio::SetAudioSourceLooping)lib.getSymbol("setAudioSourceLooping");
00079         seekAudioSourceSamples =
      (audio::SeekAudioSourceSamples)lib.getSymbol("seekAudioSourceSamples");
00080         getAudioSourceSamples = (audio::GetAudioSourceSamples)lib.getSymbol("getAudioSourceSamples");
00081         getAudioSourceSampleRate =
      (audio::GetAudioSourceSampleRate)lib.getSymbol("getAudioSourceSampleRate");
```

```
00082         destroyAudioClip = (audio::DestroyAudioClip)lib.getSymbol("destroyAudioClip");
00083         destroyAudioSource = (audio::DestroyAudioSource)lib.getSymbol("destroyAudioSource");
00084         destroy = (audio::Destroy)lib.getSymbol("destroy");
00085     }
00086 };
```

8.4 module.h

```
00001 #pragma once
00002 #include "../moduleLib.h"
00003
00004 #include "../win/module.h"
00005
00006 #include <stdio.h>
00007 #include <initializer_list>
00008 #include <cstdint>
00009 #include <cstdlib>
00010 #include <string>
00011
00012 typedef unsigned int sIndex;
00013
00014 struct GraphicsModule;
00015
00016 struct sVertexDefinition {
00017     int* elements;
00018     size_t count;
00019 };
00020
00021 size_t vertexDefinitionSize(sVertexDefinition* def) {
00022     size_t size = 0;
00023     for (size_t i = 0; i < def->count; i++) {
00024         size += def->elements[i] * sizeof(float);
00025     }
00026     return size;
00027 }
00028
00029 enum class sUniformType {
00030     FLOAT,
00031     INT,
00032     BOOL
00033 };
00034
00035 size_t uniformTypeSize(sUniformType type) {
00036     switch (type) {
00037         case sUniformType::FLOAT:
00038             return sizeof(float);
00039         case sUniformType::INT:
00040             return sizeof(int);
00041         case sUniformType::BOOL:
00042             return sizeof(bool);
00043     }
00044     return 0;
00045 }
00046
00047 enum sShaderType {
00048     VERTEX,
00049     FRAGMENT,
00050     GEOMETRY
00051 };
00052
00053 struct sUniformElement {
00054     sShaderType shaderType;
00055     const char* name;
00056     sUniformType type;
00057     size_t countx;
00058     size_t county=1;
00059
00060     sUniformElement(sShaderType shaderType, const char* name, sUniformType type, size_t countx, size_t
      county) : shaderType(shaderType), name(name), type(type), countx(countx), county(county) {}
00061     sUniformElement(sShaderType shaderType, const char* name, sUniformType type, size_t countx) :
      shaderType(shaderType), name(name), type(type), countx(countx) {}
00062 };
00063
00064 size_t uniformElementSize(sUniformElement element) {
00065     return element.countx * element.county * uniformTypeSize(element.type);
00066 }
00067
00068 struct sUniformDefinition {
00069     sUniformElement* elements;
00070     size_t count;
00071
00072     sUniformDefinition(std::initializer_list<sUniformElement> elements) {
00073         // we cant just cast begin to a pointer, because the array will be destroyed
```

```
00074         this->elements = (sUniformElement*)malloc(sizeof(sUniformElement) * elements.size());
00075         this->count = elements.size();
00076         size_t i = 0;
00077         for (auto it = elements.begin(); it != elements.end(); it++) {
00078             this->elements[i++] = *it;
00079         }
00080     }
00081
00082     size_t size() {
00083         size_t size = 0;
00084         for (size_t i = 0; i < count; i++) {
00085             size += elements[i].countx * elements[i].county * uniformTypeSize(elements[i].type);
00086         }
00087         return size;
00088     }
00089
00090     sUniformDefinition() : elements(nullptr), count(0) {}
00091     sUniformDefinition(sUniformElement* elements, size_t count) : elements(elements), count(count) {}
00092 };
00093
00094 sUniformDefinition getPartialf(sUniformDefinition def, sShaderType type) {
00095     size_t count = 0;
00096     for (size_t i = 0; i < def.count; i++) {
00097         if (def.elements[i].shaderType == type) {
00098             count++;
00099         }
00100     }
00101     sUniformElement* elements = (sUniformElement*)malloc(sizeof(sUniformElement) * count);
00102     if (elements == nullptr) {
00103         printf("ERROR: Malloc failed\n");
00104     }
00105     size_t j = 0;
00106     for (size_t i = 0; i < def.count; i++) {
00107         if (def.elements[i].shaderType == type) {
00108             elements[j++] = def.elements[i];
00109         }
00110     }
00111     return {elements, count};
00112 }
00113
00114 struct sUniforms {
00115     void* internal;
00116 };
00117
00118 struct sMesh {
00119     void* internal;
00120     GraphicsModule* creator;
00121 };
00122
00123 struct sShader {
00124     void* internal;
00125 };
00126
00127 struct GraphicsModule;
00128
00129 struct sShaderProgram {
00130     void* internal;
00131     GraphicsModule* creator;
00132     void* gfx_internal;
00133 };
00134
00135 struct sTextureDefinition {
00136     size_t width;
00137     size_t height;
00138     size_t channels;
00139     unsigned char* data;
00140 };
00141
00142 struct sTexture {
00143     void* internal;
00144 };
00145
00146 namespace graphics {
00147     typedef void (*SetClearColor)(float r, float g, float b, float a);
00148     typedef void (*Clear)();
00149     typedef void (*Init)(sWindow* win);
00150     typedef sMesh (*CreateMesh)(sShader vertexShader, void* vertices, size_t vertexSize, sIndex*
    indices, size_t indexSize);
00151     typedef void (*DrawMesh)(sMesh mesh);
00152     typedef void (*UseShaderProgram)(sShaderProgram shader);
00153     typedef sShader (*CreateShader)(const char* source, sShaderType type, sVertexDefinition* vertDef);
00154     typedef sShaderProgram (*CreateShaderProgram)(sShader* shaders, size_t count);
00155     typedef void (*Present)();
00156     typedef const char* (*GetShaderType)();
00157     typedef sUniforms (*CreateUniforms)(sShaderProgram program, sUniformDefinition def);
00158     typedef void (*SetUniforms)(sUniforms uniforms, void* data);
00159     typedef sTexture (*CreateTexture)(sTextureDefinition def);
```

```
00160     typedef void (*UseTexture)(sShaderProgram program, sTexture texture, const char* name);
00161     typedef void (*FreeTexture)(sTexture texture);
00162     typedef void (*FreeShader)(sShader shader);
00163     typedef void (*FreeShaderProgram)(sShaderProgram program);
00164     typedef void (*FreeMesh)(sMesh mesh);
00165     typedef void (*FreeUniforms)(sUniforms uniforms);
00166     typedef void (*Destroy)();
00167     typedef void (*SetScissor)(int x, int y, int width, int height);
00168     typedef void (*EnableScissor)();
00169     typedef void (*DisableScissor)();
00170 }
00171
00172 struct GraphicsModule : Module {
00173     graphics::SetClearColor setClearColor;
00174     graphics::Clear clear;
00175     graphics::Init internal_init;
00176     graphics::CreateMesh internal_createMesh;
00177     graphics::DrawMesh drawMesh;
00178     graphics::UseShaderProgram useShaderProgram;
00179     graphics::CreateShaderProgram internal_createShaderProgram;
00180     graphics::CreateShader internal_createShader;
00181     graphics::Present present;
00182     graphics::GetShaderType getShaderType;
00183     graphics::CreateUniforms createUniforms;
00184     graphics::SetUniforms setUniforms;
00185     graphics::CreateTexture createTexture;
00186     graphics::UseTexture useTexture;
00187     graphics::FreeTexture freeTexture;
00188     graphics::FreeShader freeShader;
00189     graphics::FreeShaderProgram freeShaderProgram;
00190     graphics::FreeMesh freeMesh;
00191     graphics::FreeUniforms freeUniforms;
00192     graphics::Destroy destroy;
00193     graphics::SetScissor setScissor;
00194     graphics::EnableScissor enableScissor;
00195     graphics::DisableScissor disableScissor;
00196
00197     sWindow* win;
00198
00199     void init(sWindow* win) {
00200         this->win = win;
00201         internal_init(win);
00202     }
00203
00204     sMesh createMesh(sShader vertexShader, void* vertices, size_t vertexCount, sIndex* indices, size_t
      indexCount) {
00205         sMesh mesh = internal_createMesh(vertexShader, vertices, vertexCount, indices, indexCount);
00206         mesh.creator = this;
00207         return mesh;
00208     }
00209
00210     sShaderProgram createShaderProgram(sShader* shaders, size_t count) {
00211         sShaderProgram program = internal_createShaderProgram(shaders, count);
00212         program.creator = this;
00213         return program;
00214     }
00215
00216     sShaderProgram createShaderProgram(std::initializer_list<sShader> shaders) {
00217         return createShaderProgram((sShader*)shaders.begin(), shaders.size());
00218     }
00219
00220     sVertexDefinition* createVertexDefinition(int* elements, size_t count) {
00221         sVertexDefinition* def = (sVertexDefinition*)malloc(sizeof(sVertexDefinition));
00222         // def->elements = elements;
00223         def->elements = (int*)malloc(sizeof(int) * count);
00224         if (def->elements == nullptr) {
00225             printf("ERROR: Malloc failed\n");
00226         }
00227         for (size_t i = 0; i < count; i++) {
00228             def->elements[i] = elements[i];
00229         }
00230         def->count = count;
00231         return def;
00232     }
00233
00234     sVertexDefinition* createVertexDefinition(std::initializer_list<int> elements) {
00235         return createVertexDefinition((int*)elements.begin(), elements.size());
00236     }
00237
00238     void freeVertexDefinition(sVertexDefinition* def) {
00239         // free(def->elements);
00240         free(def);
00241     }
00242
00243     sShader createShader(const char* source, sShaderType type, sVertexDefinition* vertDef) {
00244         sShader shader = internal_createShader(source, type, vertDef);
00245         return shader;
```

```
00246     }
00247
00248     sShader createShader(const char* source, sShaderType type) {
00249         if (type == sShaderType::VERTEX) {
00250             printf("ERROR: Vertex shader must have a vertex definition\n");
00251         }
00252         return createShader(source, type, nullptr);
00253     }
00254
00255     sShader loadShader(const char* path, sShaderType type, sVertexDefinition* vertDef) {
00256         std::string source;
00257         if (!readFile(path, source)) {
00258             printf("Error reading file\n");
00259             return {nullptr};
00260         }
00261         return createShader(source.c_str(), type, vertDef);
00262     }
00263
00264     sShader loadShader(const char* path, sShaderType type) {
00265         return loadShader(path, type, nullptr);
00266     }
00267
00268     explicit GraphicsModule(const char* dynlib) : Module(dynlib, "gfx") {
00269         setClearColor = (graphics::SetClearColor)lib.getSymbol("setClearColor");
00270         clear = (graphics::Clear)lib.getSymbol("clear");
00271         internal_init = (graphics::Init)lib.getSymbol("init");
00272         internal_createMesh = (graphics::CreateMesh)lib.getSymbol("createMesh");
00273         drawMesh = (graphics::DrawMesh)lib.getSymbol("drawMesh");
00274         useShaderProgram = (graphics::UseShaderProgram)lib.getSymbol("useShaderProgram");
00275         internal_createShaderProgram =
      (graphics::CreateShaderProgram)lib.getSymbol("createShaderProgram");
00276         internal_createShader = (graphics::CreateShader)lib.getSymbol("createShader");
00277         present = (graphics::Present)lib.getSymbol("present");
00278         getShaderType = (graphics::GetShaderType)lib.getSymbol("getShaderType");
00279         createUniforms = (graphics::CreateUniforms)lib.getSymbol("createUniforms");
00280         setUniforms = (graphics::SetUniforms)lib.getSymbol("setUniforms");
00281         createTexture = (graphics::CreateTexture)lib.getSymbol("createTexture");
00282         useTexture = (graphics::UseTexture)lib.getSymbol("useTexture");
00283         freeTexture = (graphics::FreeTexture)lib.getSymbol("freeTexture");
00284         freeShader = (graphics::FreeShader)lib.getSymbol("freeShader");
00285         freeShaderProgram = (graphics::FreeShaderProgram)lib.getSymbol("freeShaderProgram");
00286         freeMesh = (graphics::FreeMesh)lib.getSymbol("freeMesh");
00287         freeUniforms = (graphics::FreeUniforms)lib.getSymbol("freeUniforms");
00288         destroy = (graphics::Destroy)lib.getSymbol("destroy");
00289         setScissor = (graphics::SetScissor)lib.getSymbol("setScissor");
00290         enableScissor = (graphics::EnableScissor)lib.getSymbol("enableScissor");
00291         disableScissor = (graphics::DisableScissor)lib.getSymbol("disableScissor");
00292     }
00293 };
```

8.5 module.h

```
00001 #pragma once
00002
00003 #ifdef IUI_IMPLEMENTATION
00004 #define CLAY_IMPLEMENTATION
00005 #endif
00006 #include "clay.h"
00007
00008 #include "../gfx/module.h"
00009 #include "../text/module.h"
00010 #include "../shdr/module.h"
00011 #include "../asset.h"
00012
00013 #define CLAY_COLOR_TO_VEC4(color) {(color).r / 255.0f, (color).g / 255.0f, (color).b / 255.0f,
      (color).a / 255.0f}
00014
00015 struct sIUIGlobalState {
00016     WindowModule* winm;
00017     GraphicsModule* gfxm;
00018     TextModule* textm;
00019     ShaderModule* shdr;
00020
00021     sShaderProgram rect_shader;
00022     sVertexDefinition* rect_vert_def;
00023     sMesh rect_mesh;
00024     sUniforms rect_uniforms;
00025
00026     // rounded rects use the same mesh and vertex defs, but a different shader
00027     sShaderProgram rounded_rect_shader;
00028     sUniforms rounded_rect_uniforms;
00029
00030     sWindow* win;
```

```
00031
00032     sFont* fonts=nullptr;
00033 };
00034
00035 sIUIGlobalState __globalIUIState;
00036
00037 struct sInternalRectVertex {
00038     vec2 pos;
00039 };
00040 // the rectangle will actually be stretched using a model matrix, so we can just put a basic rectangle
      here
00041 sInternalRectVertex __rect_vertices[] = {
00042     {{0.0f, 0.0f}},
00043     {{1.0f, 0.0f}},
00044     {{1.0f, 1.0f}},
00045     {{0.0f, 1.0f}}
00046 };
00047
00048 sIndex __rect_indices[] = {
00049     0, 1, 2,
00050     2, 3, 0
00051 };
00052 struct sInternalRectUniforms {
00053     vec4 color;
00054
00055     mat4 proj;
00056     mat4 view;
00057     mat4 model;
00058     float z;
00059 };
00060
00061 struct sInternalRoundedRectUniforms {
00062     vec4 color;
00063     vec2 topleft;
00064     vec2 widheight;
00065     float radius;
00066
00067     mat4 proj;
00068     mat4 view;
00069     mat4 model;
00070     float z;
00071 };
00072
00073
00074 void clayerr(Clay_ErrorData errorData) {
00075     printf("UI Error: %s\n", errorData.errorText.chars);
00076 }
00077
00078 Clay_Dimensions Clay_Spectral_MeasureText(Clay_StringSlice text, Clay_TextElementConfig *config,
      uintptr_t userData) {
00079     if (!__globalIUIState.fonts) {
00080         return {};
00081     }
00082     Clay_Dimensions result = {};
00083     sFont font = __globalIUIState.fonts[0];
00084     vec2 size = __globalIUIState.textm->measureText(font, text.chars);
00085     int fs = config->fontSize;
00086     float scaleFactor = (float)fs / (float)(2*font.size);
00087     result.width = size.x * scaleFactor;
00088     result.height = size.y * scaleFactor;
00089     return result;
00090 }
00091
00092 void Clay_Spectral_Init(WindowModule* winm, GraphicsModule* gfxm, TextModule* textm, ShaderModule*
      shdr, sWindow* win, sFont* fonts, AssetLoader* assetm) {
00093     __globalIUIState.fonts = fonts;
00094     __globalIUIState.winm = winm;
00095     __globalIUIState.gfxm = gfxm;
00096     __globalIUIState.textm = textm;
00097     __globalIUIState.shdr = shdr;
00098     __globalIUIState.win = win;
00099     uint64_t totalMemorySize = Clay_MinMemorySize();
00100     Clay_Arena clayMemory = (Clay_Arena) {
00101         .capacity = totalMemorySize,
00102         .memory = (char*)malloc(totalMemorySize)
00103     };
00104     Clay_Initialize(clayMemory, (Clay_Dimensions){win->width, win->height},
      (Clay_ErrorHandler)clayerr);
00105     Clay_SetMeasureTextFunction(Clay_Spectral_MeasureText, 0);
00106
00107     sVertexDefinition* rect_vert_def = gfxm->createVertexDefinition({2});
00108     if (vertexDefinitionSize(rect_vert_def) != sizeof(sInternalRectVertex)) {
00109         printf("ERROR: Vertex definition size mismatch\n");
00110         return;
00111     }
00112     __globalIUIState.rect_vert_def = rect_vert_def;
00113     // sShader rect_vert_shader = shdr->compile(gfxm, "spsl/iui/rect.spslv", sShaderType::VERTEX,
```

```
    rect_vert_def);
00114     // sShader rect_frag_shader = shdr->compile(gfxm, "spsl/iui/rect.spslf", sShaderType::FRAGMENT);
00115     TextAssetBuffer rect_vert_abuf = assetm->loadTextAsset("spsl/iui/rect.spslv");
00116     TextAssetBuffer rect_frag_abuf = assetm->loadTextAsset("spsl/iui/rect.spslf");
00117     sShader rect_vert_shader = shdr->createShader(gfxm, (const char*)rect_vert_abuf.data,
    rect_vert_abuf.len, sShaderType::VERTEX, rect_vert_def);
00118     sShader rect_frag_shader = shdr->createShader(gfxm, (const char*)rect_frag_abuf.data,
    rect_frag_abuf.len, sShaderType::FRAGMENT);
00119     sShaderProgram rect_shader = gfxm->createShaderProgram({rect_vert_shader, rect_frag_shader});
00120     __globalIUIState.rect_shader = rect_shader;
00121     __globalIUIState.rect_mesh = gfxm->createMesh(rect_vert_shader, __rect_vertices,
    sizeof(__rect_vertices), __rect_indices, sizeof(__rect_indices));
00122
00123     sUniformDefinition rect_uniform_def = {
00124         {sShaderType::FRAGMENT, "uColor", sUniformType::FLOAT, 4},
00125         {sShaderType::VERTEX, "uProj", sUniformType::FLOAT, 4, 4},
00126         {sShaderType::VERTEX, "uView", sUniformType::FLOAT, 4, 4},
00127         {sShaderType::VERTEX, "uModel", sUniformType::FLOAT, 4, 4},
00128         {sShaderType::VERTEX, "uZ", sUniformType::FLOAT, 1}
00129     };
00130     if (rect_uniform_def.size() != sizeof(sInternalRectUniforms)) {
00131         printf("ERROR: Uniform definition size mismatch\n");
00132         return;
00133     }
00134     __globalIUIState.rect_uniforms = gfxm->createUniforms(rect_shader, rect_uniform_def);
00135
00136     // sShader rounded_rect_vert_shader = shdr->compile(gfxm, "spsl/iui/rounded_rect.spslv",
    sShaderType::VERTEX, rect_vert_def);
00137     // sShader rounded_rect_frag_shader = shdr->compile(gfxm, "spsl/iui/rounded_rect.spslf",
    sShaderType::FRAGMENT);
00138     TextAssetBuffer rounded_rect_vert_abuf = assetm->loadTextAsset("spsl/iui/rounded_rect.spslv");
00139     TextAssetBuffer rounded_rect_frag_abuf = assetm->loadTextAsset("spsl/iui/rounded_rect.spslf");
00140     sShader rounded_rect_vert_shader = shdr->createShader(gfxm, (const
    char*)rounded_rect_vert_abuf.data, rounded_rect_vert_abuf.len, sShaderType::VERTEX, rect_vert_def);
00141     sShader rounded_rect_frag_shader = shdr->createShader(gfxm, (const
    char*)rounded_rect_frag_abuf.data, rounded_rect_frag_abuf.len, sShaderType::FRAGMENT);
00142     sShaderProgram rounded_rect_shader = gfxm->createShaderProgram({rounded_rect_vert_shader,
    rounded_rect_frag_shader});
00143     __globalIUIState.rounded_rect_shader = rounded_rect_shader;
00144
00145     sUniformDefinition rounded_rect_uniform_def = {
00146         {sShaderType::FRAGMENT, "uColor", sUniformType::FLOAT, 4},
00147         {sShaderType::FRAGMENT, "uTopLeft", sUniformType::FLOAT, 2},
00148         {sShaderType::FRAGMENT, "uWidthHeight", sUniformType::FLOAT, 2},
00149         {sShaderType::FRAGMENT, "uRadius", sUniformType::FLOAT, 1},
00150         {sShaderType::VERTEX, "uProj", sUniformType::FLOAT, 4, 4},
00151         {sShaderType::VERTEX, "uView", sUniformType::FLOAT, 4, 4},
00152         {sShaderType::VERTEX, "uModel", sUniformType::FLOAT, 4, 4},
00153         {sShaderType::VERTEX, "uZ", sUniformType::FLOAT, 1}
00154     };
00155     if (rounded_rect_uniform_def.size() != sizeof(sInternalRoundedRectUniforms)) {
00156         printf("ERROR: Uniform definition size mismatch\n");
00157         return;
00158     }
00159
00160     __globalIUIState.rounded_rect_uniforms = gfxm->createUniforms(rounded_rect_shader,
    rounded_rect_uniform_def);
00161 }
00162
00163 // custom clay implementation using our graphics library
00164 void Clay_Spectral_Render(sWindow* win, Clay_RenderCommandArray renderCommands, mat4 proj, mat4 view)
    {
00165     // set clay viewport
00166     Clay_SetLayoutDimensions(Clay_Dimensions{(float)win->width, (float)win->height});
00167     float mousex, mousey;
00168     __globalIUIState.winm->getMousePosition(*win, &mousex, &mousey);
00169     Clay_Vector2 mousePos = {mousex, mousey};
00170     bool mouse = __globalIUIState.winm->isMouseButtonPressed(*win, 0);
00171     Clay_SetPointerState(mousePos, mouse);
00172
00173
00174     float z = -1.0f+0.01f;
00175     for (uint32_t i = 0; i < renderCommands.length; i++) {
00176         Clay_RenderCommand* renderCommand = Clay_RenderCommandArray_Get(&renderCommands, i);
00177         Clay_BoundingBox boundingBox = renderCommand->boundingBox;
00178         boundingBox.y = win->height - boundingBox.y - boundingBox.height;
00179         switch (renderCommand->commandType) {
00180             case CLAY_RENDER_COMMAND_TYPE_RECTANGLE: {
00181                 Clay_RectangleElementConfig *config = renderCommand->config.rectangleElementConfig;
00182                 Clay_Color color = config->color;
00183
00184                 if (config->cornerRadius.topLeft == 0) {
00185                     sInternalRectUniforms uniforms = {};
00186                     uniforms.color = {color.r / 255.0f, color.g / 255.0f, color.b / 255.0f, color.a /
    255.0f};
00187                     uniforms.proj = proj;
00188                     uniforms.view = view;
```

```
00189                     uniforms.model = scale({boundingBox.width, boundingBox.height, 1.0f}) *
      translate({boundingBox.x, boundingBox.y, 0.0f});
00190                     uniforms.z = z;
00191                     z += 0.01f;
00192 
00193                     __globalUIState.gfxm->useShaderProgram(__globalUIState.rect_shader);
00194                     __globalUIState.gfxm->setUniforms(__globalUIState.rect_uniforms, &uniforms);
00195                     __globalUIState.gfxm->drawMesh(__globalUIState.rect_mesh);
00196                 } else {
00197                     sInternalRoundedRectUniforms uniforms = {};
00198                     uniforms.color = {color.r / 255.0f, color.g / 255.0f, color.b / 255.0f, color.a /
      255.0f};
00199                     uniforms.proj = proj;
00200                     uniforms.view = view;
00201                     uniforms.model = scale({boundingBox.width, boundingBox.height, 1.0f}) *
      translate({boundingBox.x, boundingBox.y, 0.0f});
00202                     uniforms.z = z;
00203                     uniforms.topleft = {boundingBox.x, renderCommand->boundingBox.y};
00204                     uniforms.widheight = {boundingBox.width, boundingBox.height};
00205                     uniforms.radius = config->cornerRadius.topLeft;
00206                     z += 0.01f;
00207 
00208                     __globalUIState.gfxm->useShaderProgram(__globalUIState.rounded_rect_shader);
00209                     __globalUIState.gfxm->setUniforms(__globalUIState.rounded_rect_uniforms,
      &uniforms);
00210                     __globalUIState.gfxm->drawMesh(__globalUIState.rect_mesh);
00211 
00212                 }
00213             } break;
00214             case CLAY_RENDER_COMMAND_TYPE_TEXT: {
00215                 Clay_TextElementConfig *config = renderCommand->config.textElementConfig;
00216                 Clay_StringSlice text = renderCommand->text;
00217                 int fs = config->fontSize;
00218                 char *cloned = (char*)malloc(text.length + 1);
00219                 memcpy(cloned, text.chars, text.length);
00220                 cloned[text.length] = '\0';
00221                 sFont font = __globalUIState.fonts[0];
00222                 float scaleFactor = (float)fs / (float)(2*font.size);
00223                 sText textel = __globalUIState.textm->createText(font, cloned);
00224                 __globalUIState.textm->setTextProj(textel, proj);
00225                 __globalUIState.textm->setTextView(textel, view);
00226                 __globalUIState.textm->setTextModel(textel, scale({scaleFactor, scaleFactor, 1.0f}) *
      translate({boundingBox.x, boundingBox.y, 0.0f}));
00227                 __globalUIState.textm->setTextColor(textel, vec3{config->textColor.r / 255.0f,
      config->textColor.g / 255.0f, config->textColor.b / 255.0f});
00228                 __globalUIState.textm->setTextZ(textel, z);
00229                 z += 0.01f;
00230                 __globalUIState.textm->drawText(textel);
00231                 __globalUIState.textm->freeText(textel);
00232             } break;
00233             case CLAY_RENDER_COMMAND_TYPE_BORDER: {
00234                 Clay_BorderElementConfig *config = renderCommand->config.borderElementConfig;
00235 
00236                 sInternalRectUniforms uniforms = {};
00237                 uniforms.proj = proj;
00238                 uniforms.view = view;
00239                 uniforms.z = z;
00240 
00241                 if (config->bottom.width > 0) {
00242                     // draw a rect
00243                     uniforms.color = CLAY_COLOR_TO_VEC4(config->bottom.color);
00244                     uniforms.model = scale({boundingBox.width, (float)config->bottom.width, 1.0f}) *
      translate({boundingBox.x, boundingBox.y, 0.0f});
00245                     __globalUIState.gfxm->useShaderProgram(__globalUIState.rect_shader);
00246                     __globalUIState.gfxm->setUniforms(__globalUIState.rect_uniforms, &uniforms);
00247                     __globalUIState.gfxm->drawMesh(__globalUIState.rect_mesh);
00248                 }
00249                 if (config->left.width > 0) {
00250                     // draw a rect
00251                     uniforms.color = CLAY_COLOR_TO_VEC4(config->left.color);
00252                     uniforms.model = scale({(float)config->left.width, boundingBox.height, 1.0f}) *
      translate({boundingBox.x, boundingBox.y, 0.0f});
00253                     __globalUIState.gfxm->useShaderProgram(__globalUIState.rect_shader);
00254                     __globalUIState.gfxm->setUniforms(__globalUIState.rect_uniforms, &uniforms);
00255                     __globalUIState.gfxm->drawMesh(__globalUIState.rect_mesh);
00256                 }
00257                 if (config->right.width > 0) {
00258                     // draw a rect
00259                     uniforms.color = CLAY_COLOR_TO_VEC4(config->right.color);
00260                     uniforms.model = scale({(float)config->right.width, boundingBox.height, 1.0f}) *
      translate({boundingBox.x + boundingBox.width - config->right.width, boundingBox.y, 0.0f});
00261                     __globalUIState.gfxm->useShaderProgram(__globalUIState.rect_shader);
00262                     __globalUIState.gfxm->setUniforms(__globalUIState.rect_uniforms, &uniforms);
00263                     __globalUIState.gfxm->drawMesh(__globalUIState.rect_mesh);
00264                 }
00265                 if (config->top.width > 0) {
00266                     // draw a rect
```

```
00267                     uniforms.color = CLAY_COLOR_TO_VEC4(config->top.color);
00268                     uniforms.model = scale({boundingBox.width, (float)config->top.width, 1.0f}) *
    translate({boundingBox.x, boundingBox.y + boundingBox.height - config->top.width, 0.0f});
00269                     __globalIUIState.gfxm->useShaderProgram(__globalIUIState.rect_shader);
00270                     __globalIUIState.gfxm->setUniforms(__globalIUIState.rect_uniforms, &uniforms);
00271                     __globalIUIState.gfxm->drawMesh(__globalIUIState.rect_mesh);
00272                 }
00273                 z += 0.01f;
00274             } break;
00275         }
00276     }
00277 }
```

8.6 module.h

```
00001 #pragma once
00002 #include <cmath>
00003
00004
00005 union vec3 {
00006     struct {
00007         float x, y, z;
00008     };
00009     struct {
00010         float r, g, b;
00011     };
00012     float v[3];
00013 };
00014 inline vec3 operator+(vec3 a, vec3 b) {
00015     return {a.x + b.x, a.y + b.y, a.z + b.z};
00016 }
00017 inline vec3 operator-(vec3 a, vec3 b) {
00018     return {a.x - b.x, a.y - b.y, a.z - b.z};
00019 }
00020 inline vec3 operator*(vec3 a, float b) {
00021     return {a.x * b, a.y * b, a.z * b};
00022 }
00023 inline vec3 operator/(vec3 a, float b) {
00024     return {a.x / b, a.y / b, a.z / b};
00025 }
00026 inline vec3 operator*(float a, vec3 b) {
00027     return {a * b.x, a * b.y, a * b.z};
00028 }
00029 inline vec3 operator/(float a, vec3 b) {
00030     return {a / b.x, a / b.y, a / b.z};
00031 }
00032 inline float dot(vec3 a, vec3 b) {
00033     return a.x * b.x + a.y * b.y + a.z * b.z;
00034 }
00035 inline vec3 cross(vec3 a, vec3 b) {
00036     return {a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x};
00037 }
00038 inline float length(vec3 a) {
00039     return sqrtf(dot(a, a));
00040 }
00041 inline vec3 normalize(vec3 a) {
00042     return a / length(a);
00043 }
00044 inline vec3 operator-(vec3 a) {
00045     return {-a.x, -a.y, -a.z};
00046 }
00047 inline vec3 lerp(vec3 a, vec3 b, float t) {
00048     return a + (b - a) * t;
00049 }
00050 inline bool operator==(vec3 a, vec3 b) {
00051     return a.x == b.x && a.y == b.y && a.z == b.z;
00052 }
00053 inline bool operator!=(vec3 a, vec3 b) {
00054     return a.x != b.x || a.y != b.y || a.z != b.z;
00055 }
00056
00057 union vec4 {
00058     struct {
00059         float x, y, z, w;
00060     };
00061     struct {
00062         float r, g, b, a;
00063     };
00064     float v[4];
00065 };
00066 inline vec4 operator+(vec4 a, vec4 b) {
00067     return {a.x + b.x, a.y + b.y, a.z + b.z, a.w + b.w};
00068 }
```

```
00069 inline vec4 operator-(vec4 a, vec4 b) {
00070     return {a.x - b.x, a.y - b.y, a.z - b.z, a.w - b.w};
00071 }
00072 inline vec4 operator*(vec4 a, float b) {
00073     return {a.x * b, a.y * b, a.z * b, a.w * b};
00074 }
00075 inline vec4 operator/(vec4 a, float b) {
00076     return {a.x / b, a.y / b, a.z / b, a.w / b};
00077 }
00078 inline vec4 operator*(float a, vec4 b) {
00079     return {a * b.x, a * b.y, a * b.z, a * b.w};
00080 }
00081 inline vec4 operator/(float a, vec4 b) {
00082     return {a / b.x, a / b.y, a / b.z, a / b.w};
00083 }
00084 inline float dot(vec4 a, vec4 b) {
00085     return a.x * b.x + a.y * b.y + a.z * b.z + a.w * b.w;
00086 }
00087 inline float length(vec4 a) {
00088     return sqrtf(dot(a, a));
00089 }
00090 inline vec4 normalize(vec4 a) {
00091     return a / length(a);
00092 }
00093
00094 union vec2 {
00095     struct {
00096         float x, y;
00097     };
00098     struct {
00099         float u, v;
00100     };
00101     float f[2];
00102
00103     vec2(float x, float y) : x(x), y(y) {}
00104     vec2() : x(0), y(0) {}
00105     vec2(int x, int y) : x((float)x), y((float)y) {}
00106     vec2(unsigned int x, unsigned int y) : x((float)x), y((float)y) {}
00107 };
00108 inline vec2 operator+(vec2 a, vec2 b) {
00109     return {a.x + b.x, a.y + b.y};
00110 }
00111 inline vec2 operator-(vec2 a, vec2 b) {
00112     return {a.x - b.x, a.y - b.y};
00113 }
00114 inline vec2 operator*(vec2 a, float b) {
00115     return {a.x * b, a.y * b};
00116 }
00117 inline vec2 operator/(vec2 a, float b) {
00118     return {a.x / b, a.y / b};
00119 }
00120 inline vec2 operator*(float a, vec2 b) {
00121     return {a * b.x, a * b.y};
00122 }
00123 inline vec2 operator/(float a, vec2 b) {
00124     return {a / b.x, a / b.y};
00125 }
00126 inline float dot(vec2 a, vec2 b) {
00127     return a.x * b.x + a.y * b.y;
00128 }
00129 inline float length(vec2 a) {
00130     return sqrtf(dot(a, a));
00131 }
00132 inline vec2 normalize(vec2 a) {
00133     return a / length(a);
00134 }
00135
00136
00137 union mat4 {
00138     float m[4][4];
00139     struct {
00140         vec4 x, y, z, w;
00141     };
00142 };
00143 inline mat4 operator*(mat4 a, mat4 b) {
00144     mat4 result = {};
00145     for (int i = 0; i < 4; i++) {
00146         for (int j = 0; j < 4; j++) {
00147             result.m[i][j] = a.m[i][0] * b.m[0][j] + a.m[i][1] * b.m[1][j] + a.m[i][2] * b.m[2][j] +
      a.m[i][3] * b.m[3][j];
00148         }
00149     }
00150     return result;
00151 }
00152 inline vec4 operator*(mat4 a, vec4 b) {
00153     vec4 result = {};
00154     for (int i = 0; i < 4; i++) {
```

```
00155         result.v[i] = a.m[i][0] * b.v[0] + a.m[i][1] * b.v[1] + a.m[i][2] * b.v[2] + a.m[i][3] *
      b.v[3];
00156     }
00157     return result;
00158 }
00159 inline mat4 identity() {
00160     mat4 result = {};
00161     for (int i = 0; i < 4; i++) {
00162         result.m[i][i] = 1.0f;
00163     }
00164     return result;
00165 }
00166 inline mat4 translate(vec3 v) {
00167     mat4 result = identity();
00168     result.w.x = v.x;
00169     result.w.y = v.y;
00170     result.w.z = v.z;
00171     return result;
00172 }
00173 inline mat4 scale(vec3 v) {
00174     mat4 result = {};
00175     result.x.x = v.x;
00176     result.y.y = v.y;
00177     result.z.z = v.z;
00178     result.w.w = 1.0f;
00179     return result;
00180 }
00181 inline mat4 rotate(float angle, vec3 axis) {
00182     axis = normalize(axis);
00183     float s = sinf(angle);
00184     float c = cosf(angle);
00185     float oc = 1.0f - c;
00186     mat4 result = {};
00187     result.x.x = oc * axis.x * axis.x + c;
00188     result.x.y = oc * axis.x * axis.y - axis.z * s;
00189     result.x.z = oc * axis.x * axis.z + axis.y * s;
00190     result.y.x = oc * axis.y * axis.x + axis.z * s;
00191     result.y.y = oc * axis.y * axis.y + c;
00192     result.y.z = oc * axis.y * axis.z - axis.x * s;
00193     result.z.x = oc * axis.z * axis.x - axis.y * s;
00194     result.z.y = oc * axis.z * axis.y + axis.x * s;
00195     result.z.z = oc * axis.z * axis.z + c;
00196     result.w.w = 1.0f;
00197     return result;
00198 }
00199
00200 inline mat4 rotate(vec3 angles) {
00201     mat4 result = rotate(angles.z, {0, 0, 1}) * rotate(angles.y, {0, 1, 0}) * rotate(angles.x, {1, 0,
      0});
00202     return result;
00203 }
00204
00205 inline mat4 perspective(float fov, float aspect, float nearp, float farp) {
00206     float f = 1.0f / tanf(fov * 0.5f * 3.14159f / 180.0f);
00207     mat4 result = {};
00208     result.x.x = f / aspect;
00209     result.y.y = f;
00210     result.z.z = (farp + nearp) / (nearp - farp);
00211     result.z.w = -1.0f;
00212     result.w.z = 2.0f * farp * nearp / (nearp - farp);
00213     return result;
00214 }
00215
00216 // this matrix only works when bl is the origin (for some reason)
00217 inline mat4 orthographic(float left, float right, float bottom, float top, float nearp, float farp) {
00218     mat4 result = {};
00219     result.x.x = 2.0f / (right - left);
00220     result.y.y = 2.0f / (top - bottom);
00221     result.z.z = -2.0f / (farp - nearp);
00222     result.w.x = -(right + left) / (right - left);
00223     result.w.y = -(top + bottom) / (top - bottom);
00224     result.w.z = -(farp + nearp) / (farp - nearp);
00225     result.w.w = 1.0f;
00226     return result;
00227 }
00228
00229 struct sCamera {
00230     vec3 pos = {0, 0, 0};
00231     vec3 up = {0, 1, 0};
00232     vec3 forward = {0, 0, -1};
00233     float yaw = 0.0f;
00234     float pitch = 0.0f;
00235
00236     vec3 right() {
00237         return normalize(cross(forward, up));
00238     }
00239     vec3 left() {
```

```
00240         return normalize(cross(up, forward));
00241     }
00242     vec3 back() {
00243         return normalize(-forward);
00244     }
00245     vec3 down() {
00246         return normalize(-up);
00247     }
00248 
00249     vec3 right(vec3 forward) {
00250         return normalize(cross(forward, up));
00251     }
00252     vec3 left(vec3 forward) {
00253         return normalize(cross(up, forward));
00254     }
00255     vec3 back(vec3 forward) {
00256         return normalize(-forward);
00257     }
00258     vec3 down(vec3 up) {
00259         return normalize(-up);
00260     }
00261 };
00262 
00263 inline mat4 view(sCamera camera) {
00264     vec3 z = normalize(-camera.forward);
00265     vec3 x = normalize(cross(camera.up, z));
00266     vec3 y = cross(z, x);
00267     mat4 result = {};
00268     result.x.x = x.x;
00269     result.x.y = y.x;
00270     result.x.z = z.x;
00271     result.y.x = x.y;
00272     result.y.y = y.y;
00273     result.y.z = z.y;
00274     result.z.x = x.z;
00275     result.z.y = y.z;
00276     result.z.z = z.z;
00277     result.w.x = -dot(x, camera.pos);
00278     result.w.y = -dot(y, camera.pos);
00279     result.w.z = -dot(z, camera.pos);
00280     result.w.w = 1.0f;
00281     return result;
00282 }
00283 
00284 inline void camYaw(sCamera *camera, float angle) {
00285     camera->yaw += angle;
00286     if (camera->yaw > 3.14159f) camera->yaw -= 2.0f * 3.14159f;
00287     if (camera->yaw < -3.14159f) camera->yaw += 2.0f * 3.14159f;
00288     camera->forward.x = cosf(camera->yaw) * cosf(camera->pitch);
00289     camera->forward.y = sinf(camera->pitch);
00290     camera->forward.z = sinf(camera->yaw) * cosf(camera->pitch);
00291     camera->forward = normalize(camera->forward);
00292 }
00293 
00294 inline void camPitch(sCamera *camera, float angle) {
00295     camera->pitch += angle;
00296     if (camera->pitch > 3.14159f / 2.0f) camera->pitch = 3.14159f / 2.0f;
00297     if (camera->pitch < -3.14159f / 2.0f) camera->pitch = -3.14159f / 2.0f;
00298     camera->forward.x = cosf(camera->yaw) * cosf(camera->pitch);
00299     camera->forward.y = sinf(camera->pitch);
00300     camera->forward.z = sinf(camera->yaw) * cosf(camera->pitch);
00301     camera->forward = normalize(camera->forward);
00302 }
00303 
00304 inline void camMove(sCamera *camera, vec3 dir, float speed) {
00305     camera->pos = camera->pos + dir * speed;
00306 }
00307 
00308 struct sModelTransform {
00309     vec3 pos = {0, 0, 0};
00310     vec3 sca = {1, 1, 1};
00311     vec3 rot = {0, 0, 0};
00312 
00313     vec3 lastPos = {0, 0, 0};
00314     vec3 lastSca = {1, 1, 1};
00315     vec3 lastRot = {0, 0, 0};
00316 
00317     mat4 internal_matrix = identity();
00318     mat4 matrix() {
00319         if (pos != lastPos || sca != lastSca || rot != lastRot) {
00320             internal_matrix = translate(pos) * rotate(rot) * scale(sca);
00321             lastPos = pos;
00322             lastSca = sca;
00323             lastRot = rot;
00324         }
00325         return internal_matrix;
00326     }
```

```
00327 };
00328
```

8.7 module.h

```
00001 #pragma once
00002
00003 // this file is the module.h for the scriptloading module
00004 // defines common types and functions for the scriptloading module
00005 // everything that is "language-specfic" is in the dll file, and this file is the interface to that
00006
00007
00008 #include "../moduleLib.h"
00009 #include <string.h>
00010 #include <stdlib.h>
00011
00012 typedef void (*ScriptInit)();
00013 typedef void (*ScriptUpdate)(float dt);
00014
00015 struct Script {
00016     void* internal;
00017     ScriptInit init;
00018     ScriptUpdate update;
00019 };
00020
00021 namespace scrload {
00022     typedef Script (*ScriptLoader)(const char *path, const char *scriptName);
00023     typedef void (*ScriptCompiler)(const char *scriptPath, const char *outputPath, const char
      *scriptName);
00024     typedef void (*MultiScriptCompiler)(const char **paths, size_t paths_num, const char *outputPath,
      const char *scriptName);
00025 }
00026 struct ScriptLoaderModule : public Module {
00027     scrload::ScriptLoader loadScript;
00028     scrload::ScriptCompiler compileScript;
00029     scrload::MultiScriptCompiler compileScripts;
00030     char* inputExtension;
00031     char* outputExtension;
00032
00033     explicit ScriptLoaderModule(const char* dynlib) : Module(dynlib, "scrld") {
00034         loadScript = (scrload::ScriptLoader)lib.getSymbol("loadScript");
00035         compileScript = (scrload::ScriptCompiler)lib.getSymbol("compileScript");
00036         compileScripts = (scrload::MultiScriptCompiler)lib.getSymbol("compileScripts");
00037         auto iext = (const char**)lib.getSymbol("inputExtension");
00038         auto oext = (const char**)lib.getSymbol("outputExtension");
00039         // copy strings because the dll will be unloaded
00040         inputExtension = (char*)malloc(strlen(*iext) + 1);
00041         outputExtension = (char*)malloc(strlen(*oext) + 1);
00042         strcpy((char*)inputExtension, *iext);
00043         strcpy((char*)outputExtension, *oext);
00044     }
00045 };
```

8.8 module.h

```
00001 #pragma once
00002 #include "../moduleLib.h"
00003
00004 #include "../gfx/module.h"
00005
00006 #include <stdio.h>
00007
00008 const char* combine_strs_with_delim(const char* a, const char* b, char delim) {
00009     size_t len = strlen(a) + strlen(b) + 2;
00010     char* out = (char*)malloc(len);
00011     snprintf(out, len, "%s%c%s", a, delim, b);
00012     return out;
00013 }
00014
00015 namespace shader {
00016      typedef sShader (*Compile)(GraphicsModule* gfxm, const char* path, sShaderType type,
      sVertexDefinition* vertDef);
00017      typedef sShader (*CreateShader)(GraphicsModule* gfxm, const char* data, size_t len, sShaderType
      type, sVertexDefinition* vertDef);
00018 }
00019
00020 struct ShaderModule : Module {
00021     shader::Compile internal_compile;
00022     shader::CreateShader internal_createShader;
```

```
00023
00024     sShader compile(GraphicsModule* gfxm, const char* path, sShaderType type, sVertexDefinition*
      vertDef=nullptr) {
00025         return internal_compile(gfxm, path, type, vertDef);
00026     }
00027
00028     sShader createShader(GraphicsModule* gfxm, const char* data, size_t len, sShaderType type,
      sVertexDefinition* vertDef=nullptr) {
00029         return internal_createShader(gfxm, data, len, type, vertDef);
00030     }
00031
00032     explicit ShaderModule(const char* dynlib, const char* dynp2) :
      Module(combine_strs_with_delim(dynlib, dynp2, '_'), "shdr") {
00033         internal_compile = (shader::Compile)lib.getSymbol("compile");
00034         internal_createShader = (shader::CreateShader)lib.getSymbol("createShader");
00035     }
00036 };
```

8.9 module.h

```
00001 #pragma once
00002
00003 // this file is the module.h for the scriptloading module
00004 // defines common types and functions for the scriptloading module
00005 // everything that is "language-specfic" is in the dll file, and this file is the interface to that
00006
00007
00008 #include "../moduleLib.h"
00009 #include "../gfx/module.h"
00010
00011
00012 namespace texload {
00013     typedef sTextureDefinition (*LoadTexture)(const char*);
00014     typedef void (*FreeTexture)(sTextureDefinition);
00015     typedef sTextureDefinition (*LoadTextureFromBuffer)(const uint8_t*, size_t);
00016 }
00017 struct TextureModule : public Module {
00018     texload::LoadTexture loadTexture;
00019     texload::FreeTexture freeTexture;
00020     texload::LoadTextureFromBuffer loadTextureFromBuffer;
00021
00022     TextureModule(const char* path) : Module(path, "tex") {
00023         loadTexture = (texload::LoadTexture)lib.getSymbol("loadTexture");
00024         freeTexture = (texload::FreeTexture)lib.getSymbol("freeTexture");
00025         loadTextureFromBuffer =
      (texload::LoadTextureFromBuffer)lib.getSymbol("loadTextureFromBuffer");
00026     }
00027 };
```

8.10 module.h

```
00001 #pragma once
00002 #include "../moduleLib.h"
00003
00004 #include "../math/module.h"
00005 #include "../gfx/module.h"
00006 #include "../shdr/module.h"
00007 #include "../asset.h"
00008
00009 struct sFont {
00010     void* internal;
00011     int size;
00012 };
00013
00014 struct sText {
00015     void* internal;
00016 };
00017
00018 namespace text {
00019     typedef void (*Init)(GraphicsModule* gfxm, ShaderModule* shdr, AssetLoader* assetm);
00020     typedef sFont (*LoadFont)(const char* path, int size, const char* vertpath, const char* fragpath);
00021     typedef sFont (*LoadFontAsset)(const char* path, int size, const char* vertpath, const char*
      fragpath);
00022     typedef sText (*CreateText)(sFont font, const char* text);
00023     typedef void (*DrawText)(sText text);
00024     typedef void (*FreeText)(sText text);
00025     typedef void (*FreeFont)(sFont font);
00026     typedef void (*SetTextColor)(sText text, vec3 color);
00027     typedef void (*SetTextModel)(sText text, mat4 model);
```

```
00028     typedef void (*SetTextView)(sText text, mat4 view);
00029     typedef void (*SetTextProj)(sText text, mat4 proj);
00030     typedef vec2 (*MeasureText)(sFont font, const char* text);
00031     typedef void (*SetTextZ)(sText text, float z);
00032 }
00033
00034 struct TextModule : public Module {
00035     text::Init init;
00036     text::LoadFont internal_loadFont;
00037     text::LoadFontAsset internal_loadFontAsset;
00038     text::CreateText createText;
00039     text::DrawText drawText;
00040     text::FreeText freeText;
00041     text::FreeFont freeFont;
00042     text::SetTextColor setTextColor;
00043     text::SetTextModel setTextModel;
00044     text::SetTextView setTextView;
00045     text::SetTextProj setTextProj;
00046     text::MeasureText measureText;
00047     text::SetTextZ setTextZ;
00048
00049     sFont loadFont(const char* path, int size, const char* vertpath, const char* fragpath) {
00050         sFont f = internal_loadFont(path, size, vertpath, fragpath);
00051         f.size = size;
00052         return f;
00053     }
00054
00055     sFont loadFontAsset(const char* path, int size, const char* vertpath, const char* fragpath) {
00056         sFont f = internal_loadFontAsset(path, size, vertpath, fragpath);
00057         f.size = size;
00058         return f;
00059     }
00060
00061     TextModule(const char* path) : Module(path, "text") {
00062         init = (text::Init)lib.getSymbol("init");
00063         internal_loadFont = (text::LoadFont)lib.getSymbol("loadFont");
00064         internal_loadFontAsset = (text::LoadFontAsset)lib.getSymbol("loadFontAsset");
00065         createText = (text::CreateText)lib.getSymbol("createText");
00066         drawText = (text::DrawText)lib.getSymbol("drawText");
00067         freeText = (text::FreeText)lib.getSymbol("freeText");
00068         freeFont = (text::FreeFont)lib.getSymbol("freeFont");
00069         setTextColor = (text::SetTextColor)lib.getSymbol("setTextColor");
00070         setTextModel = (text::SetTextModel)lib.getSymbol("setTextModel");
00071         setTextView = (text::SetTextView)lib.getSymbol("setTextView");
00072         setTextProj = (text::SetTextProj)lib.getSymbol("setTextProj");
00073         measureText = (text::MeasureText)lib.getSymbol("measureText");
00074         setTextZ = (text::SetTextZ)lib.getSymbol("setTextZ");
00075     }
00076 };
```

8.11 module.h

```
00001
00007 #pragma once
00008 #include "../moduleLib.h"
00009
00010 #include <stdio.h>
00011 #include <chrono>
00012
00013 struct WindowModule;
00014
00015 /*
00016  * @brief Window module interface.
00017  *
00018  * Window flags structure. This is used to set various properties of the window.
00019
00020  * @note The flags are not guaranteed to be supported on all platforms or windowing libraries.
00021  * @note The flags may be ignored or have no effect depending on the windowing library used.
00022 */
00023 struct sWindowFlags {
00028     bool vsync;
00032     bool resizable;
00033 };
00034
00043 struct sWindow {
00046     double dt;
00049     int width;
00052     int height;
00056     sWindowFlags flags;
00057
00058     void* internal;
00059     WindowModule* creator;
00060     double lastTime;
```

```
00061     std::chrono::high_resolution_clock::time_point startTime;
00062     bool did_resize;
00063 };
00064
00065 // Required for compatibility with different windowing libraries
00066 enum class Key {
00067     A=0, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,
00068     Num0, Num1, Num2, Num3, Num4, Num5, Num6, Num7, Num8, Num9,
00069     Escape, LControl, LShift, LAlt, LSystem, RControl, RShift, RAlt, RSystem,
00070     Menu, LBracket, RBracket, SemiColon, Comma, Period, Quote, Slash, BackSlash,
00071     Tilde, Equal, Dash, Space, Return, BackSpace, Tab, PageUp, PageDown, End, Home,
00072     Insert, Delete, Add, Subtract, Multiply, Divide, Left, Right, Up, Down, Numpad0,
00073     Numpad1, Numpad2, Numpad3, Numpad4, Numpad5, Numpad6, Numpad7, Numpad8,
00074     Numpad9, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11,F12, F13, F14, F15,
00075     Pause, KeyCount
00076 };
00085 enum class CursorMode {
00089     Normal = 0,
00093     Hidden,
00098     Disabled
00099 };
00100
00101 namespace window {
00102     typedef sWindow* (*WindowLoader)(const char *name, int width, int height, sWindowFlags flags);
00103     typedef void (*WindowDestructor)(sWindow* window);
00104     typedef void (*WindowUpdate)(sWindow* window);
00105     typedef void (*WindowSwapBuffers)(sWindow window);
00106     typedef bool (*WindowShouldClose)(sWindow window);
00107     typedef void (*WindowSetShouldClose)(sWindow window, bool value);
00108     typedef void* (*WindowGetHandle)(sWindow window);
00109     typedef bool (*WindowIsKeyPressed)(sWindow window, Key key);
00110     typedef bool (*WindowIsMouseButtonPressed)(sWindow window, int button);
00111     typedef void (*WindowGetMousePosition)(sWindow window, float* x, float* y);
00112     typedef void (*WindowSetMousePosition)(sWindow window, float x, float y);
00113     typedef void (*WindowSetCursorMode)(sWindow window, CursorMode mode);
00114     typedef void (*WindowSetWindowTitle)(sWindow window, const char* title);
00115     typedef void (*WindowSetResizable)(sWindow window, bool resizable);
00116 }
00117
00127 struct WindowModule : Module {
00128     window::WindowLoader internal_loadWindow;
00129     window::WindowUpdate internal_updateWindow;
00130
00134     window::WindowDestructor destroyWindow;
00135     window::WindowSwapBuffers swapBuffers;
00136     window::WindowShouldClose shouldClose;
00137     window::WindowSetShouldClose setShouldClose;
00138     window::WindowGetHandle getHandle;
00139     window::WindowIsKeyPressed isKeyPressed;
00140     window::WindowIsMouseButtonPressed isMouseButtonPressed;
00141     window::WindowGetMousePosition getMousePosition;
00142     window::WindowSetMousePosition setMousePosition;
00143     window::WindowSetCursorMode setCursorMode;
00144     window::WindowSetWindowTitle setWindowTitle;
00145
00146     sWindow* loadWindow(const char* name, int width, int height, sWindowFlags flags) {
00147         sWindow* w = internal_loadWindow(name, width, height, flags);
00148         w->creator = this;
00149         w->startTime = std::chrono::high_resolution_clock::now();
00150         w->flags = flags;
00151         w->width = width;
00152         w->height = height;
00153         w->did_resize = false;
00154         return w;
00155     }
00156
00157     sWindow* loadWindow(const char* name, int width, int height, bool vsync=true, bool
      resizable=false) {
00158         sWindowFlags flags = {vsync, false};
00159         return loadWindow(name, width, height, flags);
00160     }
00161
00162     void updateWindow(sWindow* window) {
00163         internal_updateWindow(window);
00164         double cur = getTime(*window);
00165         window->dt = cur - window->lastTime;
00166         window->lastTime = cur;
00167     }
00168
00169     double getTime(sWindow window) {
00170         auto now = std::chrono::high_resolution_clock::now();
00171         return std::chrono::duration<double>(now - window.startTime).count();
00172     }
00173
00174     explicit WindowModule(const char* dynlib) : Module(dynlib, "win") {
00175         internal_loadWindow = (window::WindowLoader)lib.getSymbol("loadWindow");
00176         destroyWindow = (window::WindowDestructor)lib.getSymbol("destroyWindow");
```

```
00177         internal_updateWindow = (window::WindowUpdate)lib.getSymbol("updateWindow");
00178         swapBuffers = (window::WindowSwapBuffers)lib.getSymbol("swapBuffers");
00179         shouldClose = (window::WindowShouldClose)lib.getSymbol("shouldClose");
00180         setShouldClose = (window::WindowSetShouldClose)lib.getSymbol("setShouldClose");
00181         getHandle = (window::WindowGetHandle)lib.getSymbol("getHandle");
00182         isKeyPressed = (window::WindowIsKeyPressed)lib.getSymbol("isKeyPressed");
00183         isMouseButtonPressed =
      (window::WindowIsMouseButtonPressed)lib.getSymbol("isMouseButtonPressed");
00184         getMousePosition = (window::WindowGetMousePosition)lib.getSymbol("getMousePosition");
00185         setMousePosition = (window::WindowSetMousePosition)lib.getSymbol("setMousePosition");
00186         setCursorMode = (window::WindowSetCursorMode)lib.getSymbol("setCursorMode");
00187         setWindowTitle = (window::WindowSetWindowTitle)lib.getSymbol("setWindowTitle");
00188     }
00189
00190 };
00191
00192 // @}
```

8.12 module.h

```
00001 /*
00002 This module is one of the most complicated, even compared to gfx
00003 This module handles the world, level data, and pretty much everything related to the game
00004 It is stored in a custom format, as well as a custom format for things contained within it
00005 This format actually is a zip file, containing even more sub-formats
00006 The editor is completely separate from the engine, and is used to edit, create, convert, and view
      these files
00007 The editor (as of now) doesn't even exist, but it is planned to be made in the future
00008
00009 Format specification:
00010 filename (description) [format] // if the format is [zip], [png], [json], or any other format, it is
      identical to that format, just renamed
00011 Format:
00012 - gmae.spgam (game data) [zip] // this file might eng up being split up into multiple files if it gets
      too big (idk how i'm gonna do that yet)
00013     - world.spwld (world data) [json]
00014     - models/
00015         - MODEL.spmdl (model data) [similar to obj]
00016     - textures/
00017         - TEXTURE.(png/jpg/etc) (texture data) [png/jpg/etc, but png is is recommended]
00018     - materials/
00019         - MATERIAL.spmat (material data) [json]
00020     - levels/
00021         - LEVEL.splvl (level data) [json]
00022     - scripts/
00023         - SCRIPT.spscr (script meta) [json]
00024         - SCRIPT.(cpp/py/etc) (script code) [whatever language the script is in, many languages are
      supported (although cpp is the most common, and most performant)]
00025     - audio/
00026         - AUDIO.(wav/mp3/etc) (audio data) [wav/mp3/etc but wav is recommended]
00027     - etc/
00028         - ETC.spetc (etc meta) [json]
00029         - ETC.(etc) (etc data) [etc, any custom data can be stored here. WARNING: This is STATIC data,
      and cannot be changed during runtime, this is for things like configuration files that don't need to
      be changed once the game is compiled]
00030
00031 File structure of compiled game:
00032 - game.spgam
00033 - Spectral.PLATFORM.exe
00034 - modules/
00035     - Contains all the modules needed to run the game
00036 - data/
00037     - Contains all NON-STATIC data, such as logs, save files (only if the game enables portable
      saving, otherwise it is stored in appdata), etc
00038     - This folder can be used to store configuration files that the user can change, and it will
      probably be used for settings (if they aren't stored in the save)
00039     - This folder can also be used for modding purposes if the game wants to support mods (DISCLAIMER:
      Mods are NOT supported by the engine, and would have to be implemented by the developer)
00040 - lib/
00041     - Contains all the libraries needed to run the game
00042 - logs/
00043     - Log files are stored here, and are used for debugging purposes
00044 - engine/
00045     - Contains engine settings, and is used to store engine data that the user is allowed to change
00046     - This includes settings like resolution, graphics settings, etc
00047     - This folder will also contain configs that allow the user to modify which modules are loaded
      (for example, choosing OpenGL instead of DirectX)
00048 - cache/
00049     - Contains cached data (maybe, in the future)
00050
00051 most structs in the engine are prefixed with "s" (for Spectral), but this module is prefixed with "sw"
      (for Spectral World)
00052 */
```

```
00053
00054 /*
00055 The above comment is outdated, as I have decided to make some changes as to how the game will be
      stored
00056 The game (on the developer's side) will be in a folder, with basically the same structure as the above
      specs
00057 But when the game is compiled, much of this data will be converted into C++ code and compiled into
      dlls
00058 Each level/scene will be a separate dll, and the user scripts will all be compiled into a single dll
00059 The main change, is that a level, asset, or script can be marked as a "moddable" asset, and it will
      not be compiled into anything, it will be stored in the final game folder under a user/ folder
00060 The engine will also make a mods/ folder, where the user can place mods which will contain
      replacements for the assets in the moddable folder. The engine will hash every dll (modules, user
      scripts, and levels) as well as the moddable scripts. and will prevent the game from running if any of
      these files are modified
00061 When the engine loads (on computers only, consoles do not support mods and selection of api), it will
      open up a popup letting the user select which modules to use, and mods to enable (if any)
00062 This will allow the user to select things like OpenGL or DirectX. and will allow them to add custom
      modules. For example: If the user wanted to use vulkan, and someone made a vulkan module, they could
      add the vulkan module to the modules folder, and select it in the engine settings (this will give a
      warning that the module is not official, and could be dangerous)
00063 The engine will never select a non-official module by default, and will always open the config popup
      if the user doesn't click "don't show this again"
00064 If the user selects a non-official module, and then "don't show this again", it will still be shown to
      give the user a warning that they are using a non-official module
00065 This popup menu is completely seperate from the rest of the engine, and will be run before anything
      else in the engine is loaded.
00066 So the engine can recognize official modules, when the engine is compiled, it will hash all the
      official modules, and the hash will be stored inside of the engine executable. When the engine is run,
      it will verify that the official modules have not been modified, and if they have, it will act as if
      that module is non-official
00067 The engine executable and official modules will all be the same for every game, since the game is
      actually just a few dlls.
00068
00069 All assets (non-moddable) will be compiled into an asset dll, which will store them as C++ code (just
      an array of bytes), and will be loaded into the game when the game starts
00070 The assets that are moddable will be in the user/ folder.
00071
00072 This part of the engine (runtime) has no idea how to deal with the json format of the game, only the
      compiled format
00073
00074 The engine is split into 3 parts:
00075 - Runtime: The part of the engine that runs the game. This contains all the modules, and the main
      executable. This part is compiled into the engine exe, and the module dlls
00076 - Editor: This is the last part that I will make, and it will be used to create and edit the game file
      (in the dev format json). This part will be a completely separate program, but will borrow some of the
      modules from the runtime
00077 - Compiler: This part is used to compile the game into the runtime format. This part will integrate
      with CMake, and will be used to make a game project into a game dll. This part will be put in the
      game/ directory of the output from the first part.
00078
00079
00080 So a final compiled game will look like this:
00081 - game/
00082   - assets.dll
00083   - levels/
00084     - level1.dll
00085     - level2.dll
00086   - scripts.dll
00087 - modules/
00088   - all the modules needed to run the game
00089 - user/
00090   - assets/
00091     - moddable assets
00092   - levels/
00093     - moddable levels
00094   - scripts/
00095     - moddable scripts
00096 - mods/
00097     - mod1/
00098         - assets/
00099         - moddable assets
00100         - levels/
00101         - moddable levels
00102         - scripts/
00103         - moddable scripts
00104     - mod2/
00105         - assets/
00106         - moddable assets
00107         - levels/
00108         - moddable levels
00109         - scripts/
00110         - moddable scripts
00111 - Spectral.PLATFORM.exe
00112 - data/
00113     - logs/
00114     - save/
```

```
00115     - settings/
00116 - lib/
00117     - all the libraries needed to run the game
00118 - engine/
00119     - engine settings
00120 - cache/
00121     - cached data
00122 */
00123
00124 #pragma once
00125
00126 #include "../moduleLib.h"
00127
00128 #include <string>
00129 #include <vector>
00130 #include <cstdint>
00131
00132 #include "../math/module.h"
00133
00134 struct swWorld { // spwld [json]
00135     std::string name;
00136     std::string author;
00137     std::string description;
00138 };
00139
00140 struct swModel { // spmdl [similar to obj]
00141     std::vector<float> vertices;
00142     std::vector<unsigned int> indices;
00143 };
00144
00145 struct swMaterial { // spmat [json]
00146     std::string shader;
00147     std::vector<std::string> samplers;
00148     std::vector<std::string> vertexUniforms;
00149     std::vector<std::string> fragmentUniforms;
00150 };
00151
00152 struct swLevelObject {
00153     sModelTransform transform;
00154     void* ecsObject; // TODO: Implement ECS
00155 };
00156
00157 struct swLevel { // splvl [json]
00158     std::vector<swLevelObject> objects;
00159 };
00160
00161 struct swScript { // spscr [json]
00162     std::string ext; // the extension of the script file
00163     std::string mod; // the name of the scriptloader module to load this language
00164 };
00165
00166 struct swEtc { // spetc [json]
00167     enum swEtcType {
00168         PRE_LOAD, // this data is required before the game's initial load/init (like settings and
      other data that is required to start the game)
00169         PRE_LOOP, // this data is required before the game's main loop starts (like level data and
      other data that is required to run the game)
00170         POST_LOOP, // this data can be lazy-loaded after the game's main loop starts (like extra data
      or dlc content that can be loaded after the game starts)
00171         POST_GAME // this data is loaded right before the game's cleanup (not used very often, but
      could be used for save templates, but remember, this can't be modified)
00172     } type;
00173 };
00174
00175 struct swTexture {
00176     std::string path;
00177     std::vector<uint8_t> data;
00178 };
00179
00180 struct swAudio {
00181     std::string path;
00182     std::vector<uint8_t> data;
00183 };
00184
00185 struct swGame { // spgam [zip]
00186     swWorld world;
00187     std::vector<swModel> models;
00188     std::vector<swMaterial> materials;
00189     std::vector<swLevel> levels;
00190     std::vector<swScript> scripts;
00191     std::vector<swEtc> etc;
00192     std::vector<swTexture> textures;
00193     std::vector<swAudio> audio;
00194 };
00195
00196 namespace world {
00197     // we use pointers to swGame instead of swGame itself because the swGame struct is pretty big
```

```
00198     // and we don't want to copy it around a lot
00199
00200     typedef swGame* (*LoadGame)(const char*);
00201     typedef void (*FreeGame)(swGame*);
00202     typedef void (*SaveGame)(swGame*, const char*);
00203 }
00204
00205 struct WorldModule : public Module {
00206     world::LoadGame loadGame;
00207     world::FreeGame freeGame;
00208     world::SaveGame saveGame;
00209
00210     WorldModule() : Module("main", "wrld") {
00211         loadGame = (world::LoadGame)lib.getSymbol("loadGame");
00212         freeGame = (world::FreeGame)lib.getSymbol("freeGame");
00213         saveGame = (world::SaveGame)lib.getSymbol("saveGame");
00214     }
00215 };
```

8.13 game.h

```
00001 #pragma once
00002 #include "win/module.h"
00003 #include "gfx/module.h"
00004 #include "shdr/module.h"
00005 #include "tex/module.h"
00006 #include "text/module.h"
00007
00008 #include "asset.h"
00009
00010 struct GameContext {
00011     WindowModule winm;
00012     GraphicsModule gfxm;
00013     ShaderModule shdr;
00014     TextureModule texm;
00015     TextModule textm;
00016     AssetLoader assetm;
00017 };
00018
00019 class Game : Module {
00020     typedef int (*GameMain)(GameContext*);
00021     public:
00022     GameMain main;
00023     Game() : Module("game", "game") {
00024         if (!lib.valid()) {
00025             printf("Error loading main game module\n");
00026             return;
00027         }
00028         main = (GameMain)lib.getSymbol("game_main");
00029         if (!main) {
00030             printf("Error loading main game function\n");
00031             return;
00032         }
00033     }
00034 };
```

8.14 glutils.h

```
00001 #pragma once
00002 #ifdef _WIN32
00003 #include <Windows.h>
00004
00005 #elif __linux__
00006 #include <GL/glx.h>
00007 #include <GL/glxext.h>
00008 #include <dlfcn.h>
00009 #include <unistd.h>
00010 #include <sys/types.h>
00011 #else
00012 #error "Unsupported platform"
00013 #endif
00014
00015 #ifdef _WIN32
00016 extern "C" void *getProcAddress(const char *name) {
00017     void *p = (void *)wglGetProcAddress(name);
00018     if(p == 0 ||
00019        (p == (void*)0x1) || (p == (void*)0x2) || (p == (void*)0x3) ||
00020        (p == (void*)-1) )
00021     {
```

```
00022         HMODULE module = LoadLibraryA("opengl32.dll");
00023         p = (void *)GetProcAddress(module, name);
00024     }
00025
00026     return p;
00027 }
00028 #elif __linux__
00029 extern "C" void *getProcAddress(const char *name) {
00030     void *p = (void *)glXGetProcAddress((const GLubyte *)name);
00031     if(p == 0 ||
00032        (p == (void*)0x1) || (p == (void*)0x2) || (p == (void*)0x3) ||
00033        (p == (void*)-1) )
00034     {
00035         void *handle = dlopen("libGL.so.1", RTLD_LAZY);
00036         p = (void *)dlsym(handle, name);
00037     }
00038
00039     return p;
00040 }
00041 #else
00042 #error "Unsupported platform"
00043 #endif
```

8.15 clay.h

```
00001 // VERSION: 0.12
00002
00003 /*
00004     NOTE: In order to use this library you must define
00005     the following macro in exactly one file, _before_ including clay.h:
00006
00007     #define CLAY_IMPLEMENTATION
00008     #include "clay.h"
00009
00010     See the examples folder for details.
00011 */
00012
00013 #include <stdint.h>
00014 #include <stdbool.h>
00015 #include <stddef.h>
00016
00017 // -----------------------------------------
00018 // HEADER DECLARATIONS ---------------------
00019 // -----------------------------------------
00020
00021 #ifndef CLAY_HEADER
00022 #define CLAY_HEADER
00023
00024 #if !( \
00025     (defined(__cplusplus) && __cplusplus >= 202002L) || \
00026     (defined(__STDC__) && __STDC__ == 1 && defined(__STDC_VERSION__) && __STDC_VERSION__ >= 199901L)
    || \
00027     defined(_MSC_VER) \
00028 )
00029 #error "Clay requires C99, C++20, or MSVC"
00030 #endif
00031
00032 #ifdef CLAY_WASM
00033 #define CLAY_WASM_EXPORT(name) __attribute__((export_name(name)))
00034 #else
00035 #define CLAY_WASM_EXPORT(null)
00036 #endif
00037
00038 // Public Macro API ------------------------
00039
00040 #define CLAY__WRAPPER_TYPE(type) Clay__##type##Wrapper
00041 #define CLAY__WRAPPER_STRUCT(type) typedef struct { type wrapped; } CLAY__WRAPPER_TYPE(type)
00042 #define CLAY__CONFIG_WRAPPER(type, ...) (CLAY__INIT(CLAY__WRAPPER_TYPE(type)) { __VA_ARGS__ }).wrapped
00043
00044 #define CLAY__MAX(x, y) (((x) > (y)) ? (x) : (y))
00045 #define CLAY__MIN(x, y) (((x) < (y)) ? (x) : (y))
00046
00047 #define CLAY_LAYOUT(...)
      Clay__AttachLayoutConfig(Clay__StoreLayoutConfig(CLAY__CONFIG_WRAPPER(Clay_LayoutConfig,
      __VA_ARGS__)))
00048
00049 #define CLAY_RECTANGLE(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .rectangleElementConfig =
      Clay__StoreRectangleElementConfig(CLAY__CONFIG_WRAPPER(Clay_RectangleElementConfig, __VA_ARGS__)) },
      CLAY__ELEMENT_CONFIG_TYPE_RECTANGLE)
00050
00051 #define CLAY_TEXT_CONFIG(...)
      Clay__StoreTextElementConfig(CLAY__CONFIG_WRAPPER(Clay_TextElementConfig, __VA_ARGS__))
```

```
00052
00053 #define CLAY_IMAGE(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .imageElementConfig = Clay__StoreImageElementConfig(CLAY__CONFIG_WRAPPER(Clay_ImageElementConfig,
      __VA_ARGS__)) }, CLAY__ELEMENT_CONFIG_TYPE_IMAGE)
00054
00055 #define CLAY_FLOATING(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .floatingElementConfig =
      Clay__StoreFloatingElementConfig(CLAY__CONFIG_WRAPPER(Clay_FloatingElementConfig, __VA_ARGS__)) },
      CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER)
00056
00057 #define CLAY_CUSTOM_ELEMENT(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .customElementConfig = Clay__StoreCustomElementConfig(CLAY__CONFIG_WRAPPER(Clay_CustomElementConfig,
      __VA_ARGS__)) }, CLAY__ELEMENT_CONFIG_TYPE_CUSTOM)
00058
00059 #define CLAY_SCROLL(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .scrollElementConfig = Clay__StoreScrollElementConfig(CLAY__CONFIG_WRAPPER(Clay_ScrollElementConfig,
      __VA_ARGS__)) }, CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER)
00060
00061 #define CLAY_BORDER(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .borderElementConfig = Clay__StoreBorderElementConfig(CLAY__CONFIG_WRAPPER(Clay_BorderElementConfig,
      __VA_ARGS__)) }, CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER)
00062
00063 #define CLAY_BORDER_OUTSIDE(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .borderElementConfig = Clay__StoreBorderElementConfig(CLAY__INIT(Clay_BorderElementConfig) { .left =
      __VA_ARGS__, .right = __VA_ARGS__, .top = __VA_ARGS__, .bottom = __VA_ARGS__ }) },
      CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER)
00064
00065 #define CLAY_BORDER_OUTSIDE_RADIUS(width, color, radius)
      Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) { .borderElementConfig =
      Clay__StoreBorderElementConfig(CLAY__INIT(Clay_BorderElementConfig) { .left = { width, color }, .right
      = { width, color }, .top = { width, color }, .bottom = { width, color }, .cornerRadius =
      CLAY_CORNER_RADIUS(radius) })}, CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER)
00066
00067 #define CLAY_BORDER_ALL(...) Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) {
      .borderElementConfig = Clay__StoreBorderElementConfig(CLAY__INIT(Clay_BorderElementConfig) { .left =
      __VA_ARGS__, .right = __VA_ARGS__, .top = __VA_ARGS__, .bottom = __VA_ARGS__, .betweenChildren =
      __VA_ARGS__ }) }, CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER)
00068
00069 #define CLAY_BORDER_ALL_RADIUS(width, color, radius)
      Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) { .borderElementConfig =
      Clay__StoreBorderElementConfig(CLAY__INIT(Clay_BorderElementConfig) { .left = { width, color }, .right
      = { width, color }, .top = { width, color }, .bottom = { width, color }, .betweenChildren = { width,
      color }, .cornerRadius = CLAY_CORNER_RADIUS(radius)}) }, CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER)
00070
00071 #define CLAY_CORNER_RADIUS(radius) (CLAY__INIT(Clay_CornerRadius) { radius, radius, radius, radius })
00072
00073 #define CLAY_PADDING_ALL(padding) CLAY__CONFIG_WRAPPER(Clay_Padding, { padding, padding, padding,
      padding })
00074
00075 #define CLAY_SIZING_FIT(...) (CLAY__INIT(Clay_SizingAxis) { .size = { .minMax = { __VA_ARGS__ } },
      .type = CLAY__SIZING_TYPE_FIT })
00076
00077 #define CLAY_SIZING_GROW(...) (CLAY__INIT(Clay_SizingAxis) { .size = { .minMax = { __VA_ARGS__ } },
      .type = CLAY__SIZING_TYPE_GROW })
00078
00079 #define CLAY_SIZING_FIXED(fixedSize) (CLAY__INIT(Clay_SizingAxis) { .size = { .minMax = { fixedSize,
      fixedSize } }, .type = CLAY__SIZING_TYPE_FIXED })
00080
00081 #define CLAY_SIZING_PERCENT(percentOfParent) (CLAY__INIT(Clay_SizingAxis) { .size = { .percent =
      (percentOfParent) }, .type = CLAY__SIZING_TYPE_PERCENT })
00082
00083 #define CLAY_ID(label) Clay__AttachId(Clay__HashString(CLAY_STRING(label), 0, 0))
00084
00085 #define CLAY_IDI(label, index) Clay__AttachId(Clay__HashString(CLAY_STRING(label), index, 0))
00086
00087 #define CLAY_ID_LOCAL(label) CLAY_IDI_LOCAL(label, 0)
00088
00089 #define CLAY_IDI_LOCAL(label, index) Clay__AttachId(Clay__HashString(CLAY_STRING(label), index,
      Clay__GetParentElementId()))
00090
00091 #define CLAY__STRING_LENGTH(s) ((sizeof(s) / sizeof((s)[0])) - sizeof((s)[0]))
00092
00093 #define CLAY__ENSURE_STRING_LITERAL(x) ("" x "")
00094
00095 // Note: If an error led you here, it's because CLAY_STRING can only be used with string literals,
      i.e. CLAY_STRING("SomeString") and not CLAY_STRING(yourString)
00096 #define CLAY_STRING(string) (CLAY__INIT(Clay_String) { .length =
      CLAY__STRING_LENGTH(CLAY__ENSURE_STRING_LITERAL(string)), .chars = (string) })
00097
00098 #define CLAY_STRING_CONST(string) { .length =
      CLAY__STRING_LENGTH(CLAY__ENSURE_STRING_LITERAL(string)), .chars = (string) }
00099
00100 static uint8_t CLAY__ELEMENT_DEFINITION_LATCH;
00101
00102 // Publicly visible layout element macros ----------------------------------------------------
00103
00104 /* This macro looks scary on the surface, but is actually quite simple.
```

```
00105   It turns a macro call like this:
00106
00107   CLAY(
00108     CLAY_RECTANGLE(),
00109     CLAY_ID()
00110   ) {
00111       ...children declared here
00112   }
00113
00114   Into calls like this:
00115
00116   Clay_OpenElement();
00117   CLAY_RECTANGLE();
00118   CLAY_ID();
00119   Clay_ElementPostConfiguration();
00120   ...children declared here
00121   Clay_CloseElement();
00122
00123   The for loop will only ever run a single iteration, putting Clay__CloseElement() in the increment of
      the loop
00124   means that it will run after the body - where the children are declared. It just exists to make sure
      you don't forget
00125   to call Clay_CloseElement().
00126 */
00127 #define CLAY(...) \
00128     for (\
00129         CLAY__ELEMENT_DEFINITION_LATCH = (Clay__OpenElement(), __VA_ARGS__,
      Clay__ElementPostConfiguration(), 0); \
00130         CLAY__ELEMENT_DEFINITION_LATCH < 1; \
00131         ++CLAY__ELEMENT_DEFINITION_LATCH, Clay__CloseElement() \
00132     )
00133
00134 #define CLAY_TEXT(text, textConfig) Clay__OpenTextElement(text, textConfig)
00135
00136 #ifdef __cplusplus
00137
00138 #define CLAY__INIT(type) type
00139 #define CLAY__TYPEDEF(name, ...) typedef __VA_ARGS__ name; CLAY__WRAPPER_STRUCT(name)
00140 #define CLAY__ALIGNMENT(type) alignof(type)
00141 #define CLAY__POINTER_ALIGNMENT alignof(void *)
00142
00143 #define CLAY_PACKED_ENUM enum : uint8_t
00144
00145 #define CLAY__DEFAULT_STRUCT {}
00146
00147 #else
00148
00149 #define CLAY__INIT(type) (type)
00150
00151 #define CLAY__ALIGNMENT_STRUCT(type) struct Clay__Align##type { char c; type x; }
00152 #define CLAY__TYPEDEF(name, ...) typedef __VA_ARGS__ name; CLAY__ALIGNMENT_STRUCT(name);
      CLAY__WRAPPER_STRUCT(name)
00153 #define CLAY__ALIGNMENT(type) (offsetof(struct Clay__Align##type, x))
00154 #define CLAY__POINTER_ALIGNMENT CLAY__ALIGNMENT(pointer)
00155
00156 // NOTE: If you need to get the offset for other standard types in the future, add them here.
00157 struct Clay__Alignpointer { char c; void *x; };
00158 CLAY__ALIGNMENT_STRUCT(bool);
00159 CLAY__ALIGNMENT_STRUCT(uint8_t);
00160 CLAY__ALIGNMENT_STRUCT(int32_t);
00161
00162 #if defined(_MSC_VER) && !defined(__clang__)
00163 #define CLAY_PACKED_ENUM __pragma(pack(push, 1)) enum __pragma(pack(pop))
00164 #else
00165 #define CLAY_PACKED_ENUM enum __attribute__((__packed__))
00166 #endif
00167
00168 #if __STDC_VERSION__ >= 202311L
00169 #define CLAY__DEFAULT_STRUCT {}
00170 #else
00171 #define CLAY__DEFAULT_STRUCT {0}
00172 #endif
00173
00174 #endif // __cplusplus
00175
00176 #ifdef __cplusplus
00177 extern "C" {
00178 #endif
00179
00180 // Utility Structs ------------------------
00181 // Note: Clay_String is not guaranteed to be null terminated. It may be if created from a literal C
      string,
00182 // but it is also used to represent slices.
00183 CLAY__TYPEDEF(Clay_String, struct {
00184     int32_t length;
00185     const char *chars;
00186 });
```

```
00187
00188 CLAY__TYPEDEF(Clay__StringArray, struct {
00189     int32_t capacity;
00190     int32_t length;
00191     Clay_String *internalArray;
00192 });
00193
00194 CLAY__TYPEDEF(Clay_StringSlice, struct {
00195     int32_t length;
00196     const char *chars;
00197     // The source string / char* that this slice was derived from
00198     const char *baseChars;
00199 });
00200
00201 typedef struct Clay_Context Clay_Context;
00202
00203 CLAY__TYPEDEF(Clay_Arena, struct {
00204     uintptr_t nextAllocation;
00205     size_t capacity;
00206     char *memory;
00207 });
00208
00209 CLAY__TYPEDEF(Clay_Dimensions, struct {
00210     float width, height;
00211 });
00212
00213 CLAY__TYPEDEF(Clay_Vector2, struct {
00214     float x, y;
00215 });
00216
00217 CLAY__TYPEDEF(Clay_Color, struct {
00218     float r, g, b, a;
00219 });
00220
00221 CLAY__TYPEDEF(Clay_BoundingBox, struct {
00222     float x, y, width, height;
00223 });
00224
00225 // baseId + offset = id
00226 CLAY__TYPEDEF(Clay_ElementId, struct {
00227     uint32_t id;
00228     uint32_t offset;
00229     uint32_t baseId;
00230     Clay_String stringId;
00231 });
00232
00233 CLAY__TYPEDEF(Clay_CornerRadius, struct {
00234     float topLeft;
00235     float topRight;
00236     float bottomLeft;
00237     float bottomRight;
00238 });
00239
00240 CLAY__TYPEDEF(Clay__ElementConfigType, CLAY_PACKED_ENUM {
00241     CLAY__ELEMENT_CONFIG_TYPE_NONE = 0,
00242     CLAY__ELEMENT_CONFIG_TYPE_RECTANGLE = 1,
00243     CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER = 2,
00244     CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER = 4,
00245     CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER = 8,
00246     CLAY__ELEMENT_CONFIG_TYPE_IMAGE = 16,
00247     CLAY__ELEMENT_CONFIG_TYPE_TEXT = 32,
00248     CLAY__ELEMENT_CONFIG_TYPE_CUSTOM = 64,
00249 });
00250
00251 // Element Configs ---------------------------
00252 // Layout
00253 CLAY__TYPEDEF(Clay_LayoutDirection, CLAY_PACKED_ENUM {
00254     CLAY_LEFT_TO_RIGHT,
00255     CLAY_TOP_TO_BOTTOM,
00256 });
00257
00258 CLAY__TYPEDEF(Clay_LayoutAlignmentX, CLAY_PACKED_ENUM {
00259     CLAY_ALIGN_X_LEFT,
00260     CLAY_ALIGN_X_RIGHT,
00261     CLAY_ALIGN_X_CENTER,
00262 });
00263
00264 CLAY__TYPEDEF(Clay_LayoutAlignmentY, CLAY_PACKED_ENUM {
00265     CLAY_ALIGN_Y_TOP,
00266     CLAY_ALIGN_Y_BOTTOM,
00267     CLAY_ALIGN_Y_CENTER,
00268 });
00269
00270 CLAY__TYPEDEF(Clay__SizingType, CLAY_PACKED_ENUM {
00271     CLAY__SIZING_TYPE_FIT,
00272     CLAY__SIZING_TYPE_GROW,
00273     CLAY__SIZING_TYPE_PERCENT,
```

```
00274     CLAY__SIZING_TYPE_FIXED,
00275 });
00276
00277 CLAY__TYPEDEF(Clay_ChildAlignment, struct {
00278     Clay_LayoutAlignmentX x;
00279     Clay_LayoutAlignmentY y;
00280 });
00281
00282 CLAY__TYPEDEF(Clay_SizingMinMax, struct {
00283     float min;
00284     float max;
00285 });
00286
00287 CLAY__TYPEDEF(Clay_SizingAxis, struct {
00288     union {
00289         Clay_SizingMinMax minMax;
00290         float percent;
00291     } size;
00292     Clay__SizingType type;
00293 });
00294
00295 CLAY__TYPEDEF(Clay_Sizing, struct {
00296     Clay_SizingAxis width;
00297     Clay_SizingAxis height;
00298 });
00299
00300 CLAY__TYPEDEF(Clay_Padding, struct {
00301     uint16_t left;
00302     uint16_t right;
00303     uint16_t top;
00304     uint16_t bottom;
00305 });
00306
00307 CLAY__TYPEDEF(Clay_LayoutConfig, struct {
00308     Clay_Sizing sizing;
00309     Clay_Padding padding;
00310     uint16_t childGap;
00311     Clay_ChildAlignment childAlignment;
00312     Clay_LayoutDirection layoutDirection;
00313 });
00314
00315 extern Clay_LayoutConfig CLAY_LAYOUT_DEFAULT;
00316
00317 // Rectangle
00318 // NOTE: Not declared in the typedef as an ifdef inside macro arguments is UB
00319 struct Clay_RectangleElementConfig {
00320     Clay_Color color;
00321     Clay_CornerRadius cornerRadius;
00322     #ifdef CLAY_EXTEND_CONFIG_RECTANGLE
00323     CLAY_EXTEND_CONFIG_RECTANGLE
00324     #endif
00325 };
00326 CLAY__TYPEDEF(Clay_RectangleElementConfig, struct Clay_RectangleElementConfig);
00327
00328 // Text
00329 CLAY__TYPEDEF(Clay_TextElementConfigWrapMode, enum {
00330     CLAY_TEXT_WRAP_WORDS,
00331     CLAY_TEXT_WRAP_NEWLINES,
00332     CLAY_TEXT_WRAP_NONE,
00333 });
00334
00335 struct Clay_TextElementConfig {
00336     Clay_Color textColor;
00337     uint16_t fontId;
00338     uint16_t fontSize;
00339     uint16_t letterSpacing;
00340     uint16_t lineHeight;
00341     Clay_TextElementConfigWrapMode wrapMode;
00342     #ifdef CLAY_EXTEND_CONFIG_TEXT
00343     CLAY_EXTEND_CONFIG_TEXT
00344     #endif
00345 };
00346 CLAY__TYPEDEF(Clay_TextElementConfig, struct Clay_TextElementConfig);
00347
00348 // Image
00349 struct Clay_ImageElementConfig {
00350     void *imageData;
00351     Clay_Dimensions sourceDimensions;
00352     #ifdef CLAY_EXTEND_CONFIG_IMAGE
00353     CLAY_EXTEND_CONFIG_IMAGE
00354     #endif
00355 };
00356 CLAY__TYPEDEF(Clay_ImageElementConfig, struct Clay_ImageElementConfig);
00357
00358 // Floating
00359 CLAY__TYPEDEF(Clay_FloatingAttachPointType, CLAY_PACKED_ENUM {
00360     CLAY_ATTACH_POINT_LEFT_TOP,
```

```
00361     CLAY_ATTACH_POINT_LEFT_CENTER,
00362     CLAY_ATTACH_POINT_LEFT_BOTTOM,
00363     CLAY_ATTACH_POINT_CENTER_TOP,
00364     CLAY_ATTACH_POINT_CENTER_CENTER,
00365     CLAY_ATTACH_POINT_CENTER_BOTTOM,
00366     CLAY_ATTACH_POINT_RIGHT_TOP,
00367     CLAY_ATTACH_POINT_RIGHT_CENTER,
00368     CLAY_ATTACH_POINT_RIGHT_BOTTOM,
00369 });
00370
00371 CLAY__TYPEDEF(Clay_FloatingAttachPoints, struct {
00372     Clay_FloatingAttachPointType element;
00373     Clay_FloatingAttachPointType parent;
00374 });
00375
00376 CLAY__TYPEDEF(Clay_PointerCaptureMode, enum {
00377     CLAY_POINTER_CAPTURE_MODE_CAPTURE,
00378 //    CLAY_POINTER_CAPTURE_MODE_PARENT, TODO pass pointer through to attached parent
00379     CLAY_POINTER_CAPTURE_MODE_PASSTHROUGH,
00380 });
00381
00382 CLAY__TYPEDEF(Clay_FloatingElementConfig, struct {
00383     Clay_Vector2 offset;
00384     Clay_Dimensions expand;
00385     uint16_t zIndex;
00386     uint32_t parentId;
00387     Clay_FloatingAttachPoints attachment;
00388     Clay_PointerCaptureMode pointerCaptureMode;
00389 });
00390
00391 // Custom
00392 struct Clay_CustomElementConfig {
00393     #ifndef CLAY_EXTEND_CONFIG_CUSTOM
00394     void *customData;
00395     #else
00396     CLAY_EXTEND_CONFIG_CUSTOM
00397     #endif
00398 };
00399 CLAY__TYPEDEF(Clay_CustomElementConfig, struct Clay_CustomElementConfig);
00400
00401 // Scroll
00402 CLAY__TYPEDEF(Clay_ScrollElementConfig, struct {
00403     bool horizontal;
00404     bool vertical;
00405 });
00406
00407 // Border
00408 CLAY__TYPEDEF(Clay_Border, struct {
00409     uint32_t width;
00410     Clay_Color color;
00411 });
00412
00413 struct Clay_BorderElementConfig {
00414     Clay_Border left;
00415     Clay_Border right;
00416     Clay_Border top;
00417     Clay_Border bottom;
00418     Clay_Border betweenChildren;
00419     Clay_CornerRadius cornerRadius;
00420     #ifdef CLAY_EXTEND_CONFIG_BORDER
00421     CLAY_EXTEND_CONFIG_BORDER
00422     #endif
00423 };
00424 CLAY__TYPEDEF(Clay_BorderElementConfig, struct Clay_BorderElementConfig);
00425
00426 CLAY__TYPEDEF(Clay_ElementConfigUnion, union {
00427     Clay_RectangleElementConfig *rectangleElementConfig;
00428     Clay_TextElementConfig *textElementConfig;
00429     Clay_ImageElementConfig *imageElementConfig;
00430     Clay_FloatingElementConfig *floatingElementConfig;
00431     Clay_CustomElementConfig *customElementConfig;
00432     Clay_ScrollElementConfig *scrollElementConfig;
00433     Clay_BorderElementConfig *borderElementConfig;
00434 });
00435
00436 CLAY__TYPEDEF(Clay_ElementConfig, struct {
00437     Clay__ElementConfigType type;
00438     Clay_ElementConfigUnion config;
00439 });
00440
00441 // Miscellaneous Structs & Enums ---------------------------------
00442 CLAY__TYPEDEF(Clay_ScrollContainerData, struct {
00443     // Note: This is a pointer to the real internal scroll position, mutating it may cause a change in
      final layout.
00444     // Intended for use with external functionality that modifies scroll position, such as scroll bars
      or auto scrolling.
00445     Clay_Vector2 *scrollPosition;
```

```
00446     Clay_Dimensions scrollContainerDimensions;
00447     Clay_Dimensions contentDimensions;
00448     Clay_ScrollElementConfig config;
00449     // Indicates whether an actual scroll container matched the provided ID or if the default struct
      was returned.
00450     bool found;
00451 });
00452
00453 CLAY__TYPEDEF(Clay_ElementData, struct
00454 {
00455     Clay_BoundingBox boundingBox;
00456     // Indicates whether an actual Element matched the provided ID or if the default struct was
      returned.
00457     bool found;
00458 });
00459
00460 CLAY__TYPEDEF(Clay_RenderCommandType, CLAY_PACKED_ENUM {
00461     CLAY_RENDER_COMMAND_TYPE_NONE,
00462     CLAY_RENDER_COMMAND_TYPE_RECTANGLE,
00463     CLAY_RENDER_COMMAND_TYPE_BORDER,
00464     CLAY_RENDER_COMMAND_TYPE_TEXT,
00465     CLAY_RENDER_COMMAND_TYPE_IMAGE,
00466     CLAY_RENDER_COMMAND_TYPE_SCISSOR_START,
00467     CLAY_RENDER_COMMAND_TYPE_SCISSOR_END,
00468     CLAY_RENDER_COMMAND_TYPE_CUSTOM,
00469 });
00470
00471 CLAY__TYPEDEF(Clay_RenderCommand, struct {
00472     Clay_BoundingBox boundingBox;
00473     Clay_ElementConfigUnion config;
00474     Clay_StringSlice text; // TODO I wish there was a way to avoid having to have this on every render
      command
00475     int32_t zIndex;
00476     uint32_t id;
00477     Clay_RenderCommandType commandType;
00478 });
00479
00480 CLAY__TYPEDEF(Clay_RenderCommandArray, struct {
00481     int32_t capacity;
00482     int32_t length;
00483     Clay_RenderCommand *internalArray;
00484 });
00485
00486 CLAY__TYPEDEF(Clay_PointerDataInteractionState, enum {
00487     CLAY_POINTER_DATA_PRESSED_THIS_FRAME,
00488     CLAY_POINTER_DATA_PRESSED,
00489     CLAY_POINTER_DATA_RELEASED_THIS_FRAME,
00490     CLAY_POINTER_DATA_RELEASED,
00491 });
00492
00493 CLAY__TYPEDEF(Clay_PointerData, struct {
00494     Clay_Vector2 position;
00495     Clay_PointerDataInteractionState state;
00496 });
00497
00498 CLAY__TYPEDEF(Clay_ErrorType, enum {
00499     CLAY_ERROR_TYPE_TEXT_MEASUREMENT_FUNCTION_NOT_PROVIDED,
00500     CLAY_ERROR_TYPE_ARENA_CAPACITY_EXCEEDED,
00501     CLAY_ERROR_TYPE_ELEMENTS_CAPACITY_EXCEEDED,
00502     CLAY_ERROR_TYPE_TEXT_MEASUREMENT_CAPACITY_EXCEEDED,
00503     CLAY_ERROR_TYPE_DUPLICATE_ID,
00504     CLAY_ERROR_TYPE_FLOATING_CONTAINER_PARENT_NOT_FOUND,
00505     CLAY_ERROR_TYPE_INTERNAL_ERROR,
00506 });
00507
00508 CLAY__TYPEDEF(Clay_ErrorData, struct {
00509     Clay_ErrorType errorType;
00510     Clay_String errorText;
00511     uintptr_t userData;
00512 });
00513
00514 CLAY__TYPEDEF(Clay_ErrorHandler, struct {
00515     void (*errorHandlerFunction)(Clay_ErrorData errorText);
00516     uintptr_t userData;
00517 });
00518
00519 // Function Forward Declarations ---------------------------------
00520 // Public API functions ---
00521 uint32_t Clay_MinMemorySize(void);
00522 Clay_Arena Clay_CreateArenaWithCapacityAndMemory(uint32_t capacity, void *offset);
00523 void Clay_SetPointerState(Clay_Vector2 position, bool pointerDown);
00524 Clay_Context* Clay_Initialize(Clay_Arena arena, Clay_Dimensions layoutDimensions, Clay_ErrorHandler
      errorHandler);
00525 Clay_Context* Clay_GetCurrentContext(void);
00526 void Clay_SetCurrentContext(Clay_Context* context);
00527 void Clay_UpdateScrollContainers(bool enableDragScrolling, Clay_Vector2 scrollDelta, float deltaTime);
00528 void Clay_SetLayoutDimensions(Clay_Dimensions dimensions);
```

```
00529 void Clay_BeginLayout(void);
00530 Clay_RenderCommandArray Clay_EndLayout(void);
00531 Clay_ElementId Clay_GetElementId(Clay_String idString);
00532 Clay_ElementId Clay_GetElementIdWithIndex(Clay_String idString, uint32_t index);
00533 Clay_ElementData Clay_GetElementData (Clay_ElementId id);
00534 bool Clay_Hovered(void);
00535 void Clay_OnHover(void (*onHoverFunction)(Clay_ElementId elementId, Clay_PointerData pointerData,
      intptr_t userData), intptr_t userData);
00536 bool Clay_PointerOver(Clay_ElementId elementId);
00537 Clay_ScrollContainerData Clay_GetScrollContainerData(Clay_ElementId id);
00538 void Clay_SetMeasureTextFunction(Clay_Dimensions (*measureTextFunction)(Clay_StringSlice text,
      Clay_TextElementConfig *config, uintptr_t userData), uintptr_t userData);
00539 void Clay_SetQueryScrollOffsetFunction(Clay_Vector2 (*queryScrollOffsetFunction)(uint32_t elementId,
      uintptr_t userData), uintptr_t userData);
00540 Clay_RenderCommand * Clay_RenderCommandArray_Get(Clay_RenderCommandArray* array, int32_t index);
00541 void Clay_SetDebugModeEnabled(bool enabled);
00542 bool Clay_IsDebugModeEnabled(void);
00543 void Clay_SetCullingEnabled(bool enabled);
00544 int32_t Clay_GetMaxElementCount(void);
00545 void Clay_SetMaxElementCount(int32_t maxElementCount);
00546 int32_t Clay_GetMaxMeasureTextCacheWordCount(void);
00547 void Clay_SetMaxMeasureTextCacheWordCount(int32_t maxMeasureTextCacheWordCount);
00548 void Clay_ResetMeasureTextCache(void);
00549
00550 // Internal API functions required by macros
00551 void Clay__OpenElement(void);
00552 void Clay__CloseElement(void);
00553 Clay_LayoutConfig * Clay__StoreLayoutConfig(Clay_LayoutConfig config);
00554 void Clay__ElementPostConfiguration(void);
00555 void Clay__AttachId(Clay_ElementId id);
00556 void Clay__AttachLayoutConfig(Clay_LayoutConfig *config);
00557 void Clay__AttachElementConfig(Clay_ElementConfigUnion config, Clay__ElementConfigType type);
00558 Clay_RectangleElementConfig * Clay__StoreRectangleElementConfig(Clay_RectangleElementConfig config);
00559 Clay_TextElementConfig * Clay__StoreTextElementConfig(Clay_TextElementConfig config);
00560 Clay_ImageElementConfig * Clay__StoreImageElementConfig(Clay_ImageElementConfig config);
00561 Clay_FloatingElementConfig * Clay__StoreFloatingElementConfig(Clay_FloatingElementConfig config);
00562 Clay_CustomElementConfig * Clay__StoreCustomElementConfig(Clay_CustomElementConfig config);
00563 Clay_ScrollElementConfig * Clay__StoreScrollElementConfig(Clay_ScrollElementConfig config);
00564 Clay_BorderElementConfig * Clay__StoreBorderElementConfig(Clay_BorderElementConfig config);
00565 Clay_ElementId Clay__HashString(Clay_String key, uint32_t offset, uint32_t seed);
00566 void Clay__OpenTextElement(Clay_String text, Clay_TextElementConfig *textConfig);
00567 uint32_t Clay__GetParentElementId(void);
00568
00569 extern Clay_Color Clay__debugViewHighlightColor;
00570 extern uint32_t Clay__debugViewWidth;
00571
00572 #ifdef __cplusplus
00573 }
00574 #endif
00575
00576 #endif // CLAY_HEADER
00577
00578 // -----------------------------------------
00579 // IMPLEMENTATION --------------------------
00580 // -----------------------------------------
00581 #ifdef CLAY_IMPLEMENTATION
00582 #undef CLAY_IMPLEMENTATION
00583
00584 #ifndef CLAY__NULL
00585 #define CLAY__NULL 0
00586 #endif
00587
00588 #ifndef CLAY__MAXFLOAT
00589 #define CLAY__MAXFLOAT 3.4028234663852886e+38F
00590 #endif
00591
00592 Clay_Context *Clay__currentContext;
00593 int32_t Clay__defaultMaxElementCount = 8192;
00594 int32_t Clay__defaultMaxMeasureTextWordCacheCount = 16384;
00595
00596 void Clay__ErrorHandlerFunctionDefault(Clay_ErrorData errorText) {
00597     (void) errorText;
00598 }
00599
00600 Clay_String CLAY__SPACECHAR = { .length = 1, .chars = " " };
00601 Clay_String CLAY__STRING_DEFAULT = { .length = 0, .chars = NULL };
00602
00603 CLAY__TYPEDEF(Clay_BooleanWarnings, struct {
00604     bool maxElementsExceeded;
00605     bool maxRenderCommandsExceeded;
00606     bool maxTextMeasureCacheExceeded;
00607     bool textMeasurementFunctionNotSet;
00608 });
00609
00610 CLAY__TYPEDEF(Clay__Warning, struct {
00611     Clay_String baseMessage;
00612     Clay_String dynamicMessage;
```

```
00613 });
00614
00615 Clay__Warning CLAY__WARNING_DEFAULT = CLAY__DEFAULT_STRUCT;
00616
00617 CLAY__TYPEDEF(Clay__WarningArray, struct {
00618     int32_t capacity;
00619     int32_t length;
00620     Clay__Warning *internalArray;
00621 });
00622
00623 Clay__WarningArray Clay__WarningArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena);
00624 Clay__Warning *Clay__WarningArray_Add(Clay__WarningArray *array, Clay__Warning item);
00625 void* Clay__Array_Allocate_Arena(int32_t capacity, uint32_t itemSize, uint32_t alignment, Clay_Arena
      *arena);
00626 bool Clay__Array_RangeCheck(int32_t index, int32_t length);
00627 bool Clay__Array_AddCapacityCheck(int32_t length, int32_t capacity);
00628
00629 // __GENERATED__ template array_define,array_allocate TYPE=bool NAME=Clay__BoolArray
00630 #pragma region generated
00631 CLAY__TYPEDEF(Clay__BoolArray, struct
00632 {
00633     int32_t capacity;
00634     int32_t length;
00635     bool *internalArray;
00636 });
00637 Clay__BoolArray Clay__BoolArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
00638     return CLAY__INIT(Clay__BoolArray){.capacity = capacity, .length = 0, .internalArray = (bool
      *)Clay__Array_Allocate_Arena(capacity, sizeof(bool), CLAY__ALIGNMENT(bool), arena)};
00639 }
00640 #pragma endregion
00641 // __GENERATED__ template
00642
00643 Clay_ElementId CLAY__ELEMENT_ID_DEFAULT = CLAY__DEFAULT_STRUCT;
00644
00645 // __GENERATED__ template array_define,array_allocate,array_get,array_add TYPE=Clay_ElementId
      NAME=Clay__ElementIdArray DEFAULT_VALUE=&CLAY__ELEMENT_ID_DEFAULT
00646 #pragma region generated
00647 CLAY__TYPEDEF(Clay__ElementIdArray, struct
00648 {
00649     int32_t capacity;
00650     int32_t length;
00651     Clay_ElementId *internalArray;
00652 });
00653 Clay__ElementIdArray Clay__ElementIdArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
00654     return CLAY__INIT(Clay__ElementIdArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay_ElementId *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_ElementId),
      CLAY__ALIGNMENT(Clay_ElementId), arena)};
00655 }
00656 Clay_ElementId *Clay__ElementIdArray_Get(Clay__ElementIdArray *array, int32_t index) {
00657     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__ELEMENT_ID_DEFAULT;
00658 }
00659 Clay_ElementId *Clay__ElementIdArray_Add(Clay__ElementIdArray *array, Clay_ElementId item) {
00660     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00661         array->internalArray[array->length++] = item;
00662         return &array->internalArray[array->length - 1];
00663     }
00664     return &CLAY__ELEMENT_ID_DEFAULT;
00665 }
00666 #pragma endregion
00667 // __GENERATED__ template
00668
00669 Clay_ElementConfig CLAY__ELEMENT_CONFIG_DEFAULT = {CLAY__ELEMENT_CONFIG_TYPE_NONE,
      CLAY__DEFAULT_STRUCT};
00670
00671 // __GENERATED__ template
      array_define,array_define_slice,array_allocate,array_get,array_add,array_get_slice
      TYPE=Clay_ElementConfig NAME=Clay__ElementConfigArray DEFAULT_VALUE=&CLAY__ELEMENT_CONFIG_DEFAULT
00672 #pragma region generated
00673 CLAY__TYPEDEF(Clay__ElementConfigArray, struct
00674 {
00675     int32_t capacity;
00676     int32_t length;
00677     Clay_ElementConfig *internalArray;
00678 });
00679 CLAY__TYPEDEF(Clay__ElementConfigArraySlice, struct
00680 {
00681     int32_t length;
00682     Clay_ElementConfig *internalArray;
00683 });
00684 Clay__ElementConfigArray Clay__ElementConfigArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena)
      {
00685     return CLAY__INIT(Clay__ElementConfigArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay_ElementConfig *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_ElementConfig),
      CLAY__ALIGNMENT(Clay_ElementConfig), arena)};
00686 }
00687 Clay_ElementConfig *Clay__ElementConfigArray_Get(Clay__ElementConfigArray *array, int32_t index) {
```

```
00688     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__ELEMENT_CONFIG_DEFAULT;
00689 }
00690 Clay_ElementConfig *Clay__ElementConfigArray_Add(Clay__ElementConfigArray *array, Clay_ElementConfig
      item) {
00691     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00692         array->internalArray[array->length++] = item;
00693         return &array->internalArray[array->length - 1];
00694     }
00695     return &CLAY__ELEMENT_CONFIG_DEFAULT;
00696 }
00697 Clay_ElementConfig *Clay__ElementConfigArraySlice_Get(Clay__ElementConfigArraySlice *slice, int32_t
      index) {
00698     return Clay__Array_RangeCheck(index, slice->length) ? &slice->internalArray[index] :
      &CLAY__ELEMENT_CONFIG_DEFAULT;
00699 }
00700 #pragma endregion
00701 // __GENERATED__ template
00702
00703 Clay_LayoutConfig CLAY_LAYOUT_DEFAULT = { .sizing = { .width = { .size = { .minMax = {0,
      CLAY__MAXFLOAT } }, .type = CLAY__SIZING_TYPE_FIT }, .height = { .size = { .minMax = {0,
      CLAY__MAXFLOAT } }, .type = CLAY__SIZING_TYPE_FIT } } };
00704
00705 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_LayoutConfig
      NAME=Clay__LayoutConfigArray DEFAULT_VALUE=&CLAY_LAYOUT_DEFAULT
00706 #pragma region generated
00707 CLAY__TYPEDEF(Clay__LayoutConfigArray, struct
00708 {
00709     int32_t capacity;
00710     int32_t length;
00711     Clay_LayoutConfig *internalArray;
00712 });
00713 Clay__LayoutConfigArray Clay__LayoutConfigArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
00714     return CLAY__INIT(Clay__LayoutConfigArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay_LayoutConfig *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_LayoutConfig),
      CLAY__ALIGNMENT(Clay_LayoutConfig), arena)};
00715 }
00716 Clay_LayoutConfig *Clay__LayoutConfigArray_Add(Clay__LayoutConfigArray *array, Clay_LayoutConfig item)
      {
00717     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00718         array->internalArray[array->length++] = item;
00719         return &array->internalArray[array->length - 1];
00720     }
00721     return &CLAY_LAYOUT_DEFAULT;
00722 }
00723 #pragma endregion
00724 // __GENERATED__ template
00725
00726 Clay_RectangleElementConfig CLAY__RECTANGLE_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00727
00728 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_RectangleElementConfig
      NAME=Clay__RectangleElementConfigArray DEFAULT_VALUE=&CLAY__RECTANGLE_ELEMENT_CONFIG_DEFAULT
00729 #pragma region generated
00730 CLAY__TYPEDEF(Clay__RectangleElementConfigArray, struct
00731 {
00732     int32_t capacity;
00733     int32_t length;
00734     Clay_RectangleElementConfig *internalArray;
00735 });
00736 Clay__RectangleElementConfigArray Clay__RectangleElementConfigArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
00737     return CLAY__INIT(Clay__RectangleElementConfigArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay_RectangleElementConfig *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay_RectangleElementConfig), CLAY__ALIGNMENT(Clay_RectangleElementConfig), arena)};
00738 }
00739 Clay_RectangleElementConfig *Clay__RectangleElementConfigArray_Add(Clay__RectangleElementConfigArray
      *array, Clay_RectangleElementConfig item) {
00740     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00741         array->internalArray[array->length++] = item;
00742         return &array->internalArray[array->length - 1];
00743     }
00744     return &CLAY__RECTANGLE_ELEMENT_CONFIG_DEFAULT;
00745 }
00746 #pragma endregion
00747 // __GENERATED__ template
00748
00749 Clay_TextElementConfig CLAY__TEXT_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00750
00751 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_TextElementConfig
      NAME=Clay__TextElementConfigArray DEFAULT_VALUE=&CLAY__TEXT_ELEMENT_CONFIG_DEFAULT
00752 #pragma region generated
00753 CLAY__TYPEDEF(Clay__TextElementConfigArray, struct
00754 {
00755     int32_t capacity;
00756     int32_t length;
00757     Clay_TextElementConfig *internalArray;
00758 });
```

```
00759 Clay__TextElementConfigArray Clay__TextElementConfigArray_Allocate_Arena(int32_t capacity, Clay_Arena
      *arena) {
00760     return CLAY__INIT(Clay__TextElementConfigArray){.capacity = capacity, .length = 0, .internalArray
      = (Clay_TextElementConfig *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_TextElementConfig),
      CLAY__ALIGNMENT(Clay_TextElementConfig), arena)};
00761 }
00762 Clay_TextElementConfig *Clay__TextElementConfigArray_Add(Clay__TextElementConfigArray *array,
      Clay_TextElementConfig item) {
00763     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00764         array->internalArray[array->length++] = item;
00765         return &array->internalArray[array->length - 1];
00766     }
00767     return &CLAY__TEXT_ELEMENT_CONFIG_DEFAULT;
00768 }
00769 #pragma endregion
00770 // __GENERATED__ template
00771
00772 Clay_ImageElementConfig CLAY__IMAGE_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00773
00774 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_ImageElementConfig
      NAME=Clay__ImageElementConfigArray DEFAULT_VALUE=&CLAY__IMAGE_ELEMENT_CONFIG_DEFAULT
00775 #pragma region generated
00776 CLAY__TYPEDEF(Clay__ImageElementConfigArray, struct
00777 {
00778     int32_t capacity;
00779     int32_t length;
00780     Clay_ImageElementConfig *internalArray;
00781 });
00782 Clay__ImageElementConfigArray Clay__ImageElementConfigArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
00783     return CLAY__INIT(Clay__ImageElementConfigArray){.capacity = capacity, .length = 0, .internalArray
      = (Clay_ImageElementConfig *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_ImageElementConfig),
      CLAY__ALIGNMENT(Clay_ImageElementConfig), arena)};
00784 }
00785 Clay_ImageElementConfig *Clay__ImageElementConfigArray_Add(Clay__ImageElementConfigArray *array,
      Clay_ImageElementConfig item) {
00786     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00787         array->internalArray[array->length++] = item;
00788         return &array->internalArray[array->length - 1];
00789     }
00790     return &CLAY__IMAGE_ELEMENT_CONFIG_DEFAULT;
00791 }
00792 #pragma endregion
00793 // __GENERATED__ template
00794
00795 Clay_FloatingElementConfig CLAY__FLOATING_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00796
00797 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_FloatingElementConfig
      NAME=Clay__FloatingElementConfigArray DEFAULT_VALUE=&CLAY__FLOATING_ELEMENT_CONFIG_DEFAULT
00798 #pragma region generated
00799 CLAY__TYPEDEF(Clay__FloatingElementConfigArray, struct
00800 {
00801     int32_t capacity;
00802     int32_t length;
00803     Clay_FloatingElementConfig *internalArray;
00804 });
00805 Clay__FloatingElementConfigArray Clay__FloatingElementConfigArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
00806     return CLAY__INIT(Clay__FloatingElementConfigArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay_FloatingElementConfig *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay_FloatingElementConfig), CLAY__ALIGNMENT(Clay_FloatingElementConfig), arena)};
00807 }
00808 Clay_FloatingElementConfig *Clay__FloatingElementConfigArray_Add(Clay__FloatingElementConfigArray
      *array, Clay_FloatingElementConfig item) {
00809     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00810         array->internalArray[array->length++] = item;
00811         return &array->internalArray[array->length - 1];
00812     }
00813     return &CLAY__FLOATING_ELEMENT_CONFIG_DEFAULT;
00814 }
00815 #pragma endregion
00816 // __GENERATED__ template
00817
00818 Clay_CustomElementConfig CLAY__CUSTOM_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00819
00820 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_CustomElementConfig
      NAME=Clay__CustomElementConfigArray DEFAULT_VALUE=&CLAY__CUSTOM_ELEMENT_CONFIG_DEFAULT
00821 #pragma region generated
00822 CLAY__TYPEDEF(Clay__CustomElementConfigArray, struct
00823 {
00824     int32_t capacity;
00825     int32_t length;
00826     Clay_CustomElementConfig *internalArray;
00827 });
00828 Clay__CustomElementConfigArray Clay__CustomElementConfigArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
00829     return CLAY__INIT(Clay__CustomElementConfigArray){.capacity = capacity, .length = 0,
```

```
    .internalArray = (Clay_CustomElementConfig *)Clay__Array_Allocate_Arena(capacity,
    sizeof(Clay_CustomElementConfig), CLAY__ALIGNMENT(Clay_CustomElementConfig), arena)};
00830 }
00831 Clay_CustomElementConfig *Clay__CustomElementConfigArray_Add(Clay__CustomElementConfigArray *array,
    Clay_CustomElementConfig item) {
00832     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00833         array->internalArray[array->length++] = item;
00834         return &array->internalArray[array->length - 1];
00835     }
00836     return &CLAY__CUSTOM_ELEMENT_CONFIG_DEFAULT;
00837 }
00838 #pragma endregion
00839 // __GENERATED__ template
00840
00841 Clay_ScrollElementConfig CLAY__SCROLL_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00842
00843 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_ScrollElementConfig
    NAME=Clay__ScrollElementConfigArray DEFAULT_VALUE=&CLAY__SCROLL_ELEMENT_CONFIG_DEFAULT
00844 #pragma region generated
00845 CLAY__TYPEDEF(Clay__ScrollElementConfigArray, struct
00846 {
00847     int32_t capacity;
00848     int32_t length;
00849     Clay_ScrollElementConfig *internalArray;
00850 });
00851 Clay__ScrollElementConfigArray Clay__ScrollElementConfigArray_Allocate_Arena(int32_t capacity,
    Clay_Arena *arena) {
00852     return CLAY__INIT(Clay__ScrollElementConfigArray){.capacity = capacity, .length = 0,
    .internalArray = (Clay_ScrollElementConfig *)Clay__Array_Allocate_Arena(capacity,
    sizeof(Clay_ScrollElementConfig), CLAY__ALIGNMENT(Clay_ScrollElementConfig), arena)};
00853 }
00854 Clay_ScrollElementConfig *Clay__ScrollElementConfigArray_Add(Clay__ScrollElementConfigArray *array,
    Clay_ScrollElementConfig item) {
00855     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00856         array->internalArray[array->length++] = item;
00857         return &array->internalArray[array->length - 1];
00858     }
00859     return &CLAY__SCROLL_ELEMENT_CONFIG_DEFAULT;
00860 }
00861 #pragma endregion
00862 // __GENERATED__ template
00863
00864 // __GENERATED__ template array_define_slice,array_allocate,array_add TYPE=Clay_String
    NAME=Clay__StringArray DEFAULT_VALUE=&CLAY__STRING_DEFAULT
00865 #pragma region generated
00866 CLAY__TYPEDEF(Clay__StringArraySlice, struct
00867 {
00868     int32_t length;
00869     Clay_String *internalArray;
00870 });
00871 Clay__StringArray Clay__StringArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
00872     return CLAY__INIT(Clay__StringArray){.capacity = capacity, .length = 0, .internalArray =
    (Clay_String *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_String), CLAY__ALIGNMENT(Clay_String),
    arena)};
00873 }
00874 Clay_String *Clay__StringArray_Add(Clay__StringArray *array, Clay_String item) {
00875     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00876         array->internalArray[array->length++] = item;
00877         return &array->internalArray[array->length - 1];
00878     }
00879     return &CLAY__STRING_DEFAULT;
00880 }
00881 #pragma endregion
00882 // __GENERATED__ template
00883
00884 CLAY__TYPEDEF(Clay__WrappedTextLine, struct {
00885     Clay_Dimensions dimensions;
00886     Clay_String line;
00887 });
00888
00889 Clay__WrappedTextLine CLAY__WRAPPED_TEXT_LINE_DEFAULT = CLAY__DEFAULT_STRUCT;
00890
00891 // __GENERATED__ template array_define,array_define_slice,array_allocate,array_add,array_get
    TYPE=Clay__WrappedTextLine NAME=Clay__WrappedTextLineArray
    DEFAULT_VALUE=&CLAY__WRAPPED_TEXT_LINE_DEFAULT
00892 #pragma region generated
00893 CLAY__TYPEDEF(Clay__WrappedTextLineArray, struct
00894 {
00895     int32_t capacity;
00896     int32_t length;
00897     Clay__WrappedTextLine *internalArray;
00898 });
00899 CLAY__TYPEDEF(Clay__WrappedTextLineArraySlice, struct
00900 {
00901     int32_t length;
00902     Clay__WrappedTextLine *internalArray;
00903 });
```

```
00904 Clay__WrappedTextLineArray Clay__WrappedTextLineArray_Allocate_Arena(int32_t capacity, Clay_Arena
      *arena) {
00905     return CLAY__INIT(Clay__WrappedTextLineArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay__WrappedTextLine *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay__WrappedTextLine),
      CLAY__ALIGNMENT(Clay__WrappedTextLine), arena)};
00906 }
00907 Clay__WrappedTextLine *Clay__WrappedTextLineArray_Add(Clay__WrappedTextLineArray *array,
      Clay__WrappedTextLine item) {
00908     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00909         array->internalArray[array->length++] = item;
00910         return &array->internalArray[array->length - 1];
00911     }
00912     return &CLAY__WRAPPED_TEXT_LINE_DEFAULT;
00913 }
00914 Clay__WrappedTextLine *Clay__WrappedTextLineArray_Get(Clay__WrappedTextLineArray *array, int32_t
      index) {
00915     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__WRAPPED_TEXT_LINE_DEFAULT;
00916 }
00917 #pragma endregion
00918 // __GENERATED__ template
00919
00920 CLAY__TYPEDEF(Clay__TextElementData, struct {
00921     Clay_String text;
00922     Clay_Dimensions preferredDimensions;
00923     int32_t elementIndex;
00924     Clay__WrappedTextLineArraySlice wrappedLines;
00925 });
00926
00927 Clay__TextElementData CLAY__TEXT_ELEMENT_DATA_DEFAULT = CLAY__DEFAULT_STRUCT;
00928
00929 // __GENERATED__ template array_define,array_allocate,array_get,array_add TYPE=Clay__TextElementData
      NAME=Clay__TextElementDataArray DEFAULT_VALUE=&CLAY__TEXT_ELEMENT_DATA_DEFAULT
00930 #pragma region generated
00931 CLAY__TYPEDEF(Clay__TextElementDataArray, struct
00932 {
00933     int32_t capacity;
00934     int32_t length;
00935     Clay__TextElementData *internalArray;
00936 });
00937 Clay__TextElementDataArray Clay__TextElementDataArray_Allocate_Arena(int32_t capacity, Clay_Arena
      *arena) {
00938     return CLAY__INIT(Clay__TextElementDataArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay__TextElementData *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay__TextElementData),
      CLAY__ALIGNMENT(Clay__TextElementData), arena)};
00939 }
00940 Clay__TextElementData *Clay__TextElementDataArray_Get(Clay__TextElementDataArray *array, int32_t
      index) {
00941     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__TEXT_ELEMENT_DATA_DEFAULT;
00942 }
00943 Clay__TextElementData *Clay__TextElementDataArray_Add(Clay__TextElementDataArray *array,
      Clay__TextElementData item) {
00944     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00945         array->internalArray[array->length++] = item;
00946         return &array->internalArray[array->length - 1];
00947     }
00948     return &CLAY__TEXT_ELEMENT_DATA_DEFAULT;
00949 }
00950 #pragma endregion
00951 // __GENERATED__ template
00952
00953 Clay_BorderElementConfig CLAY__BORDER_ELEMENT_CONFIG_DEFAULT = CLAY__DEFAULT_STRUCT;
00954
00955 // __GENERATED__ template array_define,array_allocate,array_add TYPE=Clay_BorderElementConfig
      NAME=Clay__BorderElementConfigArray DEFAULT_VALUE=&CLAY__BORDER_ELEMENT_CONFIG_DEFAULT
00956 #pragma region generated
00957 CLAY__TYPEDEF(Clay__BorderElementConfigArray, struct
00958 {
00959     int32_t capacity;
00960     int32_t length;
00961     Clay_BorderElementConfig *internalArray;
00962 });
00963 Clay__BorderElementConfigArray Clay__BorderElementConfigArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
00964     return CLAY__INIT(Clay__BorderElementConfigArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay_BorderElementConfig *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay_BorderElementConfig), CLAY__ALIGNMENT(Clay_BorderElementConfig), arena)};
00965 }
00966 Clay_BorderElementConfig *Clay__BorderElementConfigArray_Add(Clay__BorderElementConfigArray *array,
      Clay_BorderElementConfig item) {
00967     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
00968         array->internalArray[array->length++] = item;
00969         return &array->internalArray[array->length - 1];
00970     }
00971     return &CLAY__BORDER_ELEMENT_CONFIG_DEFAULT;
00972 }
```

```
00973 #pragma endregion
00974 // __GENERATED__ template
00975
00976 CLAY__TYPEDEF(Clay__LayoutElementChildren, struct {
00977     int32_t *elements;
00978     uint16_t length;
00979 });
00980
00981 CLAY__TYPEDEF(Clay_LayoutElement, struct {
00982     union {
00983         Clay__LayoutElementChildren children;
00984         Clay__TextElementData *textElementData;
00985     } childrenOrTextContent;
00986     Clay_Dimensions dimensions;
00987     Clay_Dimensions minDimensions;
00988     Clay_LayoutConfig *layoutConfig;
00989     Clay__ElementConfigArraySlice elementConfigs;
00990     uint32_t configsEnabled;
00991     uint32_t id;
00992 });
00993
00994 Clay_LayoutElement CLAY__LAYOUT_ELEMENT_DEFAULT = CLAY__DEFAULT_STRUCT;
00995
00996 // __GENERATED__ template array_define,array_allocate,array_add,array_get TYPE=Clay_LayoutElement
      NAME=Clay_LayoutElementArray DEFAULT_VALUE=&CLAY__LAYOUT_ELEMENT_DEFAULT
00997 #pragma region generated
00998 CLAY__TYPEDEF(Clay_LayoutElementArray, struct
00999 {
01000     int32_t capacity;
01001     int32_t length;
01002     Clay_LayoutElement *internalArray;
01003 });
01004 Clay_LayoutElementArray Clay_LayoutElementArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
01005     return CLAY__INIT(Clay_LayoutElementArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay_LayoutElement *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_LayoutElement),
      CLAY__ALIGNMENT(Clay_LayoutElement), arena)};
01006 }
01007 Clay_LayoutElement *Clay_LayoutElementArray_Add(Clay_LayoutElementArray *array, Clay_LayoutElement
      item) {
01008     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01009         array->internalArray[array->length++] = item;
01010         return &array->internalArray[array->length - 1];
01011     }
01012     return &CLAY__LAYOUT_ELEMENT_DEFAULT;
01013 }
01014 Clay_LayoutElement *Clay_LayoutElementArray_Get(Clay_LayoutElementArray *array, int32_t index) {
01015     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__LAYOUT_ELEMENT_DEFAULT;
01016 }
01017 #pragma endregion
01018 // __GENERATED__ template
01019
01020 // __GENERATED__ template
      array_define,array_allocate_pointer,array_add,array_get_value,array_remove_swapback
      TYPE=Clay_LayoutElement* NAME=Clay__LayoutElementPointerArray DEFAULT_VALUE=CLAY__NULL
01021 #pragma region generated
01022 CLAY__TYPEDEF(Clay__LayoutElementPointerArray, struct
01023 {
01024     int32_t capacity;
01025     int32_t length;
01026     Clay_LayoutElement* *internalArray;
01027 });
01028 Clay__LayoutElementPointerArray Clay__LayoutElementPointerArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
01029     return CLAY__INIT(Clay__LayoutElementPointerArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay_LayoutElement* *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay_LayoutElement*), CLAY__POINTER_ALIGNMENT, arena)};
01030 }
01031 Clay_LayoutElement* *Clay__LayoutElementPointerArray_Add(Clay__LayoutElementPointerArray *array,
      Clay_LayoutElement* item) {
01032     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01033         array->internalArray[array->length++] = item;
01034         return &array->internalArray[array->length - 1];
01035     }
01036     return CLAY__NULL;
01037 }
01038 Clay_LayoutElement* Clay__LayoutElementPointerArray_Get(Clay__LayoutElementPointerArray *array,
      int32_t index) {
01039     return Clay__Array_RangeCheck(index, array->length) ? array->internalArray[index] : CLAY__NULL;
01040 }
01041 Clay_LayoutElement* Clay__LayoutElementPointerArray_RemoveSwapback(Clay__LayoutElementPointerArray
      *array, int32_t index) {
01042     if (Clay__Array_RangeCheck(index, array->length)) {
01043         array->length--;
01044         Clay_LayoutElement* removed = array->internalArray[index];
01045         array->internalArray[index] = array->internalArray[array->length];
01046         return removed;
```

```
01047     }
01048     return CLAY__NULL;
01049 }
01050 #pragma endregion
01051 // __GENERATED__ template
01052
01053 Clay_RenderCommand CLAY__RENDER_COMMAND_DEFAULT = CLAY__DEFAULT_STRUCT;
01054
01055 // __GENERATED__ template array_allocate,array_add,array_get TYPE=Clay_RenderCommand
      NAME=Clay_RenderCommandArray DEFAULT_VALUE=&CLAY__RENDER_COMMAND_DEFAULT
01056 #pragma region generated
01057 Clay_RenderCommandArray Clay_RenderCommandArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
01058     return CLAY__INIT(Clay_RenderCommandArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay_RenderCommand *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay_RenderCommand),
      CLAY__ALIGNMENT(Clay_RenderCommand), arena)};
01059 }
01060 Clay_RenderCommand *Clay_RenderCommandArray_Add(Clay_RenderCommandArray *array, Clay_RenderCommand
      item) {
01061     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01062         array->internalArray[array->length++] = item;
01063         return &array->internalArray[array->length - 1];
01064     }
01065     return &CLAY__RENDER_COMMAND_DEFAULT;
01066 }
01067 Clay_RenderCommand *Clay_RenderCommandArray_Get(Clay_RenderCommandArray *array, int32_t index) {
01068     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__RENDER_COMMAND_DEFAULT;
01069 }
01070 #pragma endregion
01071 // __GENERATED__ template
01072
01073 CLAY__TYPEDEF(Clay__ScrollContainerDataInternal, struct {
01074     Clay_LayoutElement *layoutElement;
01075     Clay_BoundingBox boundingBox;
01076     Clay_Dimensions contentSize;
01077     Clay_Vector2 scrollOrigin;
01078     Clay_Vector2 pointerOrigin;
01079     Clay_Vector2 scrollMomentum;
01080     Clay_Vector2 scrollPosition;
01081     Clay_Vector2 previousDelta;
01082     float momentumTime;
01083     uint32_t elementId;
01084     bool openThisFrame;
01085     bool pointerScrollActive;
01086 });
01087
01088 Clay__ScrollContainerDataInternal CLAY__SCROLL_CONTAINER_DEFAULT = CLAY__DEFAULT_STRUCT;
01089
01090 // __GENERATED__ template array_define,array_allocate,array_add,array_get
      TYPE=Clay__ScrollContainerDataInternal NAME=Clay__ScrollContainerDataInternalArray
      DEFAULT_VALUE=&CLAY__SCROLL_CONTAINER_DEFAULT
01091 #pragma region generated
01092 CLAY__TYPEDEF(Clay__ScrollContainerDataInternalArray, struct
01093 {
01094     int32_t capacity;
01095     int32_t length;
01096     Clay__ScrollContainerDataInternal *internalArray;
01097 });
01098 Clay__ScrollContainerDataInternalArray Clay__ScrollContainerDataInternalArray_Allocate_Arena(int32_t
      capacity, Clay_Arena *arena) {
01099     return CLAY__INIT(Clay__ScrollContainerDataInternalArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay__ScrollContainerDataInternal *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay__ScrollContainerDataInternal), CLAY__ALIGNMENT(Clay__ScrollContainerDataInternal),
      arena)};
01100 }
01101 Clay__ScrollContainerDataInternal
      *Clay__ScrollContainerDataInternalArray_Add(Clay__ScrollContainerDataInternalArray *array,
      Clay__ScrollContainerDataInternal item) {
01102     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01103         array->internalArray[array->length++] = item;
01104         return &array->internalArray[array->length - 1];
01105     }
01106     return &CLAY__SCROLL_CONTAINER_DEFAULT;
01107 }
01108 Clay__ScrollContainerDataInternal
      *Clay__ScrollContainerDataInternalArray_Get(Clay__ScrollContainerDataInternalArray *array, int32_t
      index) {
01109     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__SCROLL_CONTAINER_DEFAULT;
01110 }
01111 #pragma endregion
01112 // __GENERATED__ template
01113
01114 // __GENERATED__ template array_remove_swapback TYPE=Clay__ScrollContainerDataInternal
      NAME=Clay__ScrollContainerDataInternalArray DEFAULT_VALUE=CLAY__SCROLL_CONTAINER_DEFAULT
01115 #pragma region generated
01116 Clay__ScrollContainerDataInternal
```

```
    Clay__ScrollContainerDataInternalArray_RemoveSwapback(Clay__ScrollContainerDataInternalArray *array,
    int32_t index) {
01117     if (Clay__Array_RangeCheck(index, array->length)) {
01118         array->length--;
01119         Clay__ScrollContainerDataInternal removed = array->internalArray[index];
01120         array->internalArray[index] = array->internalArray[array->length];
01121         return removed;
01122     }
01123     return CLAY__SCROLL_CONTAINER_DEFAULT;
01124 }
01125 #pragma endregion
01126 // __GENERATED__ template
01127
01128 CLAY__TYPEDEF(Clay__DebugElementData, struct {
01129     bool collision;
01130     bool collapsed;
01131 });
01132
01133 Clay__DebugElementData CLAY__DEBUG_ELEMENT_DATA_DEFAULT = CLAY__DEFAULT_STRUCT;
01134
01135 // __GENERATED__ template array_define,array_allocate,array_add,array_get TYPE=Clay__DebugElementData
    NAME=Clay__DebugElementDataArray DEFAULT_VALUE=&CLAY__DEBUG_ELEMENT_DATA_DEFAULT
01136 #pragma region generated
01137 CLAY__TYPEDEF(Clay__DebugElementDataArray, struct
01138 {
01139     int32_t capacity;
01140     int32_t length;
01141     Clay__DebugElementData *internalArray;
01142 });
01143 Clay__DebugElementDataArray Clay__DebugElementDataArray_Allocate_Arena(int32_t capacity, Clay_Arena
    *arena) {
01144     return CLAY__INIT(Clay__DebugElementDataArray){.capacity = capacity, .length = 0, .internalArray =
    (Clay__DebugElementData *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay__DebugElementData),
    CLAY__ALIGNMENT(Clay__DebugElementData), arena)};
01145 }
01146 Clay__DebugElementData *Clay__DebugElementDataArray_Add(Clay__DebugElementDataArray *array,
    Clay__DebugElementData item) {
01147     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01148         array->internalArray[array->length++] = item;
01149         return &array->internalArray[array->length - 1];
01150     }
01151     return &CLAY__DEBUG_ELEMENT_DATA_DEFAULT;
01152 }
01153 Clay__DebugElementData *Clay__DebugElementDataArray_Get(Clay__DebugElementDataArray *array, int32_t
    index) {
01154     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
    &CLAY__DEBUG_ELEMENT_DATA_DEFAULT;
01155 }
01156 #pragma endregion
01157 // __GENERATED__ template
01158
01159 CLAY__TYPEDEF(Clay_LayoutElementHashMapItem, struct { // todo get this struct into a single cache line
01160     Clay_BoundingBox boundingBox;
01161     Clay_ElementId elementId;
01162     Clay_LayoutElement* layoutElement;
01163     void (*onHoverFunction)(Clay_ElementId elementId, Clay_PointerData pointerInfo, intptr_t
    userData);
01164     intptr_t hoverFunctionUserData;
01165     int32_t nextIndex;
01166     uint32_t generation;
01167     Clay__DebugElementData *debugData;
01168 });
01169
01170 Clay_LayoutElementHashMapItem CLAY__LAYOUT_ELEMENT_HASH_MAP_ITEM_DEFAULT = { .layoutElement =
    &CLAY__LAYOUT_ELEMENT_DEFAULT };
01171
01172 // __GENERATED__ template array_define,array_allocate,array_get,array_add
    TYPE=Clay_LayoutElementHashMapItem NAME=Clay__LayoutElementHashMapItemArray
    DEFAULT_VALUE=&CLAY__LAYOUT_ELEMENT_HASH_MAP_ITEM_DEFAULT
01173 #pragma region generated
01174 CLAY__TYPEDEF(Clay__LayoutElementHashMapItemArray, struct
01175 {
01176     int32_t capacity;
01177     int32_t length;
01178     Clay_LayoutElementHashMapItem *internalArray;
01179 });
01180 Clay__LayoutElementHashMapItemArray Clay__LayoutElementHashMapItemArray_Allocate_Arena(int32_t
    capacity, Clay_Arena *arena) {
01181     return CLAY__INIT(Clay__LayoutElementHashMapItemArray){.capacity = capacity, .length = 0,
    .internalArray = (Clay_LayoutElementHashMapItem *)Clay__Array_Allocate_Arena(capacity,
    sizeof(Clay_LayoutElementHashMapItem), CLAY__ALIGNMENT(Clay_LayoutElementHashMapItem), arena)};
01182 }
01183 Clay_LayoutElementHashMapItem
    *Clay__LayoutElementHashMapItemArray_Get(Clay__LayoutElementHashMapItemArray *array, int32_t index) {
01184     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
    &CLAY__LAYOUT_ELEMENT_HASH_MAP_ITEM_DEFAULT;
01185 }
```

```
01186 Clay_LayoutElementHashMapItem
      *Clay__LayoutElementHashMapItemArray_Add(Clay__LayoutElementHashMapItemArray *array,
      Clay_LayoutElementHashMapItem item) {
01187     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01188         array->internalArray[array->length++] = item;
01189         return &array->internalArray[array->length - 1];
01190     }
01191     return &CLAY__LAYOUT_ELEMENT_HASH_MAP_ITEM_DEFAULT;
01192 }
01193 #pragma endregion
01194 // __GENERATED__ template
01195
01196 CLAY__TYPEDEF(Clay__MeasuredWord, struct {
01197     int32_t startOffset;
01198     int32_t length;
01199     float width;
01200     int32_t next;
01201 });
01202
01203 Clay__MeasuredWord CLAY__MEASURED_WORD_DEFAULT = { .next = -1 };
01204
01205 // __GENERATED__ template array_define,array_allocate,array_get,array_set,array_add
      TYPE=Clay__MeasuredWord NAME=Clay__MeasuredWordArray DEFAULT_VALUE=&CLAY__MEASURED_WORD_DEFAULT
01206 #pragma region generated
01207 CLAY__TYPEDEF(Clay__MeasuredWordArray, struct
01208 {
01209     int32_t capacity;
01210     int32_t length;
01211     Clay__MeasuredWord *internalArray;
01212 });
01213 Clay__MeasuredWordArray Clay__MeasuredWordArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
01214     return CLAY__INIT(Clay__MeasuredWordArray){.capacity = capacity, .length = 0, .internalArray =
      (Clay__MeasuredWord *)Clay__Array_Allocate_Arena(capacity, sizeof(Clay__MeasuredWord),
      CLAY__ALIGNMENT(Clay__MeasuredWord), arena)};
01215 }
01216 Clay__MeasuredWord *Clay__MeasuredWordArray_Get(Clay__MeasuredWordArray *array, int32_t index) {
01217     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__MEASURED_WORD_DEFAULT;
01218 }
01219 void Clay__MeasuredWordArray_Set(Clay__MeasuredWordArray *array, int32_t index, Clay__MeasuredWord
      value) {
01220     if (Clay__Array_RangeCheck(index, array->capacity)) {
01221         array->internalArray[index] = value;
01222         array->length = index < array->length ? array->length : index + 1;
01223     }
01224 }
01225 Clay__MeasuredWord *Clay__MeasuredWordArray_Add(Clay__MeasuredWordArray *array, Clay__MeasuredWord
      item) {
01226     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01227         array->internalArray[array->length++] = item;
01228         return &array->internalArray[array->length - 1];
01229     }
01230     return &CLAY__MEASURED_WORD_DEFAULT;
01231 }
01232 #pragma endregion
01233 // __GENERATED__ template
01234
01235 CLAY__TYPEDEF(Clay__MeasureTextCacheItem, struct {
01236     Clay_Dimensions unwrappedDimensions;
01237     int32_t measuredWordsStartIndex;
01238     bool containsNewlines;
01239     // Hash map data
01240     uint32_t id;
01241     int32_t nextIndex;
01242     uint32_t generation;
01243 });
01244
01245 Clay__MeasureTextCacheItem CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT = { .measuredWordsStartIndex = -1 };
01246
01247 // __GENERATED__ template array_define,array_allocate,array_get,array_add,array_set
      TYPE=Clay__MeasureTextCacheItem NAME=Clay__MeasureTextCacheItemArray
      DEFAULT_VALUE=&CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT
01248 #pragma region generated
01249 CLAY__TYPEDEF(Clay__MeasureTextCacheItemArray, struct
01250 {
01251     int32_t capacity;
01252     int32_t length;
01253     Clay__MeasureTextCacheItem *internalArray;
01254 });
01255 Clay__MeasureTextCacheItemArray Clay__MeasureTextCacheItemArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
01256     return CLAY__INIT(Clay__MeasureTextCacheItemArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay__MeasureTextCacheItem *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay__MeasureTextCacheItem), CLAY__ALIGNMENT(Clay__MeasureTextCacheItem), arena)};
01257 }
01258 Clay__MeasureTextCacheItem *Clay__MeasureTextCacheItemArray_Get(Clay__MeasureTextCacheItemArray
      *array, int32_t index) {
```

```
01259     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT;
01260 }
01261 Clay__MeasureTextCacheItem *Clay__MeasureTextCacheItemArray_Add(Clay__MeasureTextCacheItemArray
      *array, Clay__MeasureTextCacheItem item) {
01262     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01263         array->internalArray[array->length++] = item;
01264         return &array->internalArray[array->length - 1];
01265     }
01266     return &CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT;
01267 }
01268 void Clay__MeasureTextCacheItemArray_Set(Clay__MeasureTextCacheItemArray *array, int32_t index,
      Clay__MeasureTextCacheItem value) {
01269     if (Clay__Array_RangeCheck(index, array->capacity)) {
01270         array->internalArray[index] = value;
01271         array->length = index < array->length ? array->length : index + 1;
01272     }
01273 }
01274 #pragma endregion
01275 // __GENERATED__ template
01276
01277 // __GENERATED__ template
      array_define,array_allocate,array_get_value,array_add_value,array_set,array_remove_swapback
      TYPE=int32_t NAME=Clay__int32_tArray DEFAULT_VALUE=-1
01278 #pragma region generated
01279 CLAY__TYPEDEF(Clay__int32_tArray, struct
01280 {
01281     int32_t capacity;
01282     int32_t length;
01283     int32_t *internalArray;
01284 });
01285 Clay__int32_tArray Clay__int32_tArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
01286     return CLAY__INIT(Clay__int32_tArray){.capacity = capacity, .length = 0, .internalArray = (int32_t
      *)Clay__Array_Allocate_Arena(capacity, sizeof(int32_t), CLAY__ALIGNMENT(int32_t), arena)};
01287 }
01288 int32_t Clay__int32_tArray_Get(Clay__int32_tArray *array, int32_t index) {
01289     return Clay__Array_RangeCheck(index, array->length) ? array->internalArray[index] : -1;
01290 }
01291 void Clay__int32_tArray_Add(Clay__int32_tArray *array, int32_t item) {
01292     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01293         array->internalArray[array->length++] = item;
01294     }
01295 }
01296 void Clay__int32_tArray_Set(Clay__int32_tArray *array, int32_t index, int32_t value) {
01297     if (Clay__Array_RangeCheck(index, array->capacity)) {
01298         array->internalArray[index] = value;
01299         array->length = index < array->length ? array->length : index + 1;
01300     }
01301 }
01302 int32_t Clay__int32_tArray_RemoveSwapback(Clay__int32_tArray *array, int32_t index) {
01303     if (Clay__Array_RangeCheck(index, array->length)) {
01304         array->length--;
01305         int32_t removed = array->internalArray[index];
01306         array->internalArray[index] = array->internalArray[array->length];
01307         return removed;
01308     }
01309     return -1;
01310 }
01311 #pragma endregion
01312 // __GENERATED__ template
01313
01314 CLAY__TYPEDEF(Clay__LayoutElementTreeNode, struct {
01315     Clay_LayoutElement *layoutElement;
01316     Clay_Vector2 position;
01317     Clay_Vector2 nextChildOffset;
01318 });
01319
01320 Clay__LayoutElementTreeNode CLAY__LAYOUT_ELEMENT_TREE_NODE_DEFAULT = CLAY__DEFAULT_STRUCT;
01321
01322 // __GENERATED__ template array_define,array_allocate,array_add,array_get
      TYPE=Clay__LayoutElementTreeNode NAME=Clay__LayoutElementTreeNodeArray
      DEFAULT_VALUE=&CLAY__LAYOUT_ELEMENT_TREE_NODE_DEFAULT
01323 #pragma region generated
01324 CLAY__TYPEDEF(Clay__LayoutElementTreeNodeArray, struct
01325 {
01326     int32_t capacity;
01327     int32_t length;
01328     Clay__LayoutElementTreeNode *internalArray;
01329 });
01330 Clay__LayoutElementTreeNodeArray Clay__LayoutElementTreeNodeArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
01331     return CLAY__INIT(Clay__LayoutElementTreeNodeArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay__LayoutElementTreeNode *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay__LayoutElementTreeNode), CLAY__ALIGNMENT(Clay__LayoutElementTreeNode), arena)};
01332 }
01333 Clay__LayoutElementTreeNode *Clay__LayoutElementTreeNodeArray_Add(Clay__LayoutElementTreeNodeArray
      *array, Clay__LayoutElementTreeNode item) {
```

```
01334     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01335         array->internalArray[array->length++] = item;
01336         return &array->internalArray[array->length - 1];
01337     }
01338     return &CLAY__LAYOUT_ELEMENT_TREE_NODE_DEFAULT;
01339 }
01340 Clay__LayoutElementTreeNode *Clay__LayoutElementTreeNodeArray_Get(Clay__LayoutElementTreeNodeArray
      *array, int32_t index) {
01341     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__LAYOUT_ELEMENT_TREE_NODE_DEFAULT;
01342 }
01343 #pragma endregion
01344 // __GENERATED__ template
01345
01346 CLAY__TYPEDEF(Clay__LayoutElementTreeRoot, struct {
01347     int32_t layoutElementIndex;
01348     uint32_t parentId; // This can be zero in the case of the root layout tree
01349     uint32_t clipElementId; // This can be zero if there is no clip element
01350     int32_t zIndex;
01351     Clay_Vector2 pointerOffset; // Only used when scroll containers are managed externally
01352 });
01353
01354 Clay__LayoutElementTreeRoot CLAY__LAYOUT_ELEMENT_TREE_ROOT_DEFAULT = CLAY__DEFAULT_STRUCT;
01355
01356 // __GENERATED__ template array_define,array_allocate,array_add,array_get,array_set
      TYPE=Clay__LayoutElementTreeRoot NAME=Clay__LayoutElementTreeRootArray
      DEFAULT_VALUE=&CLAY__LAYOUT_ELEMENT_TREE_ROOT_DEFAULT
01357 #pragma region generated
01358 CLAY__TYPEDEF(Clay__LayoutElementTreeRootArray, struct
01359 {
01360     int32_t capacity;
01361     int32_t length;
01362     Clay__LayoutElementTreeRoot *internalArray;
01363 });
01364 Clay__LayoutElementTreeRootArray Clay__LayoutElementTreeRootArray_Allocate_Arena(int32_t capacity,
      Clay_Arena *arena) {
01365     return CLAY__INIT(Clay__LayoutElementTreeRootArray){.capacity = capacity, .length = 0,
      .internalArray = (Clay__LayoutElementTreeRoot *)Clay__Array_Allocate_Arena(capacity,
      sizeof(Clay__LayoutElementTreeRoot), CLAY__ALIGNMENT(Clay__LayoutElementTreeRoot), arena)};
01366 }
01367 Clay__LayoutElementTreeRoot *Clay__LayoutElementTreeRootArray_Add(Clay__LayoutElementTreeRootArray
      *array, Clay__LayoutElementTreeRoot item) {
01368     if (Clay__Array_AddCapacityCheck(array->length, array->capacity)) {
01369         array->internalArray[array->length++] = item;
01370         return &array->internalArray[array->length - 1];
01371     }
01372     return &CLAY__LAYOUT_ELEMENT_TREE_ROOT_DEFAULT;
01373 }
01374 Clay__LayoutElementTreeRoot *Clay__LayoutElementTreeRootArray_Get(Clay__LayoutElementTreeRootArray
      *array, int32_t index) {
01375     return Clay__Array_RangeCheck(index, array->length) ? &array->internalArray[index] :
      &CLAY__LAYOUT_ELEMENT_TREE_ROOT_DEFAULT;
01376 }
01377 void Clay__LayoutElementTreeRootArray_Set(Clay__LayoutElementTreeRootArray *array, int32_t index,
      Clay__LayoutElementTreeRoot value) {
01378     if (Clay__Array_RangeCheck(index, array->capacity)) {
01379         array->internalArray[index] = value;
01380         array->length = index < array->length ? array->length : index + 1;
01381     }
01382 }
01383 #pragma endregion
01384 // __GENERATED__ template
01385
01386 // __GENERATED__ template array_define,array_allocate TYPE=uint8_t NAME=Clay__CharArray
      DEFAULT_VALUE=0
01387 #pragma region generated
01388 CLAY__TYPEDEF(Clay__CharArray, struct
01389 {
01390     int32_t capacity;
01391     int32_t length;
01392     uint8_t *internalArray;
01393 });
01394 Clay__CharArray Clay__CharArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
01395     return CLAY__INIT(Clay__CharArray){.capacity = capacity, .length = 0, .internalArray = (uint8_t
      *)Clay__Array_Allocate_Arena(capacity, sizeof(uint8_t), CLAY__ALIGNMENT(uint8_t), arena)};
01396 }
01397 #pragma endregion
01398 // __GENERATED__ template
01399
01400 struct Clay_Context {
01401     int32_t maxElementCount;
01402     int32_t maxMeasureTextCacheWordCount;
01403     bool warningsEnabled;
01404     Clay_ErrorHandler errorHandler;
01405     Clay_BooleanWarnings booleanWarnings;
01406     Clay__WarningArray warnings;
01407
```

```
01408     Clay_PointerData pointerInfo;
01409     Clay_Dimensions layoutDimensions;
01410     Clay_ElementId dynamicElementIndexBaseHash;
01411     uint32_t dynamicElementIndex;
01412     bool debugModeEnabled;
01413     bool disableCulling;
01414     bool externalScrollHandlingEnabled;
01415     uint32_t debugSelectedElementId;
01416     uint32_t generation;
01417     uintptr_t arenaResetOffset;
01418     uintptr_t mesureTextUserData;
01419     uintptr_t queryScrollOffsetUserData;
01420     Clay_Arena internalArena;
01421     // Layout Elements / Render Commands
01422     Clay_LayoutElementArray layoutElements;
01423     Clay_RenderCommandArray renderCommands;
01424     Clay__int32_tArray openLayoutElementStack;
01425     Clay__int32_tArray layoutElementChildren;
01426     Clay__int32_tArray layoutElementChildrenBuffer;
01427     Clay__TextElementDataArray textElementData;
01428     Clay__LayoutElementPointerArray imageElementPointers;
01429     Clay__int32_tArray reusableElementIndexBuffer;
01430     Clay__int32_tArray layoutElementClipElementIds;
01431     // Configs
01432     Clay__LayoutConfigArray layoutConfigs;
01433     Clay__ElementConfigArray elementConfigBuffer;
01434     Clay__ElementConfigArray elementConfigs;
01435     Clay__RectangleElementConfigArray rectangleElementConfigs;
01436     Clay__TextElementConfigArray textElementConfigs;
01437     Clay__ImageElementConfigArray imageElementConfigs;
01438     Clay__FloatingElementConfigArray floatingElementConfigs;
01439     Clay__ScrollElementConfigArray scrollElementConfigs;
01440     Clay__CustomElementConfigArray customElementConfigs;
01441     Clay__BorderElementConfigArray borderElementConfigs;
01442     // Misc Data Structures
01443     Clay__StringArray layoutElementIdStrings;
01444     Clay__WrappedTextLineArray wrappedTextLines;
01445     Clay__LayoutElementTreeNodeArray layoutElementTreeNodeArray1;
01446     Clay__LayoutElementTreeRootArray layoutElementTreeRoots;
01447     Clay__LayoutElementHashMapItemArray layoutElementsHashMapInternal;
01448     Clay__int32_tArray layoutElementsHashMap;
01449     Clay__MeasureTextCacheItemArray measureTextHashMapInternal;
01450     Clay__int32_tArray measureTextHashMapInternalFreeList;
01451     Clay__int32_tArray measureTextHashMap;
01452     Clay__MeasuredWordArray measuredWords;
01453     Clay__int32_tArray measuredWordsFreeList;
01454     Clay__int32_tArray openClipElementStack;
01455     Clay__ElementIdArray pointerOverIds;
01456     Clay__ScrollContainerDataInternalArray scrollContainerDatas;
01457     Clay__BoolArray treeNodeVisited;
01458     Clay__CharArray dynamicStringData;
01459     Clay__DebugElementDataArray debugElementData;
01460 };
01461
01462 struct Clay__AlignClay_Context {
01463     char c;
01464     Clay_Context x;
01465 };
01466 typedef struct {
01467     Clay_Context wrapped;
01468 } Clay__Clay_ContextWrapper;
01469
01470 Clay_Context* Clay__Context_Allocate_Arena(Clay_Arena *arena) {
01471     uint32_t alignment = CLAY__ALIGNMENT(Clay_Context);
01472     size_t totalSizeBytes = sizeof(Clay_Context);
01473     uintptr_t nextAllocAddress = arena->nextAllocation + (uintptr_t)arena->memory;
01474     uintptr_t arenaOffsetAligned = nextAllocAddress + (alignment - (nextAllocAddress & alignment));
01475     arenaOffsetAligned -= (uintptr_t)arena->memory;
01476     if (arenaOffsetAligned + totalSizeBytes > arena->capacity)
01477     {
01478         return NULL;
01479     }
01480     arena->nextAllocation = arenaOffsetAligned + totalSizeBytes;
01481     return (Clay_Context*)((uintptr_t)arena->memory + arenaOffsetAligned);
01482 }
01483
01484 Clay_String Clay__WriteStringToCharBuffer(Clay__CharArray *buffer, Clay_String string) {
01485     for (int32_t i = 0; i < string.length; i++) {
01486         buffer->internalArray[buffer->length + i] = string.chars[i];
01487     }
01488     buffer->length += string.length;
01489     return CLAY__INIT(Clay_String) { .length = string.length, .chars = (const char
      *)(buffer->internalArray + buffer->length - string.length) };
01490 }
01491
01492 #ifdef CLAY_WASM
01493     __attribute__((import_module("clay"), import_name("measureTextFunction"))) Clay_Dimensions
```

```
    Clay__MeasureText(Clay_StringSlice text, Clay_TextElementConfig *config, uintptr_t userData);
01494     __attribute__((import_module("clay"), import_name("queryScrollOffsetFunction"))) Clay_Vector2
    Clay__QueryScrollOffset(uint32_t elementId, uintptr_t userData);
01495 #else
01496     Clay_Dimensions (*Clay__MeasureText)(Clay_StringSlice text, Clay_TextElementConfig *config,
    uintptr_t userData);
01497     Clay_Vector2 (*Clay__QueryScrollOffset)(uint32_t elementId, uintptr_t userData);
01498 #endif
01499
01500 Clay_LayoutElement* Clay__GetOpenLayoutElement(void) {
01501     Clay_Context* context = Clay_GetCurrentContext();
01502     return Clay_LayoutElementArray_Get(&context->layoutElements,
    Clay__int32_tArray_Get(&context->openLayoutElementStack, context->openLayoutElementStack.length - 1));
01503 }
01504
01505 uint32_t Clay__GetParentElementId(void) {
01506     Clay_Context* context = Clay_GetCurrentContext();
01507     return Clay_LayoutElementArray_Get(&context->layoutElements,
    Clay__int32_tArray_Get(&context->openLayoutElementStack, context->openLayoutElementStack.length -
    2))->id;
01508 }
01509
01510 bool Clay__ElementHasConfig(Clay_LayoutElement *element, Clay__ElementConfigType type) {
01511     return (element->configsEnabled & type);
01512 }
01513
01514 Clay_ElementConfigUnion Clay__FindElementConfigWithType(Clay_LayoutElement *element,
    Clay__ElementConfigType type) {
01515     for (int32_t i = 0; i < element->elementConfigs.length; i++) {
01516         Clay_ElementConfig *config = Clay__ElementConfigArraySlice_Get(&element->elementConfigs, i);
01517         if (config->type == type) {
01518             return config->config;
01519         }
01520     }
01521     return CLAY__INIT(Clay_ElementConfigUnion) { NULL };
01522 }
01523
01524 Clay_ElementId Clay__HashNumber(const uint32_t offset, const uint32_t seed) {
01525     uint32_t hash = seed;
01526     hash += (offset + 48);
01527     hash += (hash << 10);
01528     hash ^= (hash >> 6);
01529
01530     hash += (hash << 3);
01531     hash ^= (hash >> 11);
01532     hash += (hash << 15);
01533     return CLAY__INIT(Clay_ElementId) { .id = hash + 1, .offset = offset, .baseId = seed, .stringId =
    CLAY__STRING_DEFAULT }; // Reserve the hash result of zero as "null id"
01534 }
01535
01536 Clay_ElementId Clay__HashString(Clay_String key, const uint32_t offset, const uint32_t seed) {
01537     uint32_t hash = 0;
01538     uint32_t base = seed;
01539
01540     for (int32_t i = 0; i < key.length; i++) {
01541         base += key.chars[i];
01542         base += (base << 10);
01543         base ^= (base >> 6);
01544     }
01545     hash = base;
01546     hash += offset;
01547     hash += (hash << 10);
01548     hash ^= (hash >> 6);
01549
01550     hash += (hash << 3);
01551     base += (base << 3);
01552     hash ^= (hash >> 11);
01553     base ^= (base >> 11);
01554     hash += (hash << 15);
01555     base += (base << 15);
01556     return CLAY__INIT(Clay_ElementId) { .id = hash + 1, .offset = offset, .baseId = base + 1,
    .stringId = key }; // Reserve the hash result of zero as "null id"
01557 }
01558
01559 Clay_ElementId Clay__Rehash(Clay_ElementId elementId, uint32_t number) {
01560     uint32_t id = elementId.baseId;
01561     id += number;
01562     id += (id << 10);
01563     id ^= (id >> 6);
01564
01565     id += (id << 3);
01566     id ^= (id >> 11);
01567     id += (id << 15);
01568     return CLAY__INIT(Clay_ElementId) { .id = id, .offset = number, .baseId = elementId.baseId,
    .stringId = elementId.stringId };
01569 }
01570
```

```
01571 uint32_t Clay__RehashWithNumber(uint32_t id, uint32_t number) {
01572     id += number;
01573     id += (id « 10);
01574     id ^= (id » 6);
01575
01576     id += (id « 3);
01577     id ^= (id » 11);
01578     id += (id « 15);
01579     return id;
01580 }
01581
01582 uint32_t Clay__HashTextWithConfig(Clay_String *text, Clay_TextElementConfig *config) {
01583     uint32_t hash = 0;
01584     uintptr_t pointerAsNumber = (uintptr_t)text->chars;
01585
01586     hash += pointerAsNumber;
01587     hash += (hash « 10);
01588     hash ^= (hash » 6);
01589
01590     hash += text->length;
01591     hash += (hash « 10);
01592     hash ^= (hash » 6);
01593
01594     hash += config->fontId;
01595     hash += (hash « 10);
01596     hash ^= (hash » 6);
01597
01598     hash += config->fontSize;
01599     hash += (hash « 10);
01600     hash ^= (hash » 6);
01601
01602     hash += config->lineHeight;
01603     hash += (hash « 10);
01604     hash ^= (hash » 6);
01605
01606     hash += config->letterSpacing;
01607     hash += (hash « 10);
01608     hash ^= (hash » 6);
01609
01610     hash += config->wrapMode;
01611     hash += (hash « 10);
01612     hash ^= (hash » 6);
01613
01614     hash += (hash « 3);
01615     hash ^= (hash » 11);
01616     hash += (hash « 15);
01617     return hash + 1; // Reserve the hash result of zero as "null id"
01618 }
01619
01620 Clay__MeasuredWord *Clay__AddMeasuredWord(Clay__MeasuredWord word, Clay__MeasuredWord *previousWord) {
01621     Clay_Context* context = Clay_GetCurrentContext();
01622     if (context->measuredWordsFreeList.length > 0) {
01623         uint32_t newItemIndex = Clay__int32_tArray_Get(&context->measuredWordsFreeList,
      (int)context->measuredWordsFreeList.length - 1);
01624         context->measuredWordsFreeList.length--;
01625         Clay__MeasuredWordArray_Set(&context->measuredWords, (int)newItemIndex, word);
01626         previousWord->next = (int32_t)newItemIndex;
01627         return Clay__MeasuredWordArray_Get(&context->measuredWords, (int)newItemIndex);
01628     } else {
01629         previousWord->next = (int32_t)context->measuredWords.length;
01630         return Clay__MeasuredWordArray_Add(&context->measuredWords, word);
01631     }
01632 }
01633
01634 Clay__MeasureTextCacheItem *Clay__MeasureTextCached(Clay_String *text, Clay_TextElementConfig *config)
      {
01635     Clay_Context* context = Clay_GetCurrentContext();
01636     #ifndef CLAY_WASM
01637     if (!Clay__MeasureText) {
01638         if (!context->booleanWarnings.textMeasurementFunctionNotSet) {
01639             context->booleanWarnings.textMeasurementFunctionNotSet = true;
01640             context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
01641                     .errorType = CLAY_ERROR_TYPE_TEXT_MEASUREMENT_FUNCTION_NOT_PROVIDED,
01642                     .errorText = CLAY_STRING("Clay's internal MeasureText function is null. You may
      have forgotten to call Clay_SetMeasureTextFunction(), or passed a NULL function pointer by mistake."),
01643                     .userData = context->errorHandler.userData });
01644         }
01645         return &CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT;
01646     }
01647     #endif
01648     uint32_t id = Clay__HashTextWithConfig(text, config);
01649     uint32_t hashBucket = id % (context->maxMeasureTextCacheWordCount / 32);
01650     int32_t elementIndexPrevious = 0;
01651     int32_t elementIndex = context->measureTextHashMap.internalArray[hashBucket];
01652     while (elementIndex != 0) {
01653         Clay__MeasureTextCacheItem *hashEntry =
      Clay__MeasureTextCacheItemArray_Get(&context->measureTextHashMapInternal, elementIndex);
```

```
01654         if (hashEntry->id == id) {
01655             hashEntry->generation = context->generation;
01656             return hashEntry;
01657         }
01658         // This element hasn't been seen in a few frames, delete the hash map item
01659         if (context->generation - hashEntry->generation > 2) {
01660             // Add all the measured words that were included in this measurement to the freelist
01661             int32_t nextWordIndex = hashEntry->measuredWordsStartIndex;
01662             while (nextWordIndex != -1) {
01663                 Clay__MeasuredWord *measuredWord =
      Clay__MeasuredWordArray_Get(&context->measuredWords, nextWordIndex);
01664                 Clay__int32_tArray_Add(&context->measuredWordsFreeList, nextWordIndex);
01665                 nextWordIndex = measuredWord->next;
01666             }
01667
01668             int32_t nextIndex = hashEntry->nextIndex;
01669             Clay__MeasureTextCacheItemArray_Set(&context->measureTextHashMapInternal, elementIndex,
      CLAY__INIT(Clay__MeasureTextCacheItem) { .measuredWordsStartIndex = -1 });
01670             Clay__int32_tArray_Add(&context->measureTextHashMapInternalFreeList, elementIndex);
01671             if (elementIndexPrevious == 0) {
01672                 context->measureTextHashMap.internalArray[hashBucket] = nextIndex;
01673             } else {
01674                 Clay__MeasureTextCacheItem *previousHashEntry =
      Clay__MeasureTextCacheItemArray_Get(&context->measureTextHashMapInternal, elementIndexPrevious);
01675                 previousHashEntry->nextIndex = nextIndex;
01676             }
01677             elementIndex = nextIndex;
01678         } else {
01679             elementIndexPrevious = elementIndex;
01680             elementIndex = hashEntry->nextIndex;
01681         }
01682     }
01683
01684     int32_t newItemIndex = 0;
01685     Clay__MeasureTextCacheItem newCacheItem = { .measuredWordsStartIndex = -1, .id = id, .generation =
      context->generation };
01686     Clay__MeasureTextCacheItem *measured = NULL;
01687     if (context->measureTextHashMapInternalFreeList.length > 0) {
01688         newItemIndex = Clay__int32_tArray_Get(&context->measureTextHashMapInternalFreeList,
      context->measureTextHashMapInternalFreeList.length - 1);
01689         context->measureTextHashMapInternalFreeList.length--;
01690         Clay__MeasureTextCacheItemArray_Set(&context->measureTextHashMapInternal, newItemIndex,
      newCacheItem);
01691         measured = Clay__MeasureTextCacheItemArray_Get(&context->measureTextHashMapInternal,
      newItemIndex);
01692     } else {
01693         if (context->measureTextHashMapInternal.length == context->measureTextHashMapInternal.capacity
      - 1) {
01694             if (context->booleanWarnings.maxTextMeasureCacheExceeded) {
01695                 context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
01696                         .errorType = CLAY_ERROR_TYPE_ELEMENTS_CAPACITY_EXCEEDED,
01697                         .errorText = CLAY_STRING("Clay ran out of capacity while attempting to measure
      text elements. Try using Clay_SetMaxElementCount() with a higher value."),
01698                         .userData = context->errorHandler.userData });
01699                 context->booleanWarnings.maxTextMeasureCacheExceeded = true;
01700             }
01701             return &CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT;
01702         }
01703         measured = Clay__MeasureTextCacheItemArray_Add(&context->measureTextHashMapInternal,
      newCacheItem);
01704         newItemIndex = context->measureTextHashMapInternal.length - 1;
01705     }
01706
01707     int32_t start = 0;
01708     int32_t end = 0;
01709     float lineWidth = 0;
01710     float measuredWidth = 0;
01711     float measuredHeight = 0;
01712     float spaceWidth = Clay__MeasureText(CLAY__INIT(Clay_StringSlice) { .length = 1, .chars =
      CLAY__SPACECHAR.chars, .baseChars = CLAY__SPACECHAR.chars }, config,
      context->mesureTextUserData).width;
01713     Clay__MeasuredWord tempWord = { .next = -1 };
01714     Clay__MeasuredWord *previousWord = &tempWord;
01715     while (end < text->length) {
01716         if (context->measuredWords.length == context->measuredWords.capacity - 1) {
01717             if (!context->booleanWarnings.maxTextMeasureCacheExceeded) {
01718                 context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
01719                     .errorType = CLAY_ERROR_TYPE_TEXT_MEASUREMENT_CAPACITY_EXCEEDED,
01720                     .errorText = CLAY_STRING("Clay has run out of space in it's internal text
      measurement cache. Try using Clay_SetMaxMeasureTextCacheWordCount() (default 16384, with 1 unit
      storing 1 measured word)."),
01721                     .userData = context->errorHandler.userData });
01722                 context->booleanWarnings.maxTextMeasureCacheExceeded = true;
01723             }
01724             return &CLAY__MEASURE_TEXT_CACHE_ITEM_DEFAULT;
01725         }
01726         char current = text->chars[end];
```

```
01727         if (current == ' ' || current == '\n') {
01728             int32_t length = end - start;
01729             Clay_Dimensions dimensions = Clay__MeasureText(CLAY__INIT(Clay_StringSlice) { .length =
      length, .chars = &text->chars[start], .baseChars = text->chars }, config,
      context->mesureTextUserData);
01730             measuredHeight = CLAY__MAX(measuredHeight, dimensions.height);
01731             if (current == ' ') {
01732                 dimensions.width += spaceWidth;
01733                 previousWord = Clay__AddMeasuredWord(CLAY__INIT(Clay__MeasuredWord) { .startOffset =
      start, .length = length + 1, .width = dimensions.width, .next = -1 }, previousWord);
01734                 lineWidth += dimensions.width;
01735             }
01736             if (current == '\n') {
01737                 if (length > 0) {
01738                     previousWord = Clay__AddMeasuredWord(CLAY__INIT(Clay__MeasuredWord) { .startOffset
      = start, .length = length, .width = dimensions.width, .next = -1 }, previousWord);
01739                 }
01740                 previousWord = Clay__AddMeasuredWord(CLAY__INIT(Clay__MeasuredWord) { .startOffset =
      end + 1, .length = 0, .width = 0, .next = -1 }, previousWord);
01741                 lineWidth += dimensions.width;
01742                 measuredWidth = CLAY__MAX(lineWidth, measuredWidth);
01743                 measured->containsNewlines = true;
01744                 lineWidth = 0;
01745             }
01746             start = end + 1;
01747         }
01748         end++;
01749     }
01750     if (end - start > 0) {
01751         Clay_Dimensions dimensions = Clay__MeasureText(CLAY__INIT(Clay_StringSlice) { .length = end -
      start, .chars = &text->chars[start], .baseChars = text->chars }, config, context->mesureTextUserData);
01752         Clay__AddMeasuredWord(CLAY__INIT(Clay__MeasuredWord) { .startOffset = start, .length = end -
      start, .width = dimensions.width, .next = -1 }, previousWord);
01753         lineWidth += dimensions.width;
01754         measuredHeight = CLAY__MAX(measuredHeight, dimensions.height);
01755     }
01756     measuredWidth = CLAY__MAX(lineWidth, measuredWidth);
01757
01758     measured->measuredWordsStartIndex = tempWord.next;
01759     measured->unwrappedDimensions.width = measuredWidth;
01760     measured->unwrappedDimensions.height = measuredHeight;
01761
01762     if (elementIndexPrevious != 0) {
01763         Clay__MeasureTextCacheItemArray_Get(&context->measureTextHashMapInternal,
      elementIndexPrevious)->nextIndex = newItemIndex;
01764     } else {
01765         context->measureTextHashMap.internalArray[hashBucket] = newItemIndex;
01766     }
01767     return measured;
01768 }
01769
01770 bool Clay__PointIsInsideRect(Clay_Vector2 point, Clay_BoundingBox rect) {
01771     return point.x >= rect.x && point.x <= rect.x + rect.width && point.y >= rect.y && point.y <=
      rect.y + rect.height;
01772 }
01773
01774 Clay_LayoutElementHashMapItem* Clay__AddHashMapItem(Clay_ElementId elementId, Clay_LayoutElement*
      layoutElement) {
01775     Clay_Context* context = Clay_GetCurrentContext();
01776     if (context->layoutElementsHashMapInternal.length ==
      context->layoutElementsHashMapInternal.capacity - 1) {
01777         return NULL;
01778     }
01779     Clay_LayoutElementHashMapItem item = { .elementId = elementId, .layoutElement = layoutElement,
      .nextIndex = -1, .generation = context->generation + 1 };
01780     uint32_t hashBucket = elementId.id % context->layoutElementsHashMap.capacity;
01781     int32_t hashItemPrevious = -1;
01782     int32_t hashItemIndex = context->layoutElementsHashMap.internalArray[hashBucket];
01783     while (hashItemIndex != -1) { // Just replace collision, not a big deal - leave it up to the end
      user
01784         Clay_LayoutElementHashMapItem *hashItem =
      Clay__LayoutElementHashMapItemArray_Get(&context->layoutElementsHashMapInternal, hashItemIndex);
01785         if (hashItem->elementId.id == elementId.id) { // Collision - resolve based on generation
01786             item.nextIndex = hashItem->nextIndex;
01787             if (hashItem->generation <= context->generation) { // First collision - assume this is the
      "same" element
01788                 hashItem->generation = context->generation + 1;
01789                 hashItem->layoutElement = layoutElement;
01790                 hashItem->debugData->collision = false;
01791             } else { // Multiple collisions this frame - two elements have the same ID
01792                 context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
01793                     .errorType = CLAY_ERROR_TYPE_DUPLICATE_ID,
01794                     .errorText = CLAY_STRING("An element with this ID was already previously declared
      during this layout."),
01795                     .userData = context->errorHandler.userData });
01796                 if (context->debugModeEnabled) {
01797                     hashItem->debugData->collision = true;
```

```
01798                 }
01799             }
01800             return hashItem;
01801         }
01802         hashItemPrevious = hashItemIndex;
01803         hashItemIndex = hashItem->nextIndex;
01804     }
01805     Clay_LayoutElementHashMapItem *hashItem = Clay__LayoutElementHashMapItemArray_Add(&context->layoutElementsHashMapInternal, item);
01806     hashItem->debugData = Clay__DebugElementDataArray_Add(&context->debugElementData, CLAY__INIT(Clay__DebugElementData) CLAY__DEFAULT_STRUCT);
01807     if (hashItemPrevious != -1) {
01808         Clay__LayoutElementHashMapItemArray_Get(&context->layoutElementsHashMapInternal, hashItemPrevious)->nextIndex = (int32_t)context->layoutElementsHashMapInternal.length - 1;
01809     } else {
01810         context->layoutElementsHashMap.internalArray[hashBucket] = (int32_t)context->layoutElementsHashMapInternal.length - 1;
01811     }
01812     return hashItem;
01813 }
01814
01815 Clay_LayoutElementHashMapItem *Clay__GetHashMapItem(uint32_t id) {
01816     Clay_Context* context = Clay_GetCurrentContext();
01817     uint32_t hashBucket = id % context->layoutElementsHashMap.capacity;
01818     int32_t elementIndex = context->layoutElementsHashMap.internalArray[hashBucket];
01819     while (elementIndex != -1) {
01820         Clay_LayoutElementHashMapItem *hashEntry = Clay__LayoutElementHashMapItemArray_Get(&context->layoutElementsHashMapInternal, elementIndex);
01821         if (hashEntry->elementId.id == id) {
01822             return hashEntry;
01823         }
01824         elementIndex = hashEntry->nextIndex;
01825     }
01826     return &CLAY__LAYOUT_ELEMENT_HASH_MAP_ITEM_DEFAULT;
01827 }
01828
01829 void Clay__GenerateIdForAnonymousElement(Clay_LayoutElement *openLayoutElement) {
01830     Clay_Context* context = Clay_GetCurrentContext();
01831     Clay_LayoutElement *parentElement = Clay_LayoutElementArray_Get(&context->layoutElements, Clay__int32_tArray_Get(&context->openLayoutElementStack, context->openLayoutElementStack.length - 2));
01832     Clay_ElementId elementId = Clay__HashNumber(parentElement->childrenOrTextContent.children.length, parentElement->id);
01833     openLayoutElement->id = elementId.id;
01834     Clay__AddHashMapItem(elementId, openLayoutElement);
01835     Clay__StringArray_Add(&context->layoutElementIdStrings, elementId.stringId);
01836 }
01837
01838 void Clay__ElementPostConfiguration(void) {
01839     Clay_Context* context = Clay_GetCurrentContext();
01840     if (context->booleanWarnings.maxElementsExceeded) {
01841         return;
01842     }
01843     Clay_LayoutElement *openLayoutElement = Clay__GetOpenLayoutElement();
01844     // ID
01845     if (openLayoutElement->id == 0) {
01846         Clay__GenerateIdForAnonymousElement(openLayoutElement);
01847     }
01848     // Layout Config
01849     if (!openLayoutElement->layoutConfig) {
01850         openLayoutElement->layoutConfig = &CLAY_LAYOUT_DEFAULT;
01851     }
01852
01853     // Loop through element configs and handle special cases
01854     openLayoutElement->elementConfigs.internalArray = &context->elementConfigs.internalArray[context->elementConfigs.length];
01855     for (int32_t elementConfigIndex = 0; elementConfigIndex < openLayoutElement->elementConfigs.length; elementConfigIndex++) {
01856         Clay_ElementConfig *config = Clay__ElementConfigArray_Add(&context->elementConfigs, *Clay__ElementConfigArray_Get(&context->elementConfigBuffer, context->elementConfigBuffer.length - openLayoutElement->elementConfigs.length + elementConfigIndex));
01857         openLayoutElement->configsEnabled |= config->type;
01858         switch (config->type) {
01859             case CLAY__ELEMENT_CONFIG_TYPE_RECTANGLE:
01860             case CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER: break;
01861             case CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER: {
01862                 Clay_FloatingElementConfig *floatingConfig = config->config.floatingElementConfig;
01863                 // This looks dodgy but because of the auto generated root element the depth of the tree will always be at least 2 here
01864                 Clay_LayoutElement *hierarchicalParent = Clay_LayoutElementArray_Get(&context->layoutElements, Clay__int32_tArray_Get(&context->openLayoutElementStack, context->openLayoutElementStack.length - 2));
01865                 if (!hierarchicalParent) {
01866                     break;
01867                 }
01868                 uint32_t clipElementId = 0;
01869                 if (floatingConfig->parentId == 0) {
01870                     // If no parent id was specified, attach to the elements direct hierarchical
```

```
    parent
01871                     Clay_FloatingElementConfig newConfig = *floatingConfig;
01872                     newConfig.parentId = hierarchicalParent->id;
01873                     floatingConfig =
    Clay__FloatingElementConfigArray_Add(&context->floatingElementConfigs, newConfig);
01874                     config->config.floatingElementConfig = floatingConfig;
01875                     if (context->openClipElementStack.length > 0) {
01876                         clipElementId = Clay__int32_tArray_Get(&context->openClipElementStack,
    (int)context->openClipElementStack.length - 1);
01877                     }
01878                 } else {
01879                     Clay_LayoutElementHashMapItem *parentItem =
    Clay__GetHashMapItem(floatingConfig->parentId);
01880                     if (!parentItem) {
01881                         context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
01882                             .errorType = CLAY_ERROR_TYPE_FLOATING_CONTAINER_PARENT_NOT_FOUND,
01883                             .errorText = CLAY_STRING("A floating element was declared with a parentId,
    but no element with that ID was found."),
01884                             .userData = context->errorHandler.userData });
01885                     } else {
01886                         clipElementId = Clay__int32_tArray_Get(&context->layoutElementClipElementIds,
    parentItem->layoutElement - context->layoutElements.internalArray);
01887                     }
01888                 }
01889                 Clay__LayoutElementTreeRootArray_Add(&context->layoutElementTreeRoots,
    CLAY__INIT(Clay__LayoutElementTreeRoot) {
01890                     .layoutElementIndex = Clay__int32_tArray_Get(&context->openLayoutElementStack,
    context->openLayoutElementStack.length - 1),
01891                     .parentId = floatingConfig->parentId,
01892                     .clipElementId = clipElementId,
01893                     .zIndex = floatingConfig->zIndex,
01894                 });
01895                 break;
01896             }
01897             case CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER: {
01898                 Clay__int32_tArray_Add(&context->openClipElementStack, (int)openLayoutElement->id);
01899                 // Retrieve or create cached data to track scroll position across frames
01900                 Clay__ScrollContainerDataInternal *scrollOffset = CLAY__NULL;
01901                 for (int32_t i = 0; i < context->scrollContainerDatas.length; i++) {
01902                     Clay__ScrollContainerDataInternal *mapping =
    Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
01903                     if (openLayoutElement->id == mapping->elementId) {
01904                         scrollOffset = mapping;
01905                         scrollOffset->layoutElement = openLayoutElement;
01906                         scrollOffset->openThisFrame = true;
01907                     }
01908                 }
01909                 if (!scrollOffset) {
01910                     scrollOffset =
    Clay__ScrollContainerDataInternalArray_Add(&context->scrollContainerDatas,
    CLAY__INIT(Clay__ScrollContainerDataInternal){.layoutElement = openLayoutElement, .scrollOrigin =
    {-1,-1}, .elementId = openLayoutElement->id, .openThisFrame = true});
01911                 }
01912                 if (context->externalScrollHandlingEnabled) {
01913                     scrollOffset->scrollPosition = Clay__QueryScrollOffset(scrollOffset->elementId,
    context->queryScrollOffsetUserData);
01914                 }
01915                 break;
01916             }
01917             case CLAY__ELEMENT_CONFIG_TYPE_CUSTOM: break;
01918             case CLAY__ELEMENT_CONFIG_TYPE_IMAGE: {
01919                 Clay__LayoutElementPointerArray_Add(&context->imageElementPointers,
    openLayoutElement);
01920                 break;
01921             }
01922             case CLAY__ELEMENT_CONFIG_TYPE_TEXT:
01923             default: break;
01924         }
01925     }
01926     context->elementConfigBuffer.length -= openLayoutElement->elementConfigs.length;
01927 }
01928
01929 void Clay__CloseElement(void) {
01930     Clay_Context* context = Clay_GetCurrentContext();
01931     if (context->booleanWarnings.maxElementsExceeded) {
01932         return;
01933     }
01934     Clay_LayoutElement *openLayoutElement = Clay__GetOpenLayoutElement();
01935     Clay_LayoutConfig *layoutConfig = openLayoutElement->layoutConfig;
01936     bool elementHasScrollHorizontal = false;
01937     bool elementHasScrollVertical = false;
01938     if (Clay__ElementHasConfig(openLayoutElement, CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER)) {
01939         Clay_ScrollElementConfig *scrollConfig = Clay__FindElementConfigWithType(openLayoutElement,
    CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
01940         elementHasScrollHorizontal = scrollConfig->horizontal;
01941         elementHasScrollVertical = scrollConfig->vertical;
01942         context->openClipElementStack.length--;
```

```
01943     }
01944
01945     // Attach children to the current open element
01946     openLayoutElement->childrenOrTextContent.children.elements =
      &context->layoutElementChildren.internalArray[context->layoutElementChildren.length];
01947     if (layoutConfig->layoutDirection == CLAY_LEFT_TO_RIGHT) {
01948         openLayoutElement->dimensions.width = (float)(layoutConfig->padding.left +
      layoutConfig->padding.right);
01949         for (int32_t i = 0; i < openLayoutElement->childrenOrTextContent.children.length; i++) {
01950             int32_t childIndex = Clay__int32_tArray_Get(&context->layoutElementChildrenBuffer,
      (int)context->layoutElementChildrenBuffer.length -
      openLayoutElement->childrenOrTextContent.children.length + i);
01951             Clay_LayoutElement *child = Clay_LayoutElementArray_Get(&context->layoutElements,
      childIndex);
01952             openLayoutElement->dimensions.width += child->dimensions.width;
01953             openLayoutElement->dimensions.height = CLAY__MAX(openLayoutElement->dimensions.height,
      child->dimensions.height + layoutConfig->padding.top + layoutConfig->padding.bottom);
01954             // Minimum size of child elements doesn't matter to scroll containers as they can shrink
      and hide their contents
01955             if (!elementHasScrollHorizontal) {
01956                 openLayoutElement->minDimensions.width += child->minDimensions.width;
01957             }
01958             if (!elementHasScrollVertical) {
01959                 openLayoutElement->minDimensions.height =
      CLAY__MAX(openLayoutElement->minDimensions.height, child->minDimensions.height +
      layoutConfig->padding.top + layoutConfig->padding.bottom);
01960             }
01961             Clay__int32_tArray_Add(&context->layoutElementChildren, childIndex);
01962         }
01963         float childGap = (float)(CLAY__MAX(openLayoutElement->childrenOrTextContent.children.length -
      1, 0) * layoutConfig->childGap);
01964         openLayoutElement->dimensions.width += childGap; // TODO this is technically a bug with
      childgap and scroll containers
01965         openLayoutElement->minDimensions.width += childGap;
01966     }
01967     else if (layoutConfig->layoutDirection == CLAY_TOP_TO_BOTTOM) {
01968         openLayoutElement->dimensions.height = (float)(layoutConfig->padding.top +
      layoutConfig->padding.bottom);
01969         for (int32_t i = 0; i < openLayoutElement->childrenOrTextContent.children.length; i++) {
01970             int32_t childIndex = Clay__int32_tArray_Get(&context->layoutElementChildrenBuffer,
      (int)context->layoutElementChildrenBuffer.length -
      openLayoutElement->childrenOrTextContent.children.length + i);
01971             Clay_LayoutElement *child = Clay_LayoutElementArray_Get(&context->layoutElements,
      childIndex);
01972             openLayoutElement->dimensions.height += child->dimensions.height;
01973             openLayoutElement->dimensions.width = CLAY__MAX(openLayoutElement->dimensions.width,
      child->dimensions.width + layoutConfig->padding.left + layoutConfig->padding.right);
01974             // Minimum size of child elements doesn't matter to scroll containers as they can shrink
      and hide their contents
01975             if (!elementHasScrollVertical) {
01976                 openLayoutElement->minDimensions.height += child->minDimensions.height;
01977             }
01978             if (!elementHasScrollHorizontal) {
01979                 openLayoutElement->minDimensions.width =
      CLAY__MAX(openLayoutElement->minDimensions.width, child->minDimensions.width +
      layoutConfig->padding.left + layoutConfig->padding.right);
01980             }
01981             Clay__int32_tArray_Add(&context->layoutElementChildren, childIndex);
01982         }
01983         float childGap = (float)(CLAY__MAX(openLayoutElement->childrenOrTextContent.children.length -
      1, 0) * layoutConfig->childGap);
01984         openLayoutElement->dimensions.height += childGap; // TODO this is technically a bug with
      childgap and scroll containers
01985         openLayoutElement->minDimensions.height += childGap;
01986     }
01987
01988     context->layoutElementChildrenBuffer.length -=
      openLayoutElement->childrenOrTextContent.children.length;
01989
01990     // Clamp element min and max width to the values configured in the layout
01991     if (layoutConfig->sizing.width.type != CLAY__SIZING_TYPE_PERCENT) {
01992         if (layoutConfig->sizing.width.size.minMax.max <= 0) { // Set the max size if the user didn't
      specify, makes calculations easier
01993             layoutConfig->sizing.width.size.minMax.max = CLAY__MAXFLOAT;
01994         }
01995         openLayoutElement->dimensions.width = CLAY__MIN(CLAY__MAX(openLayoutElement->dimensions.width,
      layoutConfig->sizing.width.size.minMax.min), layoutConfig->sizing.width.size.minMax.max);
01996         openLayoutElement->minDimensions.width =
      CLAY__MIN(CLAY__MAX(openLayoutElement->minDimensions.width,
      layoutConfig->sizing.width.size.minMax.min), layoutConfig->sizing.width.size.minMax.max);
01997     } else {
01998         openLayoutElement->dimensions.width = 0;
01999     }
02000
02001     // Clamp element min and max height to the values configured in the layout
02002     if (layoutConfig->sizing.height.type != CLAY__SIZING_TYPE_PERCENT) {
02003         if (layoutConfig->sizing.height.size.minMax.max <= 0) { // Set the max size if the user didn't
```

```
    specify, makes calculations easier
02004             layoutConfig->sizing.height.size.minMax.max = CLAY__MAXFLOAT;
02005         }
02006         openLayoutElement->dimensions.height =
    CLAY__MIN(CLAY__MAX(openLayoutElement->dimensions.height,
    layoutConfig->sizing.height.size.minMax.min), layoutConfig->sizing.height.size.minMax.max);
02007         openLayoutElement->minDimensions.height =
    CLAY__MIN(CLAY__MAX(openLayoutElement->minDimensions.height,
    layoutConfig->sizing.height.size.minMax.min), layoutConfig->sizing.height.size.minMax.max);
02008     } else {
02009         openLayoutElement->dimensions.height = 0;
02010     }
02011
02012     bool elementIsFloating = Clay__ElementHasConfig(openLayoutElement,
    CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER);
02013
02014     // Close the currently open element
02015     int32_t closingElementIndex = Clay__int32_tArray_RemoveSwapback(&context->openLayoutElementStack,
    (int)context->openLayoutElementStack.length - 1);
02016     openLayoutElement = Clay__GetOpenLayoutElement();
02017
02018     if (!elementIsFloating && context->openLayoutElementStack.length > 1) {
02019         openLayoutElement->childrenOrTextContent.children.length++;
02020         Clay__int32_tArray_Add(&context->layoutElementChildrenBuffer, closingElementIndex);
02021     }
02022 }
02023
02024 void Clay__OpenElement(void) {
02025     Clay_Context* context = Clay_GetCurrentContext();
02026     if (context->layoutElements.length == context->layoutElements.capacity - 1 ||
    context->booleanWarnings.maxElementsExceeded) {
02027         context->booleanWarnings.maxElementsExceeded = true;
02028         return;
02029     }
02030     Clay_LayoutElement layoutElement = CLAY__DEFAULT_STRUCT;
02031     Clay_LayoutElementArray_Add(&context->layoutElements, layoutElement);
02032     Clay__int32_tArray_Add(&context->openLayoutElementStack, context->layoutElements.length - 1);
02033     if (context->openClipElementStack.length > 0) {
02034         Clay__int32_tArray_Set(&context->layoutElementClipElementIds, context->layoutElements.length -
    1, Clay__int32_tArray_Get(&context->openClipElementStack, (int)context->openClipElementStack.length -
    1));
02035     } else {
02036         Clay__int32_tArray_Set(&context->layoutElementClipElementIds, context->layoutElements.length -
    1, 0);
02037     }
02038 }
02039
02040 void Clay__OpenTextElement(Clay_String text, Clay_TextElementConfig *textConfig) {
02041     Clay_Context* context = Clay_GetCurrentContext();
02042     if (context->layoutElements.length == context->layoutElements.capacity - 1 ||
    context->booleanWarnings.maxElementsExceeded) {
02043         context->booleanWarnings.maxElementsExceeded = true;
02044         return;
02045     }
02046     Clay_LayoutElement *parentElement = Clay__GetOpenLayoutElement();
02047     parentElement->childrenOrTextContent.children.length++;
02048
02049     Clay__OpenElement();
02050     Clay_LayoutElement * openLayoutElement = Clay__GetOpenLayoutElement();
02051     Clay__int32_tArray_Add(&context->layoutElementChildrenBuffer, context->layoutElements.length - 1);
02052     Clay__MeasureTextCacheItem *textMeasured = Clay__MeasureTextCached(&text, textConfig);
02053     Clay_ElementId elementId = Clay__HashString(CLAY_STRING("Text"),
    parentElement->childrenOrTextContent.children.length, parentElement->id);
02054     openLayoutElement->id = elementId.id;
02055     Clay__AddHashMapItem(elementId, openLayoutElement);
02056     Clay__StringArray_Add(&context->layoutElementIdStrings, elementId.stringId);
02057     Clay_Dimensions textDimensions = { .width = textMeasured->unwrappedDimensions.width, .height =
    textConfig->lineHeight > 0 ? (float)textConfig->lineHeight : textMeasured->unwrappedDimensions.height
    };
02058     openLayoutElement->dimensions = textDimensions;
02059     openLayoutElement->minDimensions = CLAY__INIT(Clay_Dimensions) { .width =
    textMeasured->unwrappedDimensions.height, .height = textDimensions.height }; // TODO not sure this is
    the best way to decide min width for text
02060     openLayoutElement->childrenOrTextContent.textElementData =
    Clay__TextElementDataArray_Add(&context->textElementData, CLAY__INIT(Clay__TextElementData) { .text =
    text, .preferredDimensions = textMeasured->unwrappedDimensions, .elementIndex =
    context->layoutElements.length - 1 });
02061     openLayoutElement->elementConfigs = CLAY__INIT(Clay__ElementConfigArraySlice) {
02062         .length = 1,
02063         .internalArray = Clay__ElementConfigArray_Add(&context->elementConfigs,
    CLAY__INIT(Clay_ElementConfig) { .type = CLAY__ELEMENT_CONFIG_TYPE_TEXT, .config = {
    .textElementConfig = textConfig }})
02064     };
02065     openLayoutElement->configsEnabled |= CLAY__ELEMENT_CONFIG_TYPE_TEXT;
02066     openLayoutElement->layoutConfig = &CLAY_LAYOUT_DEFAULT;
02067     // Close the currently open element
02068     Clay__int32_tArray_RemoveSwapback(&context->openLayoutElementStack,
```

```
    (int)context->openLayoutElementStack.length - 1);
02069 }
02070
02071 void Clay__InitializeEphemeralMemory(Clay_Context* context) {
02072     int32_t maxElementCount = context->maxElementCount;
02073     // Ephemeral Memory - reset every frame
02074     Clay_Arena *arena = &context->internalArena;
02075     arena->nextAllocation = context->arenaResetOffset;
02076
02077     context->layoutElementChildrenBuffer = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02078     context->layoutElements = Clay_LayoutElementArray_Allocate_Arena(maxElementCount, arena);
02079     context->warnings = Clay__WarningArray_Allocate_Arena(100, arena);
02080
02081     context->layoutConfigs = Clay__LayoutConfigArray_Allocate_Arena(maxElementCount, arena);
02082     context->elementConfigBuffer = Clay__ElementConfigArray_Allocate_Arena(maxElementCount, arena);
02083     context->elementConfigs = Clay__ElementConfigArray_Allocate_Arena(maxElementCount, arena);
02084     context->rectangleElementConfigs =
    Clay__RectangleElementConfigArray_Allocate_Arena(maxElementCount, arena);
02085     context->textElementConfigs = Clay__TextElementConfigArray_Allocate_Arena(maxElementCount, arena);
02086     context->imageElementConfigs = Clay__ImageElementConfigArray_Allocate_Arena(maxElementCount,
    arena);
02087     context->floatingElementConfigs = Clay__FloatingElementConfigArray_Allocate_Arena(maxElementCount,
    arena);
02088     context->scrollElementConfigs = Clay__ScrollElementConfigArray_Allocate_Arena(maxElementCount,
    arena);
02089     context->customElementConfigs = Clay__CustomElementConfigArray_Allocate_Arena(maxElementCount,
    arena);
02090     context->borderElementConfigs = Clay__BorderElementConfigArray_Allocate_Arena(maxElementCount,
    arena);
02091
02092     context->layoutElementIdStrings = Clay__StringArray_Allocate_Arena(maxElementCount, arena);
02093     context->wrappedTextLines = Clay__WrappedTextLineArray_Allocate_Arena(maxElementCount, arena);
02094     context->layoutElementTreeNodeArray1 =
    Clay__LayoutElementTreeNodeArray_Allocate_Arena(maxElementCount, arena);
02095     context->layoutElementTreeRoots = Clay__LayoutElementTreeRootArray_Allocate_Arena(maxElementCount,
    arena);
02096     context->layoutElementChildren = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02097     context->openLayoutElementStack = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02098     context->textElementData = Clay__TextElementDataArray_Allocate_Arena(maxElementCount, arena);
02099     context->imageElementPointers = Clay__LayoutElementPointerArray_Allocate_Arena(maxElementCount,
    arena);
02100     context->renderCommands = Clay_RenderCommandArray_Allocate_Arena(maxElementCount, arena);
02101     context->treeNodeVisited = Clay__BoolArray_Allocate_Arena(maxElementCount, arena);
02102     context->treeNodeVisited.length = context->treeNodeVisited.capacity; // This array is accessed
    directly rather than behaving as a list
02103     context->openClipElementStack = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02104     context->reusableElementIndexBuffer = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02105     context->layoutElementClipElementIds = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02106     context->dynamicStringData = Clay__CharArray_Allocate_Arena(maxElementCount, arena);
02107 }
02108
02109 void Clay__InitializePersistentMemory(Clay_Context* context) {
02110     // Persistent memory - initialized once and not reset
02111     int32_t maxElementCount = context->maxElementCount;
02112     int32_t maxMeasureTextCacheWordCount = context->maxMeasureTextCacheWordCount;
02113     Clay_Arena *arena = &context->internalArena;
02114
02115     context->scrollContainerDatas = Clay__ScrollContainerDataInternalArray_Allocate_Arena(10, arena);
02116     context->layoutElementsHashMapInternal =
    Clay__LayoutElementHashMapItemArray_Allocate_Arena(maxElementCount, arena);
02117     context->layoutElementsHashMap = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02118     context->measureTextHashMapInternal =
    Clay__MeasureTextCacheItemArray_Allocate_Arena(maxElementCount, arena);
02119     context->measureTextHashMapInternalFreeList = Clay__int32_tArray_Allocate_Arena(maxElementCount,
    arena);
02120     context->measuredWordsFreeList = Clay__int32_tArray_Allocate_Arena(maxMeasureTextCacheWordCount,
    arena);
02121     context->measureTextHashMap = Clay__int32_tArray_Allocate_Arena(maxElementCount, arena);
02122     context->measuredWords = Clay__MeasuredWordArray_Allocate_Arena(maxMeasureTextCacheWordCount,
    arena);
02123     context->pointerOverIds = Clay__ElementIdArray_Allocate_Arena(maxElementCount, arena);
02124     context->debugElementData = Clay__DebugElementDataArray_Allocate_Arena(maxElementCount, arena);
02125     context->arenaResetOffset = arena->nextAllocation;
02126 }
02127
02128
02129 void Clay__CompressChildrenAlongAxis(bool xAxis, float totalSizeToDistribute, Clay__int32_tArray
    resizableContainerBuffer) {
02130     Clay_Context* context = Clay_GetCurrentContext();
02131     Clay__int32_tArray largestContainers = context->openClipElementStack;
02132
02133     while (totalSizeToDistribute > 0.1) {
02134         largestContainers.length = 0;
02135         float largestSize = 0;
02136         float targetSize = 0;
02137         for (int32_t i = 0; i < resizableContainerBuffer.length; ++i) {
02138             Clay_LayoutElement *childElement = Clay_LayoutElementArray_Get(&context->layoutElements,
```

```
    Clay__int32_tArray_Get(&resizableContainerBuffer, i));
02139             if (!xAxis && Clay__ElementHasConfig(childElement, CLAY__ELEMENT_CONFIG_TYPE_IMAGE)) {
02140                 continue;
02141             }
02142             float childSize = xAxis ? childElement->dimensions.width :
    childElement->dimensions.height;
02143             if ((childSize - largestSize) < 0.1 && (childSize - largestSize) > -0.1) {
02144                 Clay__int32_tArray_Add(&largestContainers,
    Clay__int32_tArray_Get(&resizableContainerBuffer, i));
02145             } else if (childSize > largestSize) {
02146                 targetSize = largestSize;
02147                 largestSize = childSize;
02148                 largestContainers.length = 0;
02149                 Clay__int32_tArray_Add(&largestContainers,
    Clay__int32_tArray_Get(&resizableContainerBuffer, i));
02150             }
02151             else if (childSize > targetSize) {
02152                 targetSize = childSize;
02153             }
02154         }
02155
02156         if (largestContainers.length == 0) {
02157             return;
02158         }
02159
02160         targetSize = CLAY__MAX(targetSize, (largestSize * largestContainers.length) -
    totalSizeToDistribute) / largestContainers.length;
02161
02162         for (int32_t childOffset = 0; childOffset < largestContainers.length; childOffset++) {
02163             int32_t childIndex = Clay__int32_tArray_Get(&largestContainers, childOffset);
02164             Clay_LayoutElement *childElement = Clay_LayoutElementArray_Get(&context->layoutElements,
    childIndex);
02165             float *childSize = xAxis ? &childElement->dimensions.width :
    &childElement->dimensions.height;
02166             float childMinSize = xAxis ? childElement->minDimensions.width :
    childElement->minDimensions.height;
02167             float oldChildSize = *childSize;
02168             *childSize = CLAY__MAX(childMinSize, targetSize);
02169             totalSizeToDistribute -= (oldChildSize - *childSize);
02170             if (*childSize == childMinSize) {
02171                 for (int32_t i = 0; i < resizableContainerBuffer.length; i++) {
02172                     if (Clay__int32_tArray_Get(&resizableContainerBuffer, i) == childIndex) {
02173                         Clay__int32_tArray_RemoveSwapback(&resizableContainerBuffer, i);
02174                         break;
02175                     }
02176                 }
02177             }
02178         }
02179     }
02180 }
02181
02182 void Clay__SizeContainersAlongAxis(bool xAxis) {
02183     Clay_Context* context = Clay_GetCurrentContext();
02184     Clay__int32_tArray bfsBuffer = context->layoutElementChildrenBuffer;
02185     Clay__int32_tArray resizableContainerBuffer = context->openLayoutElementStack;
02186     for (int32_t rootIndex = 0; rootIndex < context->layoutElementTreeRoots.length; ++rootIndex) {
02187         bfsBuffer.length = 0;
02188         Clay__LayoutElementTreeRoot *root =
    Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, rootIndex);
02189         Clay_LayoutElement *rootElement = Clay_LayoutElementArray_Get(&context->layoutElements,
    (int)root->layoutElementIndex);
02190         Clay__int32_tArray_Add(&bfsBuffer, (int32_t)root->layoutElementIndex);
02191
02192         // Size floating containers to their parents
02193         if (Clay__ElementHasConfig(rootElement, CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER)) {
02194             Clay_FloatingElementConfig *floatingElementConfig =
    Clay__FindElementConfigWithType(rootElement,
    CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER).floatingElementConfig;
02195             Clay_LayoutElementHashMapItem *parentItem =
    Clay__GetHashMapItem(floatingElementConfig->parentId);
02196             if (parentItem) {
02197                 Clay_LayoutElement *parentLayoutElement = parentItem->layoutElement;
02198                 if (rootElement->layoutConfig->sizing.width.type == CLAY__SIZING_TYPE_GROW) {
02199                     rootElement->dimensions.width = parentLayoutElement->dimensions.width;
02200                 }
02201                 if (rootElement->layoutConfig->sizing.height.type == CLAY__SIZING_TYPE_GROW) {
02202                     rootElement->dimensions.height = parentLayoutElement->dimensions.height;
02203                 }
02204             }
02205         }
02206
02207         rootElement->dimensions.width = CLAY__MIN(CLAY__MAX(rootElement->dimensions.width,
    rootElement->layoutConfig->sizing.width.size.minMax.min),
    rootElement->layoutConfig->sizing.width.size.minMax.max);
02208         rootElement->dimensions.height = CLAY__MIN(CLAY__MAX(rootElement->dimensions.height,
    rootElement->layoutConfig->sizing.height.size.minMax.min),
    rootElement->layoutConfig->sizing.height.size.minMax.max);
```

```
02209
02210         for (int32_t i = 0; i < bfsBuffer.length; ++i) {
02211             int32_t parentIndex = Clay__int32_tArray_Get(&bfsBuffer, i);
02212             Clay_LayoutElement *parent = Clay_LayoutElementArray_Get(&context->layoutElements,
      parentIndex);
02213             Clay_LayoutConfig *parentStyleConfig = parent->layoutConfig;
02214             int32_t growContainerCount = 0;
02215             float parentSize = xAxis ? parent->dimensions.width : parent->dimensions.height;
02216             float parentPadding = (float)(xAxis ? (parent->layoutConfig->padding.left +
      parent->layoutConfig->padding.right) : (parent->layoutConfig->padding.top +
      parent->layoutConfig->padding.bottom));
02217             float innerContentSize = 0, growContainerContentSize = 0, totalPaddingAndChildGaps =
      parentPadding;
02218             bool sizingAlongAxis = (xAxis && parentStyleConfig->layoutDirection == CLAY_LEFT_TO_RIGHT)
      || (!xAxis && parentStyleConfig->layoutDirection == CLAY_TOP_TO_BOTTOM);
02219             resizableContainerBuffer.length = 0;
02220             float parentChildGap = parentStyleConfig->childGap;
02221
02222             for (int32_t childOffset = 0; childOffset < parent->childrenOrTextContent.children.length;
      childOffset++) {
02223                 int32_t childElementIndex =
      parent->childrenOrTextContent.children.elements[childOffset];
02224                 Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements, childElementIndex);
02225                 Clay_SizingAxis childSizing = xAxis ? childElement->layoutConfig->sizing.width :
      childElement->layoutConfig->sizing.height;
02226                 float childSize = xAxis ? childElement->dimensions.width :
      childElement->dimensions.height;
02227
02228                 if (!Clay__ElementHasConfig(childElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT) &&
      childElement->childrenOrTextContent.children.length > 0) {
02229                     Clay__int32_tArray_Add(&bfsBuffer, childElementIndex);
02230                 }
02231
02232                 if (childSizing.type != CLAY__SIZING_TYPE_PERCENT && childSizing.type !=
      CLAY__SIZING_TYPE_FIXED && (!Clay__ElementHasConfig(childElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT) ||
      (Clay__FindElementConfigWithType(childElement,
      CLAY__ELEMENT_CONFIG_TYPE_TEXT).textElementConfig->wrapMode == CLAY_TEXT_WRAP_WORDS))) {
02233                     Clay__int32_tArray_Add(&resizableContainerBuffer, childElementIndex);
02234                 }
02235
02236                 if (sizingAlongAxis) {
02237                     innerContentSize += (childSizing.type == CLAY__SIZING_TYPE_PERCENT ? 0 :
      childSize);
02238                     if (childSizing.type == CLAY__SIZING_TYPE_GROW) {
02239                         growContainerContentSize += childSize;
02240                         growContainerCount++;
02241                     }
02242                     if (childOffset > 0) {
02243                         innerContentSize += parentChildGap; // For children after index 0, the
      childAxisOffset is the gap from the previous child
02244                         totalPaddingAndChildGaps += parentChildGap;
02245                     }
02246                 } else {
02247                     innerContentSize = CLAY__MAX(childSize, innerContentSize);
02248                 }
02249             }
02250
02251             // Expand percentage containers to size
02252             for (int32_t childOffset = 0; childOffset < parent->childrenOrTextContent.children.length;
      childOffset++) {
02253                 int32_t childElementIndex =
      parent->childrenOrTextContent.children.elements[childOffset];
02254                 Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements, childElementIndex);
02255                 Clay_SizingAxis childSizing = xAxis ? childElement->layoutConfig->sizing.width :
      childElement->layoutConfig->sizing.height;
02256                 float *childSize = xAxis ? &childElement->dimensions.width :
      &childElement->dimensions.height;
02257                 if (childSizing.type == CLAY__SIZING_TYPE_PERCENT) {
02258                     *childSize = (parentSize - totalPaddingAndChildGaps) * childSizing.size.percent;
02259                     if (sizingAlongAxis) {
02260                         innerContentSize += *childSize;
02261                     }
02262                 }
02263             }
02264
02265             if (sizingAlongAxis) {
02266                 float sizeToDistribute = parentSize - parentPadding - innerContentSize;
02267                 // The content is too large, compress the children as much as possible
02268                 if (sizeToDistribute < 0) {
02269                     // If the parent can scroll in the axis direction in this direction, don't
      compress children, just leave them alone
02270                     if (Clay__ElementHasConfig(parent, CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER)) {
02271                         Clay_ScrollElementConfig *scrollElementConfig =
      Clay__FindElementConfigWithType(parent,
      CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
```

```
02272                         if (((xAxis && scrollElementConfig->horizontal) || (!xAxis &&
      scrollElementConfig->vertical))) {
02273                             continue;
02274                         }
02275                     }
02276                     // Scrolling containers preferentially compress before others
02277                     Clay__CompressChildrenAlongAxis(xAxis, -sizeToDistribute,
      resizableContainerBuffer);
02278                 // The content is too small, allow SIZING_GROW containers to expand
02279                 } else if (sizeToDistribute > 0 && growContainerCount > 0) {
02280                     float targetSize = (sizeToDistribute + growContainerContentSize) /
      (float)growContainerCount;
02281                     for (int32_t childOffset = 0; childOffset < resizableContainerBuffer.length;
      childOffset++) {
02282                         Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements,
      Clay__int32_tArray_Get(&resizableContainerBuffer, childOffset));
02283                         Clay_SizingAxis childSizing = xAxis ? childElement->layoutConfig->sizing.width
      : childElement->layoutConfig->sizing.height;
02284                         if (childSizing.type == CLAY__SIZING_TYPE_GROW) {
02285                             float *childSize = xAxis ? &childElement->dimensions.width :
      &childElement->dimensions.height;
02286                             float *minSize = xAxis ? &childElement->minDimensions.width :
      &childElement->minDimensions.height;
02287                             if (targetSize < *minSize) {
02288                                 growContainerContentSize -= *minSize;
02289                                 Clay__int32_tArray_RemoveSwapback(&resizableContainerBuffer,
      childOffset);
02290                                 growContainerCount--;
02291                                 targetSize = (sizeToDistribute + growContainerContentSize) /
      (float)growContainerCount;
02292                                 childOffset = -1;
02293                                 continue;
02294                             }
02295                             *childSize = targetSize;
02296                         }
02297                     }
02298                 }
02299             // Sizing along the non layout axis ("off axis")
02300             } else {
02301                 for (int32_t childOffset = 0; childOffset < resizableContainerBuffer.length;
      childOffset++) {
02302                     Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements,
      Clay__int32_tArray_Get(&resizableContainerBuffer, childOffset));
02303                     Clay_SizingAxis childSizing = xAxis ? childElement->layoutConfig->sizing.width :
      childElement->layoutConfig->sizing.height;
02304                     float *childSize = xAxis ? &childElement->dimensions.width :
      &childElement->dimensions.height;
02305
02306                     if (!xAxis && Clay__ElementHasConfig(childElement,
      CLAY__ELEMENT_CONFIG_TYPE_IMAGE)) {
02307                         continue; // Currently we don't support resizing aspect ratio images on the Y
      axis because it would break the ratio
02308                     }
02309
02310                     // If we're laying out the children of a scroll panel, grow containers expand to
      the height of the inner content, not the outer container
02311                     float maxSize = parentSize - parentPadding;
02312                     if (Clay__ElementHasConfig(parent, CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER)) {
02313                         Clay_ScrollElementConfig *scrollElementConfig =
      Clay__FindElementConfigWithType(parent,
      CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
02314                         if (((xAxis && scrollElementConfig->horizontal) || (!xAxis &&
      scrollElementConfig->vertical))) {
02315                             maxSize = CLAY__MAX(maxSize, innerContentSize);
02316                         }
02317                     }
02318                     if (childSizing.type == CLAY__SIZING_TYPE_FIT) {
02319                         *childSize = CLAY__MAX(childSizing.size.minMax.min, CLAY__MIN(*childSize,
      maxSize));
02320                     } else if (childSizing.type == CLAY__SIZING_TYPE_GROW) {
02321                         *childSize = CLAY__MIN(maxSize, childSizing.size.minMax.max);
02322                     }
02323                 }
02324             }
02325         }
02326     }
02327 }
02328
02329 Clay_String Clay__IntToString(int32_t integer) {
02330     if (integer == 0) {
02331         return CLAY__INIT(Clay_String) { .length = 1, .chars = "0" };
02332     }
02333     Clay_Context* context = Clay_GetCurrentContext();
02334     char *chars = (char *)(context->dynamicStringData.internalArray +
      context->dynamicStringData.length);
```

```
02335     int32_t length = 0;
02336     int32_t sign = integer;
02337
02338     if (integer < 0) {
02339         integer = -integer;
02340     }
02341     while (integer > 0) {
02342         chars[length++] = (char)(integer % 10 + '0');
02343         integer /= 10;
02344     }
02345
02346     if (sign < 0) {
02347         chars[length++] = '-';
02348     }
02349
02350     // Reverse the string to get the correct order
02351     for (int32_t j = 0, k = length - 1; j < k; j++, k--) {
02352         char temp = chars[j];
02353         chars[j] = chars[k];
02354         chars[k] = temp;
02355     }
02356     context->dynamicStringData.length += length;
02357     return CLAY__INIT(Clay_String) { .length = length, .chars = chars };
02358 }
02359
02360 void Clay__AddRenderCommand(Clay_RenderCommand renderCommand) {
02361     Clay_Context* context = Clay_GetCurrentContext();
02362     if (context->renderCommands.length < context->renderCommands.capacity - 1) {
02363         Clay_RenderCommandArray_Add(&context->renderCommands, renderCommand);
02364     } else {
02365         if (!context->booleanWarnings.maxRenderCommandsExceeded) {
02366             context->booleanWarnings.maxRenderCommandsExceeded = true;
02367             context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
02368                 .errorType = CLAY_ERROR_TYPE_ELEMENTS_CAPACITY_EXCEEDED,
02369                 .errorText = CLAY_STRING("Clay ran out of capacity while attempting to create render
    commands. This is usually caused by a large amount of wrapping text elements while close to the max
    element capacity. Try using Clay_SetMaxElementCount() with a higher value."),
02370                 .userData = context->errorHandler.userData });
02371         }
02372     }
02373 }
02374
02375 bool Clay__ElementIsOffscreen(Clay_BoundingBox *boundingBox) {
02376     Clay_Context* context = Clay_GetCurrentContext();
02377     if (context->disableCulling) {
02378         return false;
02379     }
02380
02381     return (boundingBox->x > (float)context->layoutDimensions.width) ||
02382            (boundingBox->y > (float)context->layoutDimensions.height) ||
02383            (boundingBox->x + boundingBox->width < 0) ||
02384            (boundingBox->y + boundingBox->height < 0);
02385 }
02386
02387 void Clay__CalculateFinalLayout() {
02388     Clay_Context* context = Clay_GetCurrentContext();
02389     // Calculate sizing along the X axis
02390     Clay__SizeContainersAlongAxis(true);
02391
02392     // Wrap text
02393     for (int32_t textElementIndex = 0; textElementIndex < context->textElementData.length;
    ++textElementIndex) {
02394         Clay__TextElementData *textElementData =
    Clay__TextElementDataArray_Get(&context->textElementData, textElementIndex);
02395         textElementData->wrappedLines = CLAY__INIT(Clay__WrappedTextLineArraySlice) { .length = 0,
    .internalArray = &context->wrappedTextLines.internalArray[context->wrappedTextLines.length] };
02396         Clay_LayoutElement *containerElement = Clay_LayoutElementArray_Get(&context->layoutElements,
    (int)textElementData->elementIndex);
02397         Clay_TextElementConfig *textConfig = Clay__FindElementConfigWithType(containerElement,
    CLAY__ELEMENT_CONFIG_TYPE_TEXT).textElementConfig;
02398         Clay__MeasureTextCacheItem *measureTextCacheItem =
    Clay__MeasureTextCached(&textElementData->text, textConfig);
02399         float lineWidth = 0;
02400         float lineHeight = textConfig->lineHeight > 0 ? (float)textConfig->lineHeight :
    textElementData->preferredDimensions.height;
02401         int32_t lineLengthChars = 0;
02402         int32_t lineStartOffset = 0;
02403         if (!measureTextCacheItem->containsNewlines && textElementData->preferredDimensions.width <=
    containerElement->dimensions.width) {
02404             Clay__WrappedTextLineArray_Add(&context->wrappedTextLines,
    CLAY__INIT(Clay__WrappedTextLine) { containerElement->dimensions,  textElementData->text });
02405             textElementData->wrappedLines.length++;
02406             continue;
02407         }
02408         int32_t wordIndex = measureTextCacheItem->measuredWordsStartIndex;
02409         while (wordIndex != -1) {
02410             if (context->wrappedTextLines.length > context->wrappedTextLines.capacity - 1) {
```

```
02411                 break;
02412             }
02413             Clay__MeasuredWord *measuredWord = Clay__MeasuredWordArray_Get(&context->measuredWords,
      wordIndex);
02414             // Only word on the line is too large, just render it anyway
02415             if (lineLengthChars == 0 && lineWidth + measuredWord->width >
      containerElement->dimensions.width) {
02416                 Clay__WrappedTextLineArray_Add(&context->wrappedTextLines,
      CLAY__INIT(Clay__WrappedTextLine) { { measuredWord->width, lineHeight }, { .length =
      measuredWord->length, .chars = &textElementData->text.chars[measuredWord->startOffset] } });
02417                 textElementData->wrappedLines.length++;
02418                 wordIndex = measuredWord->next;
02419                 lineStartOffset = measuredWord->startOffset + measuredWord->length;
02420             }
02421             // measuredWord->length == 0 means a newline character
02422             else if (measuredWord->length == 0 || lineWidth + measuredWord->width >
      containerElement->dimensions.width) {
02423                 // Wrapped text lines list has overflowed, just render out the line
02424                 Clay__WrappedTextLineArray_Add(&context->wrappedTextLines,
      CLAY__INIT(Clay__WrappedTextLine) { { lineWidth, lineHeight }, { .length = lineLengthChars, .chars =
      &textElementData->text.chars[lineStartOffset] } });
02425                 textElementData->wrappedLines.length++;
02426                 if (lineLengthChars == 0 || measuredWord->length == 0) {
02427                     wordIndex = measuredWord->next;
02428                 }
02429                 lineWidth = 0;
02430                 lineLengthChars = 0;
02431                 lineStartOffset = measuredWord->startOffset;
02432             } else {
02433                 lineWidth += measuredWord->width;
02434                 lineLengthChars += measuredWord->length;
02435                 wordIndex = measuredWord->next;
02436             }
02437         }
02438         if (lineLengthChars > 0) {
02439             Clay__WrappedTextLineArray_Add(&context->wrappedTextLines,
      CLAY__INIT(Clay__WrappedTextLine) { { lineWidth, lineHeight }, {.length = lineLengthChars, .chars =
      &textElementData->text.chars[lineStartOffset] } });
02440             textElementData->wrappedLines.length++;
02441         }
02442         containerElement->dimensions.height = lineHeight *
      (float)textElementData->wrappedLines.length;
02443     }
02444
02445     // Scale vertical image heights according to aspect ratio
02446     for (int32_t i = 0; i < context->imageElementPointers.length; ++i) {
02447         Clay_LayoutElement* imageElement =
      Clay__LayoutElementPointerArray_Get(&context->imageElementPointers, i);
02448         Clay_ImageElementConfig *config = Clay__FindElementConfigWithType(imageElement,
      CLAY__ELEMENT_CONFIG_TYPE_IMAGE).imageElementConfig;
02449         imageElement->dimensions.height = (config->sourceDimensions.height /
      CLAY__MAX(config->sourceDimensions.width, 1)) * imageElement->dimensions.width;
02450     }
02451
02452     // Propagate effect of text wrapping, image aspect scaling etc. on height of parents
02453     Clay__LayoutElementTreeNodeArray dfsBuffer = context->layoutElementTreeNodeArray1;
02454     dfsBuffer.length = 0;
02455     for (int32_t i = 0; i < context->layoutElementTreeRoots.length; ++i) {
02456         Clay__LayoutElementTreeRoot *root =
      Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, i);
02457         context->treeNodeVisited.internalArray[dfsBuffer.length] = false;
02458         Clay__LayoutElementTreeNodeArray_Add(&dfsBuffer, CLAY__INIT(Clay__LayoutElementTreeNode) {
      .layoutElement = Clay_LayoutElementArray_Get(&context->layoutElements, (int)root->layoutElementIndex)
      });
02459     }
02460     while (dfsBuffer.length > 0) {
02461         Clay__LayoutElementTreeNode *currentElementTreeNode =
      Clay__LayoutElementTreeNodeArray_Get(&dfsBuffer, (int)dfsBuffer.length - 1);
02462         Clay_LayoutElement *currentElement = currentElementTreeNode->layoutElement;
02463         if (!context->treeNodeVisited.internalArray[dfsBuffer.length - 1]) {
02464             context->treeNodeVisited.internalArray[dfsBuffer.length - 1] = true;
02465             // If the element has no children or is the container for a text element, don't bother
      inspecting it
02466             if (Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT) ||
      currentElement->childrenOrTextContent.children.length == 0) {
02467                 dfsBuffer.length--;
02468                 continue;
02469             }
02470             // Add the children to the DFS buffer (needs to be pushed in reverse so that stack
      traversal is in correct layout order)
02471             for (int32_t i = 0; i < currentElement->childrenOrTextContent.children.length; i++) {
02472                 context->treeNodeVisited.internalArray[dfsBuffer.length] = false;
02473                 Clay__LayoutElementTreeNodeArray_Add(&dfsBuffer,
      CLAY__INIT(Clay__LayoutElementTreeNode) { .layoutElement =
      Clay_LayoutElementArray_Get(&context->layoutElements,
      currentElement->childrenOrTextContent.children.elements[i]) });
02474             }
```

```
02475             continue;
02476         }
02477         dfsBuffer.length--;
02478 
02479         // DFS node has been visited, this is on the way back up to the root
02480         Clay_LayoutConfig *layoutConfig = currentElement->layoutConfig;
02481         if (layoutConfig->layoutDirection == CLAY_LEFT_TO_RIGHT) {
02482             // Resize any parent containers that have grown in height along their non layout axis
02483             for (int32_t j = 0; j < currentElement->childrenOrTextContent.children.length; ++j) {
02484                 Clay_LayoutElement *childElement =
    Clay_LayoutElementArray_Get(&context->layoutElements,
    currentElement->childrenOrTextContent.children.elements[j]);
02485                 float childHeightWithPadding = CLAY__MAX(childElement->dimensions.height +
    layoutConfig->padding.top + layoutConfig->padding.bottom, currentElement->dimensions.height);
02486                 currentElement->dimensions.height = CLAY__MIN(CLAY__MAX(childHeightWithPadding,
    layoutConfig->sizing.height.size.minMax.min), layoutConfig->sizing.height.size.minMax.max);
02487             }
02488         } else if (layoutConfig->layoutDirection == CLAY_TOP_TO_BOTTOM) {
02489             // Resizing along the layout axis
02490             float contentHeight = (float)(layoutConfig->padding.top + layoutConfig->padding.bottom);
02491             for (int32_t j = 0; j < currentElement->childrenOrTextContent.children.length; ++j) {
02492                 Clay_LayoutElement *childElement =
    Clay_LayoutElementArray_Get(&context->layoutElements,
    currentElement->childrenOrTextContent.children.elements[j]);
02493                 contentHeight += childElement->dimensions.height;
02494             }
02495             contentHeight += (float)(CLAY__MAX(currentElement->childrenOrTextContent.children.length -
    1, 0) * layoutConfig->childGap);
02496             currentElement->dimensions.height = CLAY__MIN(CLAY__MAX(contentHeight,
    layoutConfig->sizing.height.size.minMax.min), layoutConfig->sizing.height.size.minMax.max);
02497         }
02498     }
02499 
02500     // Calculate sizing along the Y axis
02501     Clay__SizeContainersAlongAxis(false);
02502 
02503     // Sort tree roots by z-index
02504     int32_t sortMax = context->layoutElementTreeRoots.length - 1;
02505     while (sortMax > 0) { // todo dumb bubble sort
02506         for (int32_t i = 0; i < sortMax; ++i) {
02507             Clay__LayoutElementTreeRoot current =
    *Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, i);
02508             Clay__LayoutElementTreeRoot next =
    *Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, i + 1);
02509             if (next.zIndex < current.zIndex) {
02510                 Clay__LayoutElementTreeRootArray_Set(&context->layoutElementTreeRoots, i, next);
02511                 Clay__LayoutElementTreeRootArray_Set(&context->layoutElementTreeRoots, i + 1,
    current);
02512             }
02513         }
02514         sortMax--;
02515     }
02516 
02517     // Calculate final positions and generate render commands
02518     context->renderCommands.length = 0;
02519     dfsBuffer.length = 0;
02520     for (int32_t rootIndex = 0; rootIndex < context->layoutElementTreeRoots.length; ++rootIndex) {
02521         dfsBuffer.length = 0;
02522         Clay__LayoutElementTreeRoot *root =
    Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, rootIndex);
02523         Clay_LayoutElement *rootElement = Clay_LayoutElementArray_Get(&context->layoutElements,
    (int)root->layoutElementIndex);
02524         Clay_Vector2 rootPosition = CLAY__DEFAULT_STRUCT;
02525         Clay_LayoutElementHashMapItem *parentHashMapItem = Clay__GetHashMapItem(root->parentId);
02526         // Position root floating containers
02527         if (Clay__ElementHasConfig(rootElement, CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER) &&
    parentHashMapItem) {
02528             Clay_FloatingElementConfig *config = Clay__FindElementConfigWithType(rootElement,
    CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER).floatingElementConfig;
02529             Clay_Dimensions rootDimensions = rootElement->dimensions;
02530             Clay_BoundingBox parentBoundingBox = parentHashMapItem->boundingBox;
02531             // Set X position
02532             Clay_Vector2 targetAttachPosition = CLAY__DEFAULT_STRUCT;
02533             switch (config->attachment.parent) {
02534                 case CLAY_ATTACH_POINT_LEFT_TOP:
02535                 case CLAY_ATTACH_POINT_LEFT_CENTER:
02536                 case CLAY_ATTACH_POINT_LEFT_BOTTOM: targetAttachPosition.x = parentBoundingBox.x;
    break;
02537                 case CLAY_ATTACH_POINT_CENTER_TOP:
02538                 case CLAY_ATTACH_POINT_CENTER_CENTER:
02539                 case CLAY_ATTACH_POINT_CENTER_BOTTOM: targetAttachPosition.x = parentBoundingBox.x +
    (parentBoundingBox.width / 2); break;
02540                 case CLAY_ATTACH_POINT_RIGHT_TOP:
02541                 case CLAY_ATTACH_POINT_RIGHT_CENTER:
02542                 case CLAY_ATTACH_POINT_RIGHT_BOTTOM: targetAttachPosition.x = parentBoundingBox.x +
    parentBoundingBox.width; break;
02543             }
```

```
02544             switch (config->attachment.element) {
02545                 case CLAY_ATTACH_POINT_LEFT_TOP:
02546                 case CLAY_ATTACH_POINT_LEFT_CENTER:
02547                 case CLAY_ATTACH_POINT_LEFT_BOTTOM: break;
02548                 case CLAY_ATTACH_POINT_CENTER_TOP:
02549                 case CLAY_ATTACH_POINT_CENTER_CENTER:
02550                 case CLAY_ATTACH_POINT_CENTER_BOTTOM: targetAttachPosition.x -= (rootDimensions.width
      / 2); break;
02551                 case CLAY_ATTACH_POINT_RIGHT_TOP:
02552                 case CLAY_ATTACH_POINT_RIGHT_CENTER:
02553                 case CLAY_ATTACH_POINT_RIGHT_BOTTOM: targetAttachPosition.x -= rootDimensions.width;
      break;
02554             }
02555             switch (config->attachment.parent) { // I know I could merge the x and y switch
      statements, but this is easier to read
02556                 case CLAY_ATTACH_POINT_LEFT_TOP:
02557                 case CLAY_ATTACH_POINT_RIGHT_TOP:
02558                 case CLAY_ATTACH_POINT_CENTER_TOP: targetAttachPosition.y = parentBoundingBox.y;
      break;
02559                 case CLAY_ATTACH_POINT_LEFT_CENTER:
02560                 case CLAY_ATTACH_POINT_CENTER_CENTER:
02561                 case CLAY_ATTACH_POINT_RIGHT_CENTER: targetAttachPosition.y = parentBoundingBox.y +
      (parentBoundingBox.height / 2); break;
02562                 case CLAY_ATTACH_POINT_LEFT_BOTTOM:
02563                 case CLAY_ATTACH_POINT_CENTER_BOTTOM:
02564                 case CLAY_ATTACH_POINT_RIGHT_BOTTOM: targetAttachPosition.y = parentBoundingBox.y +
      parentBoundingBox.height; break;
02565             }
02566             switch (config->attachment.element) {
02567                 case CLAY_ATTACH_POINT_LEFT_TOP:
02568                 case CLAY_ATTACH_POINT_RIGHT_TOP:
02569                 case CLAY_ATTACH_POINT_CENTER_TOP: break;
02570                 case CLAY_ATTACH_POINT_LEFT_CENTER:
02571                 case CLAY_ATTACH_POINT_CENTER_CENTER:
02572                 case CLAY_ATTACH_POINT_RIGHT_CENTER: targetAttachPosition.y -= (rootDimensions.height
      / 2); break;
02573                 case CLAY_ATTACH_POINT_LEFT_BOTTOM:
02574                 case CLAY_ATTACH_POINT_CENTER_BOTTOM:
02575                 case CLAY_ATTACH_POINT_RIGHT_BOTTOM: targetAttachPosition.y -= rootDimensions.height;
      break;
02576             }
02577             targetAttachPosition.x += config->offset.x;
02578             targetAttachPosition.y += config->offset.y;
02579             rootPosition = targetAttachPosition;
02580         }
02581         if (root->clipElementId) {
02582             Clay_LayoutElementHashMapItem *clipHashMapItem =
      Clay__GetHashMapItem(root->clipElementId);
02583             if (clipHashMapItem) {
02584                 // Floating elements that are attached to scrolling contents won't be correctly
      positioned if external scroll handling is enabled, fix here
02585                 if (context->externalScrollHandlingEnabled) {
02586                     Clay_ScrollElementConfig *scrollConfig =
      Clay__FindElementConfigWithType(clipHashMapItem->layoutElement,
      CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
02587                     for (int32_t i = 0; i < context->scrollContainerDatas.length; i++) {
02588                         Clay__ScrollContainerDataInternal *mapping =
      Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
02589                         if (mapping->layoutElement == clipHashMapItem->layoutElement) {
02590                             root->pointerOffset = mapping->scrollPosition;
02591                             if (scrollConfig->horizontal) {
02592                                 rootPosition.x += mapping->scrollPosition.x;
02593                             }
02594                             if (scrollConfig->vertical) {
02595                                 rootPosition.y += mapping->scrollPosition.y;
02596                             }
02597                             break;
02598                         }
02599                     }
02600                 }
02601                 Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand) {
02602                     .boundingBox = clipHashMapItem->boundingBox,
02603                     .config = { .scrollElementConfig =
      Clay__StoreScrollElementConfig(CLAY__INIT(Clay_ScrollElementConfig)CLAY__DEFAULT_STRUCT) },
02604                     .zIndex = root->zIndex,
02605                     .id = Clay__RehashWithNumber(rootElement->id, 10), // TODO need a better strategy
      for managing derived ids
02606                     .commandType = CLAY_RENDER_COMMAND_TYPE_SCISSOR_START,
02607                 });
02608             }
02609         }
02610         Clay__LayoutElementTreeNodeArray_Add(&dfsBuffer, CLAY__INIT(Clay__LayoutElementTreeNode) {
      .layoutElement = rootElement, .position = rootPosition, .nextChildOffset = { .x =
      (float)rootElement->layoutConfig->padding.left, .y = (float)rootElement->layoutConfig->padding.top }
      });
02611
02612         context->treeNodeVisited.internalArray[0] = false;
```

```
02613         while (dfsBuffer.length > 0) {
02614             Clay__LayoutElementTreeNode *currentElementTreeNode =
      Clay__LayoutElementTreeNodeArray_Get(&dfsBuffer, (int)dfsBuffer.length - 1);
02615             Clay_LayoutElement *currentElement = currentElementTreeNode->layoutElement;
02616             Clay_LayoutConfig *layoutConfig = currentElement->layoutConfig;
02617             Clay_Vector2 scrollOffset = CLAY__DEFAULT_STRUCT;
02618
02619             // This will only be run a single time for each element in downwards DFS order
02620             if (!context->treeNodeVisited.internalArray[dfsBuffer.length - 1]) {
02621                 context->treeNodeVisited.internalArray[dfsBuffer.length - 1] = true;
02622
02623                 Clay_BoundingBox currentElementBoundingBox = { currentElementTreeNode->position.x,
      currentElementTreeNode->position.y, currentElement->dimensions.width,
      currentElement->dimensions.height };
02624                 if (Clay__ElementHasConfig(currentElement,
      CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER)) {
02625                     Clay_FloatingElementConfig *floatingElementConfig =
      Clay__FindElementConfigWithType(currentElement,
      CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER).floatingElementConfig;
02626                     Clay_Dimensions expand = floatingElementConfig->expand;
02627                     currentElementBoundingBox.x -= expand.width;
02628                     currentElementBoundingBox.width += expand.width * 2;
02629                     currentElementBoundingBox.y -= expand.height;
02630                     currentElementBoundingBox.height += expand.height * 2;
02631                 }
02632
02633                 Clay__ScrollContainerDataInternal *scrollContainerData = CLAY__NULL;
02634                 // Apply scroll offsets to container
02635                 if (Clay__ElementHasConfig(currentElement,
      CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER)) {
02636                     Clay_ScrollElementConfig *scrollConfig =
      Clay__FindElementConfigWithType(currentElement,
      CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
02637
02638                     // This linear scan could theoretically be slow under very strange conditions, but
      I can't imagine a real UI with more than a few 10's of scroll containers
02639                     for (int32_t i = 0; i < context->scrollContainerDatas.length; i++) {
02640                         Clay__ScrollContainerDataInternal *mapping =
      Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
02641                         if (mapping->layoutElement == currentElement) {
02642                             scrollContainerData = mapping;
02643                             mapping->boundingBox = currentElementBoundingBox;
02644                             if (scrollConfig->horizontal) {
02645                                 scrollOffset.x = mapping->scrollPosition.x;
02646                             }
02647                             if (scrollConfig->vertical) {
02648                                 scrollOffset.y = mapping->scrollPosition.y;
02649                             }
02650                             if (context->externalScrollHandlingEnabled) {
02651                                 scrollOffset = CLAY__INIT(Clay_Vector2) CLAY__DEFAULT_STRUCT;
02652                             }
02653                             break;
02654                         }
02655                     }
02656                 }
02657
02658                 Clay_LayoutElementHashMapItem *hashMapItem = Clay__GetHashMapItem(currentElement->id);
02659                 if (hashMapItem) {
02660                     hashMapItem->boundingBox = currentElementBoundingBox;
02661                 }
02662
02663                 int32_t sortedConfigIndexes[20];
02664                 for (int32_t elementConfigIndex = 0; elementConfigIndex <
      currentElement->elementConfigs.length; ++elementConfigIndex) {
02665                     sortedConfigIndexes[elementConfigIndex] = elementConfigIndex;
02666                 }
02667                 sortMax = currentElement->elementConfigs.length - 1;
02668                 while (sortMax > 0) { // todo dumb bubble sort
02669                     for (int32_t i = 0; i < sortMax; ++i) {
02670                         int32_t current = sortedConfigIndexes[i];
02671                         int32_t next = sortedConfigIndexes[i + 1];
02672                         Clay__ElementConfigType currentType =
      Clay__ElementConfigArraySlice_Get(&currentElement->elementConfigs, current)->type;
02673                         Clay__ElementConfigType nextType =
      Clay__ElementConfigArraySlice_Get(&currentElement->elementConfigs, next)->type;
02674                         if (nextType == CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER || currentType ==
      CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER) {
02675                             sortedConfigIndexes[i] = next;
02676                             sortedConfigIndexes[i + 1] = current;
02677                         }
02678                     }
02679                     sortMax--;
02680                 }
02681
02682                 // Create the render commands for this element
02683                 for (int32_t elementConfigIndex = 0; elementConfigIndex <
      currentElement->elementConfigs.length; ++elementConfigIndex) {
```

```
02684                   Clay_ElementConfig *elementConfig =
    Clay__ElementConfigArraySlice_Get(&currentElement->elementConfigs,
    sortedConfigIndexes[elementConfigIndex]);
02685                   Clay_RenderCommand renderCommand = {
02686                       .boundingBox = currentElementBoundingBox,
02687                       .config = elementConfig->config,
02688                       .id = currentElement->id,
02689                   };
02690
02691                   bool offscreen = Clay__ElementIsOffscreen(&currentElementBoundingBox);
02692                   // Culling - Don't bother to generate render commands for rectangles entirely
    outside the screen - this won't stop their children from being rendered if they overflow
02693                   bool shouldRender = !offscreen;
02694                   switch (elementConfig->type) {
02695                       case CLAY__ELEMENT_CONFIG_TYPE_RECTANGLE: {
02696                           renderCommand.commandType = CLAY_RENDER_COMMAND_TYPE_RECTANGLE;
02697                           break;
02698                       }
02699                       case CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER: {
02700                           shouldRender = false;
02701                           break;
02702                       }
02703                       case CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER: {
02704                           renderCommand.commandType = CLAY_RENDER_COMMAND_TYPE_NONE;
02705                           shouldRender = false;
02706                           break;
02707                       }
02708                       case CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER: {
02709                           renderCommand.commandType = CLAY_RENDER_COMMAND_TYPE_SCISSOR_START;
02710                           shouldRender = true;
02711                           break;
02712                       }
02713                       case CLAY__ELEMENT_CONFIG_TYPE_IMAGE: {
02714                           renderCommand.commandType = CLAY_RENDER_COMMAND_TYPE_IMAGE;
02715                           break;
02716                       }
02717                       case CLAY__ELEMENT_CONFIG_TYPE_TEXT: {
02718                           if (!shouldRender) {
02719                               break;
02720                           }
02721                           shouldRender = false;
02722                           Clay_ElementConfigUnion configUnion = elementConfig->config;
02723                           Clay_TextElementConfig *textElementConfig = configUnion.textElementConfig;
02724                           float naturalLineHeight =
    currentElement->childrenOrTextContent.textElementData->preferredDimensions.height;
02725                           float finalLineHeight = textElementConfig->lineHeight > 0 ?
    (float)textElementConfig->lineHeight : naturalLineHeight;
02726                           float lineHeightOffset = (finalLineHeight - naturalLineHeight) / 2;
02727                           float yPosition = lineHeightOffset;
02728                           for (int32_t lineIndex = 0; lineIndex <
    currentElement->childrenOrTextContent.textElementData->wrappedLines.length; ++lineIndex) {
02729                               Clay__WrappedTextLine wrappedLine =
    currentElement->childrenOrTextContent.textElementData->wrappedLines.internalArray[lineIndex]; // todo
    range check
02730                               if (wrappedLine.line.length == 0) {
02731                                   yPosition += finalLineHeight;
02732                                   continue;
02733                               }
02734                               Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand) {
02735                                   .boundingBox = { currentElementBoundingBox.x,
    currentElementBoundingBox.y + yPosition, wrappedLine.dimensions.width, wrappedLine.dimensions.height
    }, // TODO width
02736                                   .config = configUnion,
02737                                   .text = CLAY__INIT(Clay_StringSlice) { .length =
    wrappedLine.line.length, .chars = wrappedLine.line.chars, .baseChars =
    currentElement->childrenOrTextContent.textElementData->text.chars },
02738                                   .zIndex = root->zIndex,
02739                                   .id = Clay__HashNumber(lineIndex, currentElement->id).id,
02740                                   .commandType = CLAY_RENDER_COMMAND_TYPE_TEXT,
02741                               });
02742                               yPosition += finalLineHeight;
02743
02744                               if (!context->disableCulling && (currentElementBoundingBox.y +
    yPosition > context->layoutDimensions.height)) {
02745                                   break;
02746                               }
02747                           }
02748                           break;
02749                       }
02750                       case CLAY__ELEMENT_CONFIG_TYPE_CUSTOM: {
02751                           renderCommand.commandType = CLAY_RENDER_COMMAND_TYPE_CUSTOM;
02752                           break;
02753                       }
02754                       default: break;
02755                   }
02756                   if (shouldRender) {
02757                       Clay__AddRenderCommand(renderCommand);
```

```
02758                     }
02759                     if (offscreen) {
02760                         // NOTE: You may be tempted to try an early return / continue if an element is off screen. Why bother calculating layout for its children, right?
02761                         // Unfortunately, a FLOATING_CONTAINER may be defined that attaches to a child or grandchild of this element, which is large enough to still
02762                         // be on screen, even if this element isn't. That depends on this element and it's children being laid out correctly (even if they are entirely off screen)
02763                     }
02764                 }
02765
02766                 // Setup initial on-axis alignment
02767                 if (!Clay__ElementHasConfig(currentElementTreeNode->layoutElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT)) {
02768                     Clay_Dimensions contentSize = {0,0};
02769                     if (layoutConfig->layoutDirection == CLAY_LEFT_TO_RIGHT) {
02770                         for (int32_t i = 0; i < currentElement->childrenOrTextContent.children.length; ++i) {
02771                             Clay_LayoutElement *childElement = Clay_LayoutElementArray_Get(&context->layoutElements, currentElement->childrenOrTextContent.children.elements[i]);
02772                             contentSize.width += childElement->dimensions.width;
02773                             contentSize.height = CLAY__MAX(contentSize.height, childElement->dimensions.height);
02774                         }
02775                         contentSize.width += (float)(CLAY__MAX(currentElement->childrenOrTextContent.children.length - 1, 0) * layoutConfig->childGap);
02776                         float extraSpace = currentElement->dimensions.width - (float)(layoutConfig->padding.left + layoutConfig->padding.right) - contentSize.width;
02777                         switch (layoutConfig->childAlignment.x) {
02778                             case CLAY_ALIGN_X_LEFT: extraSpace = 0; break;
02779                             case CLAY_ALIGN_X_CENTER: extraSpace /= 2; break;
02780                             default: break;
02781                         }
02782                         currentElementTreeNode->nextChildOffset.x += extraSpace;
02783                     } else {
02784                         for (int32_t i = 0; i < currentElement->childrenOrTextContent.children.length; ++i) {
02785                             Clay_LayoutElement *childElement = Clay_LayoutElementArray_Get(&context->layoutElements, currentElement->childrenOrTextContent.children.elements[i]);
02786                             contentSize.width = CLAY__MAX(contentSize.width, childElement->dimensions.width);
02787                             contentSize.height += childElement->dimensions.height;
02788                         }
02789                         contentSize.height += (float)(CLAY__MAX(currentElement->childrenOrTextContent.children.length - 1, 0) * layoutConfig->childGap);
02790                         float extraSpace = currentElement->dimensions.height - (float)(layoutConfig->padding.top + layoutConfig->padding.bottom) - contentSize.height;
02791                         switch (layoutConfig->childAlignment.y) {
02792                             case CLAY_ALIGN_Y_TOP: extraSpace = 0; break;
02793                             case CLAY_ALIGN_Y_CENTER: extraSpace /= 2; break;
02794                             default: break;
02795                         }
02796                         currentElementTreeNode->nextChildOffset.y += extraSpace;
02797                     }
02798
02799                     if (scrollContainerData) {
02800                         scrollContainerData->contentSize = CLAY__INIT(Clay_Dimensions) { contentSize.width + (float)(layoutConfig->padding.left + layoutConfig->padding.right), contentSize.height + (float)(layoutConfig->padding.top + layoutConfig->padding.bottom) };
02801                     }
02802                 }
02803             }
02804             else {
02805                 // DFS is returning upwards backwards
02806                 bool closeScrollElement = false;
02807                 if (Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER)) {
02808                     closeScrollElement = true;
02809                     Clay_ScrollElementConfig *scrollConfig = Clay__FindElementConfigWithType(currentElement, CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
02810                     for (int32_t i = 0; i < context->scrollContainerDatas.length; i++) {
02811                         Clay__ScrollContainerDataInternal *mapping = Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
02812                         if (mapping->layoutElement == currentElement) {
02813                             if (scrollConfig->horizontal) { scrollOffset.x = mapping->scrollPosition.x; }
02814                             if (scrollConfig->vertical) { scrollOffset.y = mapping->scrollPosition.y; }
02815                             if (context->externalScrollHandlingEnabled) {
02816                                 scrollOffset = CLAY__INIT(Clay_Vector2) CLAY__DEFAULT_STRUCT;
02817                             }
02818                             break;
```

```
02819                         }
02820                     }
02821                 }
02822
02823                 if (Clay__ElementHasConfig(currentElement,
      CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER)) {
02824                     Clay_LayoutElementHashMapItem *currentElementData =
      Clay__GetHashMapItem(currentElement->id);
02825                     Clay_BoundingBox currentElementBoundingBox = currentElementData->boundingBox;
02826
02827                     // Culling - Don't bother to generate render commands for rectangles entirely
      outside the screen - this won't stop their children from being rendered if they overflow
02828                     if (!Clay__ElementIsOffscreen(&currentElementBoundingBox)) {
02829                         Clay_BorderElementConfig *borderConfig =
      Clay__FindElementConfigWithType(currentElement,
      CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER).borderElementConfig;
02830                         Clay_RenderCommand renderCommand = {
02831                                 .boundingBox = currentElementBoundingBox,
02832                                 .config = { .borderElementConfig = borderConfig },
02833                                 .id = Clay__RehashWithNumber(currentElement->id, 4),
02834                                 .commandType = CLAY_RENDER_COMMAND_TYPE_BORDER,
02835                         };
02836                         Clay__AddRenderCommand(renderCommand);
02837                         if (borderConfig->betweenChildren.width > 0 &&
      borderConfig->betweenChildren.color.a > 0) {
02838                             Clay_RectangleElementConfig *rectangleConfig =
      Clay__StoreRectangleElementConfig(CLAY__INIT(Clay_RectangleElementConfig) {.color =
      borderConfig->betweenChildren.color});
02839                             float halfGap = layoutConfig->childGap / 2;
02840                             Clay_Vector2 borderOffset = { (float)layoutConfig->padding.left - halfGap,
      (float)layoutConfig->padding.top - halfGap };
02841                             if (layoutConfig->layoutDirection == CLAY_LEFT_TO_RIGHT) {
02842                                 for (int32_t i = 0; i <
      currentElement->childrenOrTextContent.children.length; ++i) {
02843                                     Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements,
      currentElement->childrenOrTextContent.children.elements[i]);
02844                                     if (i > 0) {
02845                                         Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand) {
02846                                             .boundingBox = { currentElementBoundingBox.x +
      borderOffset.x + scrollOffset.x, currentElementBoundingBox.y + scrollOffset.y,
      (float)borderConfig->betweenChildren.width, currentElement->dimensions.height },
02847                                             .config = { rectangleConfig },
02848                                             .id = Clay__RehashWithNumber(currentElement->id, 5 + i),
02849                                             .commandType = CLAY_RENDER_COMMAND_TYPE_RECTANGLE,
02850                                         });
02851                                     }
02852                                     borderOffset.x += (childElement->dimensions.width +
      (float)layoutConfig->childGap);
02853                                 }
02854                             } else {
02855                                 for (int32_t i = 0; i <
      currentElement->childrenOrTextContent.children.length; ++i) {
02856                                     Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements,
      currentElement->childrenOrTextContent.children.elements[i]);
02857                                     if (i > 0) {
02858                                         Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand) {
02859                                                 .boundingBox = { currentElementBoundingBox.x +
      scrollOffset.x, currentElementBoundingBox.y + borderOffset.y + scrollOffset.y,
      currentElement->dimensions.width, (float)borderConfig->betweenChildren.width },
02860                                                 .config = { rectangleConfig },
02861                                                 .id = Clay__RehashWithNumber(currentElement->id, 5 +
      i),
02862                                                 .commandType = CLAY_RENDER_COMMAND_TYPE_RECTANGLE,
02863                                         });
02864                                     }
02865                                     borderOffset.y += (childElement->dimensions.height +
      (float)layoutConfig->childGap);
02866                                 }
02867                             }
02868                         }
02869                     }
02870                 }
02871                 // This exists because the scissor needs to end _after_ borders between elements
02872                 if (closeScrollElement) {
02873                     Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand) {
02874                         .id = Clay__RehashWithNumber(currentElement->id, 11),
02875                        .commandType = CLAY_RENDER_COMMAND_TYPE_SCISSOR_END,
02876                     });
02877                 }
02878
02879                 dfsBuffer.length--;
02880                 continue;
02881             }
02882
02883             // Add children to the DFS buffer
```

```
02884             if (!Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT)) {
02885                 dfsBuffer.length += currentElement->childrenOrTextContent.children.length;
02886                 for (int32_t i = 0; i < currentElement->childrenOrTextContent.children.length; ++i) {
02887                     Clay_LayoutElement *childElement =
      Clay_LayoutElementArray_Get(&context->layoutElements,
      currentElement->childrenOrTextContent.children.elements[i]);
02888                     // Alignment along non layout axis
02889                     if (layoutConfig->layoutDirection == CLAY_LEFT_TO_RIGHT) {
02890                         currentElementTreeNode->nextChildOffset.y =
      currentElement->layoutConfig->padding.top;
02891                         float whiteSpaceAroundChild = currentElement->dimensions.height -
      (float)(layoutConfig->padding.top + layoutConfig->padding.bottom) - childElement->dimensions.height;
02892                         switch (layoutConfig->childAlignment.y) {
02893                             case CLAY_ALIGN_Y_TOP: break;
02894                             case CLAY_ALIGN_Y_CENTER: currentElementTreeNode->nextChildOffset.y +=
      whiteSpaceAroundChild / 2; break;
02895                             case CLAY_ALIGN_Y_BOTTOM: currentElementTreeNode->nextChildOffset.y +=
      whiteSpaceAroundChild; break;
02896                         }
02897                     } else {
02898                         currentElementTreeNode->nextChildOffset.x =
      currentElement->layoutConfig->padding.left;
02899                         float whiteSpaceAroundChild = currentElement->dimensions.width -
      (float)(layoutConfig->padding.left + layoutConfig->padding.right) - childElement->dimensions.width;
02900                         switch (layoutConfig->childAlignment.x) {
02901                             case CLAY_ALIGN_X_LEFT: break;
02902                             case CLAY_ALIGN_X_CENTER: currentElementTreeNode->nextChildOffset.x +=
      whiteSpaceAroundChild / 2; break;
02903                             case CLAY_ALIGN_X_RIGHT: currentElementTreeNode->nextChildOffset.x +=
      whiteSpaceAroundChild; break;
02904                         }
02905                     }
02906
02907                     Clay_Vector2 childPosition = {
02908                         currentElementTreeNode->position.x + currentElementTreeNode->nextChildOffset.x
      + scrollOffset.x,
02909                         currentElementTreeNode->position.y + currentElementTreeNode->nextChildOffset.y
      + scrollOffset.y,
02910                     };
02911
02912                     // DFS buffer elements need to be added in reverse because stack traversal happens
      backwards
02913                     uint32_t newNodeIndex = dfsBuffer.length - 1 - i;
02914                     dfsBuffer.internalArray[newNodeIndex] = CLAY__INIT(Clay__LayoutElementTreeNode) {
02915                         .layoutElement = childElement,
02916                         .position = { childPosition.x, childPosition.y },
02917                         .nextChildOffset = { .x = (float)childElement->layoutConfig->padding.left, .y
      = (float)childElement->layoutConfig->padding.top },
02918                     };
02919                     context->treeNodeVisited.internalArray[newNodeIndex] = false;
02920
02921                     // Update parent offsets
02922                     if (layoutConfig->layoutDirection == CLAY_LEFT_TO_RIGHT) {
02923                         currentElementTreeNode->nextChildOffset.x += childElement->dimensions.width +
      (float)layoutConfig->childGap;
02924                     } else {
02925                         currentElementTreeNode->nextChildOffset.y += childElement->dimensions.height +
      (float)layoutConfig->childGap;
02926                     }
02927                 }
02928             }
02929         }
02930
02931         if (root->clipElementId) {
02932             Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand) { .id =
      Clay__RehashWithNumber(rootElement->id, 11), .commandType = CLAY_RENDER_COMMAND_TYPE_SCISSOR_END });
02933         }
02934     }
02935 }
02936
02937 void Clay__AttachId(Clay_ElementId elementId) {
02938     Clay_Context* context = Clay_GetCurrentContext();
02939     if (context->booleanWarnings.maxElementsExceeded) {
02940         return;
02941     }
02942     Clay_LayoutElement *openLayoutElement = Clay__GetOpenLayoutElement();
02943     openLayoutElement->id = elementId.id;
02944     Clay__AddHashMapItem(elementId, openLayoutElement);
02945     Clay__StringArray_Add(&context->layoutElementIdStrings, elementId.stringId);
02946 }
02947
02948 void Clay__AttachLayoutConfig(Clay_LayoutConfig *config) {
02949     Clay_Context* context = Clay_GetCurrentContext();
02950     if (context->booleanWarnings.maxElementsExceeded) {
02951         return;
02952     }
02953     Clay__GetOpenLayoutElement()->layoutConfig = config;
```

```
02954 }
02955 void Clay__AttachElementConfig(Clay_ElementConfigUnion config, Clay__ElementConfigType type) {
02956     Clay_Context* context = Clay_GetCurrentContext();
02957     if (context->booleanWarnings.maxElementsExceeded) {
02958         return;
02959     }
02960     Clay_LayoutElement *openLayoutElement = Clay__GetOpenLayoutElement();
02961     openLayoutElement->elementConfigs.length++;
02962     Clay__ElementConfigArray_Add(&context->elementConfigBuffer, CLAY__INIT(Clay_ElementConfig) { .type
      = type, .config = config });
02963 }
02964 Clay_LayoutConfig * Clay__StoreLayoutConfig(Clay_LayoutConfig config) {  return
      Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ? &CLAY_LAYOUT_DEFAULT :
      Clay__LayoutConfigArray_Add(&Clay_GetCurrentContext()->layoutConfigs, config); }
02965 Clay_RectangleElementConfig * Clay__StoreRectangleElementConfig(Clay_RectangleElementConfig config) {
      return Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ?
      &CLAY__RECTANGLE_ELEMENT_CONFIG_DEFAULT :
      Clay__RectangleElementConfigArray_Add(&Clay_GetCurrentContext()->rectangleElementConfigs, config); }
02966 Clay_TextElementConfig * Clay__StoreTextElementConfig(Clay_TextElementConfig config) {  return
      Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ? &CLAY__TEXT_ELEMENT_CONFIG_DEFAULT :
      Clay__TextElementConfigArray_Add(&Clay_GetCurrentContext()->textElementConfigs, config); }
02967 Clay_ImageElementConfig * Clay__StoreImageElementConfig(Clay_ImageElementConfig config) {  return
      Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ? &CLAY__IMAGE_ELEMENT_CONFIG_DEFAULT :
      Clay__ImageElementConfigArray_Add(&Clay_GetCurrentContext()->imageElementConfigs, config); }
02968 Clay_FloatingElementConfig * Clay__StoreFloatingElementConfig(Clay_FloatingElementConfig config) {
      return Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ?
      &CLAY__FLOATING_ELEMENT_CONFIG_DEFAULT :
      Clay__FloatingElementConfigArray_Add(&Clay_GetCurrentContext()->floatingElementConfigs, config); }
02969 Clay_CustomElementConfig * Clay__StoreCustomElementConfig(Clay_CustomElementConfig config) {  return
      Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ? &CLAY__CUSTOM_ELEMENT_CONFIG_DEFAULT :
      Clay__CustomElementConfigArray_Add(&Clay_GetCurrentContext()->customElementConfigs, config); }
02970 Clay_ScrollElementConfig * Clay__StoreScrollElementConfig(Clay_ScrollElementConfig config) {  return
      Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ? &CLAY__SCROLL_ELEMENT_CONFIG_DEFAULT :
      Clay__ScrollElementConfigArray_Add(&Clay_GetCurrentContext()->scrollElementConfigs, config); }
02971 Clay_BorderElementConfig * Clay__StoreBorderElementConfig(Clay_BorderElementConfig config) {  return
      Clay_GetCurrentContext()->booleanWarnings.maxElementsExceeded ? &CLAY__BORDER_ELEMENT_CONFIG_DEFAULT :
      Clay__BorderElementConfigArray_Add(&Clay_GetCurrentContext()->borderElementConfigs, config); }
02972
02973 #pragma region DebugTools
02974 Clay_Color CLAY__DEBUGVIEW_COLOR_1 = {58, 56, 52, 255};
02975 Clay_Color CLAY__DEBUGVIEW_COLOR_2 = {62, 60, 58, 255};
02976 Clay_Color CLAY__DEBUGVIEW_COLOR_3 = {141, 133, 135, 255};
02977 Clay_Color CLAY__DEBUGVIEW_COLOR_4 = {238, 226, 231, 255};
02978 Clay_Color CLAY__DEBUGVIEW_COLOR_SELECTED_ROW = {102, 80, 78, 255};
02979 const int32_t CLAY__DEBUGVIEW_ROW_HEIGHT = 30;
02980 const int32_t CLAY__DEBUGVIEW_OUTER_PADDING = 10;
02981 const int32_t CLAY__DEBUGVIEW_INDENT_WIDTH = 16;
02982 Clay_TextElementConfig Clay__DebugView_TextNameConfig = {.textColor = {238, 226, 231, 255}, .fontSize
      = 16, .wrapMode = CLAY_TEXT_WRAP_NONE };
02983 Clay_LayoutConfig Clay__DebugView_ScrollViewItemLayoutConfig = CLAY__DEFAULT_STRUCT;
02984
02985 CLAY__TYPEDEF(Clay__DebugElementConfigTypeLabelConfig, struct {
02986     Clay_String label;
02987     Clay_Color color;
02988 });
02989
02990 Clay__DebugElementConfigTypeLabelConfig Clay__DebugGetElementConfigTypeLabel(Clay__ElementConfigType
      type) {
02991     switch (type) {
02992         case CLAY__ELEMENT_CONFIG_TYPE_RECTANGLE: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Rectangle"), {243,134,48,255} };
02993         case CLAY__ELEMENT_CONFIG_TYPE_TEXT: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Text"), {105,210,231,255} };
02994         case CLAY__ELEMENT_CONFIG_TYPE_IMAGE: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Image"), {121,189,154,255} };
02995         case CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Floating"), {250,105,0,255} };
02996         case CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Scroll"), {242,196,90,255} };
02997         case CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Border"), {108,91,123, 255} };
02998         case CLAY__ELEMENT_CONFIG_TYPE_CUSTOM: return
      CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Custom"), {11,72,107,255} };
02999         default: break;
03000     }
03001     return CLAY__INIT(Clay__DebugElementConfigTypeLabelConfig) { CLAY_STRING("Error"), {0,0,0,255} };
03002 }
03003
03004 CLAY__TYPEDEF(Clay__RenderDebugLayoutData, struct {
03005     int32_t rowCount;
03006     int32_t selectedElementRowIndex;
03007 });
03008
03009 // Returns row count
03010 Clay__RenderDebugLayoutData Clay__RenderDebugLayoutElementsList(int32_t initialRootsLength, int32_t
      highlightedRowIndex) {
03011     Clay_Context* context = Clay_GetCurrentContext();
```

```
03012     Clay__int32_tArray dfsBuffer = context->reusableElementIndexBuffer;
03013     Clay__DebugView_ScrollViewItemLayoutConfig = CLAY__INIT(Clay_LayoutConfig) { .sizing = { .height =
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT) }, .childGap = 6, .childAlignment = { .y =
      CLAY_ALIGN_Y_CENTER }};
03014     Clay__RenderDebugLayoutData layoutData = CLAY__DEFAULT_STRUCT;
03015
03016     uint32_t highlightedElementId = 0;
03017
03018     for (int32_t rootIndex = 0; rootIndex < initialRootsLength; ++rootIndex) {
03019         dfsBuffer.length = 0;
03020         Clay__LayoutElementTreeRoot *root =
      Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, rootIndex);
03021         Clay__int32_tArray_Add(&dfsBuffer, (int32_t)root->layoutElementIndex);
03022         context->treeNodeVisited.internalArray[0] = false;
03023         if (rootIndex > 0) {
03024             CLAY(CLAY_IDI("Clay__DebugView_EmptyRowOuter", rootIndex), CLAY_LAYOUT({ .sizing = {.width
      = CLAY_SIZING_GROW(0)}, .padding = {CLAY__DEBUGVIEW_INDENT_WIDTH / 2, 0} })) {
03025                 CLAY(CLAY_IDI("Clay__DebugView_EmptyRow", rootIndex), CLAY_LAYOUT({ .sizing = { .width
      = CLAY_SIZING_GROW(0), .height = CLAY_SIZING_FIXED((float)CLAY__DEBUGVIEW_ROW_HEIGHT) }}),
      CLAY_BORDER({ .top = { .width = 1, .color = CLAY__DEBUGVIEW_COLOR_3 } })) {}
03026             }
03027             layoutData.rowCount++;
03028         }
03029         while (dfsBuffer.length > 0) {
03030             int32_t currentElementIndex = Clay__int32_tArray_Get(&dfsBuffer, (int)dfsBuffer.length -
      1);
03031             Clay_LayoutElement *currentElement = Clay_LayoutElementArray_Get(&context->layoutElements,
      (int)currentElementIndex);
03032             if (context->treeNodeVisited.internalArray[dfsBuffer.length - 1]) {
03033                 if (!Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT) &&
      currentElement->childrenOrTextContent.children.length > 0) {
03034                     Clay__CloseElement();
03035                     Clay__CloseElement();
03036                     Clay__CloseElement();
03037                 }
03038                 dfsBuffer.length--;
03039                 continue;
03040             }
03041
03042             if (highlightedRowIndex == layoutData.rowCount) {
03043                 if (context->pointerInfo.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME) {
03044                     context->debugSelectedElementId = currentElement->id;
03045                 }
03046                 highlightedElementId = currentElement->id;
03047             }
03048
03049             context->treeNodeVisited.internalArray[dfsBuffer.length - 1] = true;
03050             Clay_LayoutElementHashMapItem *currentElementData =
      Clay__GetHashMapItem(currentElement->id);
03051             bool offscreen = Clay__ElementIsOffscreen(&currentElementData->boundingBox);
03052             if (context->debugSelectedElementId == currentElement->id) {
03053                 layoutData.selectedElementRowIndex = layoutData.rowCount;
03054             }
03055             CLAY(CLAY_IDI("Clay__DebugView_ElementOuter", currentElement->id),
      Clay__AttachLayoutConfig(&Clay__DebugView_ScrollViewItemLayoutConfig)) {
03056                 // Collapse icon / button
03057                 if (!(Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT) ||
      currentElement->childrenOrTextContent.children.length == 0)) {
03058                     CLAY(CLAY_IDI("Clay__DebugView_CollapseElement", currentElement->id),
03059                         CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED(16), CLAY_SIZING_FIXED(16)},
      .childAlignment = { CLAY_ALIGN_X_CENTER, CLAY_ALIGN_Y_CENTER} }),
03060                         CLAY_BORDER_OUTSIDE_RADIUS(1, CLAY__DEBUGVIEW_COLOR_3, 4)
03061                     ) {
03062                         CLAY_TEXT((currentElementData && currentElementData->debugData->collapsed) ?
      CLAY_STRING("+") : CLAY_STRING("-"), CLAY_TEXT_CONFIG({ .textColor = CLAY__DEBUGVIEW_COLOR_4,
      .fontSize = 16 }));
03063                     }
03064                 } else { // Square dot for empty containers
03065                     CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED(16), CLAY_SIZING_FIXED(16)},
      .childAlignment = { CLAY_ALIGN_X_CENTER, CLAY_ALIGN_Y_CENTER } })) {
03066                         CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED(8), CLAY_SIZING_FIXED(8)} }),
      CLAY_RECTANGLE({ .color = CLAY__DEBUGVIEW_COLOR_3, .cornerRadius = CLAY_CORNER_RADIUS(2) })) {}
03067                     }
03068                 }
03069                 // Collisions and offscreen info
03070                 if (currentElementData) {
03071                     if (currentElementData->debugData->collision) {
03072                         CLAY(CLAY_LAYOUT({ .padding = { 8, 8, 2, 2 } }), CLAY_BORDER_OUTSIDE_RADIUS(1,
      (CLAY__INIT(Clay_Color){177, 147, 8, 255}), 4)) {
03073                             CLAY_TEXT(CLAY_STRING("Duplicate ID"), CLAY_TEXT_CONFIG({ .textColor =
      CLAY__DEBUGVIEW_COLOR_3, .fontSize = 16 }));
03074                         }
03075                     }
03076                     if (offscreen) {
03077                         CLAY(CLAY_LAYOUT({ .padding = { 8, 8, 2, 2 } }), CLAY_BORDER_OUTSIDE_RADIUS(1,
      CLAY__DEBUGVIEW_COLOR_3, 4)) {
03078                             CLAY_TEXT(CLAY_STRING("Offscreen"), CLAY_TEXT_CONFIG({ .textColor =
```

```
    CLAY__DEBUGVIEW_COLOR_3, .fontSize = 16 }));
03079                         }
03080                     }
03081                 }
03082                 Clay_String idString =
    context->layoutElementIdStrings.internalArray[currentElementIndex];
03083                 if (idString.length > 0) {
03084                     CLAY_TEXT(idString, offscreen ? CLAY_TEXT_CONFIG({ .textColor =
    CLAY__DEBUGVIEW_COLOR_3, .fontSize = 16 }) : &Clay__DebugView_TextNameConfig);
03085                 }
03086                 for (int32_t elementConfigIndex = 0; elementConfigIndex <
    currentElement->elementConfigs.length; ++elementConfigIndex) {
03087                     Clay_ElementConfig *elementConfig =
    Clay__ElementConfigArraySlice_Get(&currentElement->elementConfigs, elementConfigIndex);
03088                     Clay__DebugElementConfigTypeLabelConfig config =
    Clay__DebugGetElementConfigTypeLabel(elementConfig->type);
03089                     Clay_Color backgroundColor = config.color;
03090                     backgroundColor.a = 90;
03091                     CLAY(CLAY_LAYOUT({ .padding = { 8, 8, 2, 2 } }), CLAY_RECTANGLE({ .color =
    backgroundColor, .cornerRadius = CLAY_CORNER_RADIUS(4) }), CLAY_BORDER_OUTSIDE_RADIUS(1, config.color,
    4)) {
03092                         CLAY_TEXT(config.label, CLAY_TEXT_CONFIG({ .textColor = offscreen ?
    CLAY__DEBUGVIEW_COLOR_3 : CLAY__DEBUGVIEW_COLOR_4, .fontSize = 16 }));
03093                     }
03094                 }
03095             }
03096
03097             // Render the text contents below the element as a non-interactive row
03098             if (Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT)) {
03099                 layoutData.rowCount++;
03100                 Clay__TextElementData *textElementData =
    currentElement->childrenOrTextContent.textElementData;
03101                 Clay_TextElementConfig *rawTextConfig = offscreen ? CLAY_TEXT_CONFIG({ .textColor =
    CLAY__DEBUGVIEW_COLOR_3, .fontSize = 16 }) : &Clay__DebugView_TextNameConfig;
03102                 CLAY(CLAY_LAYOUT({ .sizing = { .height =
    CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT)}, .childAlignment = { .y = CLAY_ALIGN_Y_CENTER } }),
    CLAY_RECTANGLE(CLAY__DEFAULT_STRUCT)) {
03103                     CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_INDENT_WIDTH +
    16), CLAY__DEFAULT_STRUCT} })) {}
03104                     CLAY_TEXT(CLAY_STRING("\""), rawTextConfig);
03105                     CLAY_TEXT(textElementData->text.length > 40 ? (CLAY__INIT(Clay_String) { .length =
    40, .chars = textElementData->text.chars }) : textElementData->text, rawTextConfig);
03106                     if (textElementData->text.length > 40) {
03107                         CLAY_TEXT(CLAY_STRING("..."), rawTextConfig);
03108                     }
03109                     CLAY_TEXT(CLAY_STRING("\""), rawTextConfig);
03110                 }
03111             } else if (currentElement->childrenOrTextContent.children.length > 0) {
03112                 Clay__OpenElement();
03113                 CLAY_LAYOUT({ .padding = { 8 } });
03114                 Clay__ElementPostConfiguration();
03115                 Clay__OpenElement();
03116                 CLAY_BORDER({ .left = { .width = 1, .color = CLAY__DEBUGVIEW_COLOR_3 }});
03117                 Clay__ElementPostConfiguration();
03118                 CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED( CLAY__DEBUGVIEW_INDENT_WIDTH),
    CLAY__DEFAULT_STRUCT}, .childAlignment = { .x = CLAY_ALIGN_X_RIGHT } })) {}
03119                 Clay__OpenElement();
03120                 CLAY_LAYOUT({ .layoutDirection = CLAY_TOP_TO_BOTTOM });
03121                 Clay__ElementPostConfiguration();
03122             }
03123
03124             layoutData.rowCount++;
03125             if (!(Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT) ||
    (currentElementData && currentElementData->debugData->collapsed))) {
03126                 for (int32_t i = currentElement->childrenOrTextContent.children.length - 1; i >= 0;
    --i) {
03127                     Clay__int32_tArray_Add(&dfsBuffer,
    currentElement->childrenOrTextContent.children.elements[i]);
03128                     context->treeNodeVisited.internalArray[dfsBuffer.length - 1] = false; // TODO
    needs to be ranged checked
03129                 }
03130             }
03131         }
03132     }
03133
03134     if (context->pointerInfo.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME) {
03135         Clay_ElementId collapseButtonId =
    Clay__HashString(CLAY_STRING("Clay__DebugView_CollapseElement"), 0, 0);
03136         for (int32_t i = (int)context->pointerOverIds.length - 1; i >= 0; i--) {
03137             Clay_ElementId *elementId = Clay__ElementIdArray_Get(&context->pointerOverIds, i);
03138             if (elementId->baseId == collapseButtonId.baseId) {
03139                 Clay_LayoutElementHashMapItem *highlightedItem =
    Clay__GetHashMapItem(elementId->offset);
03140                 highlightedItem->debugData->collapsed = !highlightedItem->debugData->collapsed;
03141                 break;
03142             }
03143         }
```

```
03144     }
03145
03146     if (highlightedElementId) {
03147         CLAY(CLAY_ID("Clay__DebugView_ElementHighlight"), CLAY_LAYOUT({ .sizing =
      {CLAY_SIZING_GROW(0), CLAY_SIZING_GROW(0)} }), CLAY_FLOATING({ .zIndex = 65535, .parentId =
      highlightedElementId })) {
03148             CLAY(CLAY_ID("Clay__DebugView_ElementHighlightRectangle"), CLAY_LAYOUT({ .sizing =
      {CLAY_SIZING_GROW(0), CLAY_SIZING_GROW(0)} }), CLAY_RECTANGLE({.color = Clay__debugViewHighlightColor
      })) {}
03149         }
03150     }
03151     return layoutData;
03152 }
03153
03154 void Clay__RenderDebugLayoutSizing(Clay_SizingAxis sizing, Clay_TextElementConfig *infoTextConfig) {
03155     Clay_String sizingLabel = CLAY_STRING("GROW");
03156     if (sizing.type == CLAY__SIZING_TYPE_FIT) {
03157         sizingLabel = CLAY_STRING("FIT");
03158     } else if (sizing.type == CLAY__SIZING_TYPE_PERCENT) {
03159         sizingLabel = CLAY_STRING("PERCENT");
03160     }
03161     CLAY_TEXT(sizingLabel, infoTextConfig);
03162     if (sizing.type == CLAY__SIZING_TYPE_GROW || sizing.type == CLAY__SIZING_TYPE_FIT) {
03163         CLAY_TEXT(CLAY_STRING("("), infoTextConfig);
03164         if (sizing.size.minMax.min != 0) {
03165             CLAY_TEXT(CLAY_STRING("min: "), infoTextConfig);
03166             CLAY_TEXT(Clay__IntToString(sizing.size.minMax.min), infoTextConfig);
03167             if (sizing.size.minMax.max != CLAY__MAXFLOAT) {
03168                 CLAY_TEXT(CLAY_STRING(", "), infoTextConfig);
03169             }
03170         }
03171         if (sizing.size.minMax.max != CLAY__MAXFLOAT) {
03172             CLAY_TEXT(CLAY_STRING("max: "), infoTextConfig);
03173             CLAY_TEXT(Clay__IntToString(sizing.size.minMax.max), infoTextConfig);
03174         }
03175         CLAY_TEXT(CLAY_STRING(")"), infoTextConfig);
03176     }
03177 }
03178
03179 void Clay__RenderDebugViewElementConfigHeader(Clay_String elementId, Clay__ElementConfigType type) {
03180     Clay__DebugElementConfigTypeLabelConfig config = Clay__DebugGetElementConfigTypeLabel(type);
03181     Clay_Color backgroundColor = config.color;
03182     backgroundColor.a = 90;
03183     CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_GROW(0)}, .padding =
      CLAY_PADDING_ALL(CLAY__DEBUGVIEW_OUTER_PADDING), .childAlignment = { .y = CLAY_ALIGN_Y_CENTER } })) {
03184         CLAY(CLAY_LAYOUT({ .padding = { 8, 8, 2, 2 } }), CLAY_RECTANGLE({ .color = backgroundColor,
      .cornerRadius = CLAY_CORNER_RADIUS(4) }), CLAY_BORDER_OUTSIDE_RADIUS(1, config.color, 4)) {
03185             CLAY_TEXT(config.label, CLAY_TEXT_CONFIG({ .textColor = CLAY__DEBUGVIEW_COLOR_4, .fontSize
      = 16 }));
03186         }
03187         CLAY(CLAY_LAYOUT({ .sizing = { .width = CLAY_SIZING_GROW(0) } })) {}
03188         CLAY_TEXT(elementId, CLAY_TEXT_CONFIG({ .textColor = CLAY__DEBUGVIEW_COLOR_3, .fontSize = 16,
      .wrapMode = CLAY_TEXT_WRAP_NONE }));
03189     }
03190 }
03191
03192 void Clay__RenderDebugViewColor(Clay_Color color, Clay_TextElementConfig *textConfig) {
03193     CLAY(CLAY_LAYOUT({ .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} })) {
03194         CLAY_TEXT(CLAY_STRING("{ r: "), textConfig);
03195         CLAY_TEXT(Clay__IntToString(color.r), textConfig);
03196         CLAY_TEXT(CLAY_STRING(", g: "), textConfig);
03197         CLAY_TEXT(Clay__IntToString(color.g), textConfig);
03198         CLAY_TEXT(CLAY_STRING(", b: "), textConfig);
03199         CLAY_TEXT(Clay__IntToString(color.b), textConfig);
03200         CLAY_TEXT(CLAY_STRING(", a: "), textConfig);
03201         CLAY_TEXT(Clay__IntToString(color.a), textConfig);
03202         CLAY_TEXT(CLAY_STRING(" }"), textConfig);
03203         CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_FIXED(10), CLAY__DEFAULT_STRUCT } })) {}
03204         CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT - 8),
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT - 8)} }), CLAY_BORDER_OUTSIDE_RADIUS(1,
      CLAY__DEBUGVIEW_COLOR_4, 4)) {
03205             CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT - 8),
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT - 8)} }), CLAY_RECTANGLE({ .color = color, .cornerRadius
      = CLAY_CORNER_RADIUS(4) })) {}
03206         }
03207     }
03208 }
03209
03210 void Clay__RenderDebugViewCornerRadius(Clay_CornerRadius cornerRadius, Clay_TextElementConfig
      *textConfig) {
03211     CLAY(CLAY_LAYOUT({ .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} })) {
03212         CLAY_TEXT(CLAY_STRING("{ topLeft: "), textConfig);
03213         CLAY_TEXT(Clay__IntToString(cornerRadius.topLeft), textConfig);
03214         CLAY_TEXT(CLAY_STRING(", topRight: "), textConfig);
03215         CLAY_TEXT(Clay__IntToString(cornerRadius.topRight), textConfig);
03216         CLAY_TEXT(CLAY_STRING(", bottomLeft: "), textConfig);
03217         CLAY_TEXT(Clay__IntToString(cornerRadius.bottomLeft), textConfig);
```

```
03218         CLAY_TEXT(CLAY_STRING(", bottomRight: "), textConfig);
03219         CLAY_TEXT(Clay__IntToString(cornerRadius.bottomRight), textConfig);
03220         CLAY_TEXT(CLAY_STRING(" }"), textConfig);
03221     }
03222 }
03223
03224 void Clay__RenderDebugViewBorder(int32_t index, Clay_Border border, Clay_TextElementConfig
      *textConfig) {
03225     (void) index;
03226     CLAY(CLAY_LAYOUT({ .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} })) {
03227         CLAY_TEXT(CLAY_STRING("{ width: "), textConfig);
03228         CLAY_TEXT(Clay__IntToString(border.width), textConfig);
03229         CLAY_TEXT(CLAY_STRING(", color: "), textConfig);
03230         Clay__RenderDebugViewColor(border.color, textConfig);
03231         CLAY_TEXT(CLAY_STRING(" }"), textConfig);
03232     }
03233 }
03234
03235 void HandleDebugViewCloseButtonInteraction(Clay_ElementId elementId, Clay_PointerData pointerInfo,
      intptr_t userData) {
03236     Clay_Context* context = Clay_GetCurrentContext();
03237     (void) elementId; (void) pointerInfo; (void) userData;
03238     if (pointerInfo.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME) {
03239         context->debugModeEnabled = false;
03240     }
03241 }
03242
03243 void Clay__RenderDebugView() {
03244     Clay_Context* context = Clay_GetCurrentContext();
03245     Clay_ElementId closeButtonId =
      Clay__HashString(CLAY_STRING("Clay__DebugViewTopHeaderCloseButtonOuter"), 0, 0);
03246     if (context->pointerInfo.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME) {
03247         for (int32_t i = 0; i < context->pointerOverIds.length; ++i) {
03248             Clay_ElementId *elementId = Clay__ElementIdArray_Get(&context->pointerOverIds, i);
03249             if (elementId->id == closeButtonId.id) {
03250                 context->debugModeEnabled = false;
03251                 return;
03252             }
03253         }
03254     }
03255
03256     uint32_t initialRootsLength = context->layoutElementTreeRoots.length;
03257     uint32_t initialElementsLength = context->layoutElements.length;
03258     Clay_TextElementConfig *infoTextConfig = CLAY_TEXT_CONFIG({ .textColor = CLAY__DEBUGVIEW_COLOR_4,
      .fontSize = 16, .wrapMode = CLAY_TEXT_WRAP_NONE });
03259     Clay_TextElementConfig *infoTitleConfig = CLAY_TEXT_CONFIG({ .textColor = CLAY__DEBUGVIEW_COLOR_3,
      .fontSize = 16, .wrapMode = CLAY_TEXT_WRAP_NONE });
03260     Clay_ElementId scrollId = Clay__HashString(CLAY_STRING("Clay__DebugViewOuterScrollPane"), 0, 0);
03261     float scrollYOffset = 0;
03262     bool pointerInDebugView = context->pointerInfo.position.y < context->layoutDimensions.height -
      300;
03263     for (int32_t i = 0; i < context->scrollContainerDatas.length; ++i) {
03264         Clay__ScrollContainerDataInternal *scrollContainerData =
      Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
03265         if (scrollContainerData->elementId == scrollId.id) {
03266             if (!context->externalScrollHandlingEnabled) {
03267                 scrollYOffset = scrollContainerData->scrollPosition.y;
03268             } else {
03269                 pointerInDebugView = context->pointerInfo.position.y +
      scrollContainerData->scrollPosition.y < context->layoutDimensions.height - 300;
03270             }
03271             break;
03272         }
03273     }
03274     int32_t highlightedRow = pointerInDebugView
03275             ? (int32_t)((context->pointerInfo.position.y - scrollYOffset) /
      (float)CLAY__DEBUGVIEW_ROW_HEIGHT) - 1
03276             : -1;
03277     if (context->pointerInfo.position.x < context->layoutDimensions.width -
      (float)Clay__debugViewWidth) {
03278         highlightedRow = -1;
03279     }
03280     Clay__RenderDebugLayoutData layoutData = CLAY__DEFAULT_STRUCT;
03281     CLAY(CLAY_ID("Clay__DebugView"),
03282         CLAY_FLOATING({ .zIndex = 65000, .parentId =
      Clay__HashString(CLAY_STRING("Clay__RootContainer"), 0, 0).id, .attachment = { .element =
      CLAY_ATTACH_POINT_LEFT_CENTER, .parent = CLAY_ATTACH_POINT_RIGHT_CENTER }}),
03283         CLAY_LAYOUT({ .sizing = { CLAY_SIZING_FIXED((float)Clay__debugViewWidth) ,
      CLAY_SIZING_FIXED(context->layoutDimensions.height) }, .layoutDirection = CLAY_TOP_TO_BOTTOM }),
03284         CLAY_BORDER({ .bottom = { .width = 1, .color = CLAY__DEBUGVIEW_COLOR_3 }})
03285     ) {
03286         CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0),
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT)}, .padding = {CLAY__DEBUGVIEW_OUTER_PADDING,
      CLAY__DEBUGVIEW_OUTER_PADDING }, .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} }), CLAY_RECTANGLE({
      .color = CLAY__DEBUGVIEW_COLOR_2 })) {
03287             CLAY_TEXT(CLAY_STRING("Clay Debug Tools"), infoTextConfig);
03288             CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_GROW(0) } })) {}
```

```
03289             // Close button
03290             CLAY(CLAY_BORDER_OUTSIDE_RADIUS(1, (CLAY__INIT(Clay_Color){217,91,67,255}), 4),
03291                 CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT - 10),
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT - 10)}, .childAlignment = {CLAY_ALIGN_X_CENTER,
      CLAY_ALIGN_Y_CENTER} }),
03292                 CLAY_RECTANGLE({ .color = {217,91,67,80} }),
03293                 Clay_OnHover(HandleDebugViewCloseButtonInteraction, 0)
03294             ) {
03295                 CLAY_TEXT(CLAY_STRING("x"), CLAY_TEXT_CONFIG({ .textColor = CLAY__DEBUGVIEW_COLOR_4,
      .fontSize = 16 }));
03296             }
03297         }
03298         CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0), CLAY_SIZING_FIXED(1)} }), CLAY_RECTANGLE({
      .color = CLAY__DEBUGVIEW_COLOR_3 })) {}
03299         CLAY(Clay__AttachId(scrollId), CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0),
      CLAY_SIZING_GROW(0)} }), CLAY_SCROLL({ .horizontal = true, .vertical = true })) {
03300             CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0), CLAY_SIZING_GROW(0)}, .layoutDirection
      = CLAY_TOP_TO_BOTTOM }), CLAY_RECTANGLE({ .color = ((initialElementsLength + initialRootsLength) & 1)
      == 0 ? CLAY__DEBUGVIEW_COLOR_2 : CLAY__DEBUGVIEW_COLOR_1 })) {
03301                 Clay_ElementId panelContentsId =
      Clay__HashString(CLAY_STRING("Clay__DebugViewPaneOuter"), 0, 0);
03302                 // Element list
03303                 CLAY(Clay__AttachId(panelContentsId), CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0),
      CLAY_SIZING_GROW(0)} }), CLAY_FLOATING({ .zIndex = 65001, .pointerCaptureMode =
      CLAY_POINTER_CAPTURE_MODE_PASSTHROUGH })) {
03304                     CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0), CLAY_SIZING_GROW(0)}, .padding
      = { CLAY__DEBUGVIEW_OUTER_PADDING, CLAY__DEBUGVIEW_OUTER_PADDING }, .layoutDirection =
      CLAY_TOP_TO_BOTTOM })) {
03305                         layoutData = Clay__RenderDebugLayoutElementsList((int32_t)initialRootsLength,
      highlightedRow);
03306                     }
03307                 }
03308                 float contentWidth =
      Clay__GetHashMapItem(panelContentsId.id)->layoutElement->dimensions.width;
03309                 CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED(contentWidth), CLAY__DEFAULT_STRUCT},
      .layoutDirection = CLAY_TOP_TO_BOTTOM })) {}
03310                 for (int32_t i = 0; i < layoutData.rowCount; i++) {
03311                     Clay_Color rowColor = (i & 1) == 0 ? CLAY__DEBUGVIEW_COLOR_2 :
      CLAY__DEBUGVIEW_COLOR_1;
03312                     if (i == layoutData.selectedElementRowIndex) {
03313                         rowColor = CLAY__DEBUGVIEW_COLOR_SELECTED_ROW;
03314                     }
03315                     if (i == highlightedRow) {
03316                         rowColor.r *= 1.25f;
03317                         rowColor.g *= 1.25f;
03318                         rowColor.b *= 1.25f;
03319                     }
03320                     CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0),
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT)}, .layoutDirection = CLAY_TOP_TO_BOTTOM }),
      CLAY_RECTANGLE({ .color = rowColor })) {}
03321                 }
03322             }
03323         }
03324         CLAY(CLAY_LAYOUT({ .sizing = {.width = CLAY_SIZING_GROW(0), .height = CLAY_SIZING_FIXED(1)}
      }), CLAY_RECTANGLE({ .color = CLAY__DEBUGVIEW_COLOR_3 })) {}
03325         if (context->debugSelectedElementId != 0) {
03326             Clay_LayoutElementHashMapItem *selectedItem =
      Clay__GetHashMapItem(context->debugSelectedElementId);
03327             CLAY(
03328                 CLAY_SCROLL({ .vertical = true }),
03329                 CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0), CLAY_SIZING_FIXED(300)},
      .layoutDirection = CLAY_TOP_TO_BOTTOM }),
03330                 CLAY_RECTANGLE({ .color = CLAY__DEBUGVIEW_COLOR_2 }),
03331                 CLAY_BORDER({ .betweenChildren = { .width = 1, .color = CLAY__DEBUGVIEW_COLOR_3 }})
03332             ) {
03333                 CLAY(CLAY_LAYOUT({ .sizing = {CLAY_SIZING_GROW(0),
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT + 8)}, .padding = {CLAY__DEBUGVIEW_OUTER_PADDING,
      CLAY__DEBUGVIEW_OUTER_PADDING}, .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} })) {
03334                     CLAY_TEXT(CLAY_STRING("Layout Config"), infoTextConfig);
03335                     CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_GROW(0) } })) {}
03336                     if (selectedItem->elementId.stringId.length != 0) {
03337                         CLAY_TEXT(selectedItem->elementId.stringId, infoTitleConfig);
03338                         if (selectedItem->elementId.offset != 0) {
03339                             CLAY_TEXT(CLAY_STRING(" ("), infoTitleConfig);
03340                             CLAY_TEXT(Clay__IntToString(selectedItem->elementId.offset),
      infoTitleConfig);
03341                             CLAY_TEXT(CLAY_STRING(")"), infoTitleConfig);
03342                         }
03343                     }
03344                 }
03345                 Clay_Padding attributeConfigPadding = {CLAY__DEBUGVIEW_OUTER_PADDING,
      CLAY__DEBUGVIEW_OUTER_PADDING, 8, 8};
03346                 // Clay_LayoutConfig debug info
03347                 CLAY(CLAY_LAYOUT({ .padding = attributeConfigPadding, .childGap = 8, .layoutDirection
      = CLAY_TOP_TO_BOTTOM })) {
03348                     // .boundingBox
03349                     CLAY_TEXT(CLAY_STRING("Bounding Box"), infoTitleConfig);
```

```
03350                     CLAY(CLAY_LAYOUT(CLAY__DEFAULT_STRUCT)) {
03351                         CLAY_TEXT(CLAY_STRING("{ x: "), infoTextConfig);
03352                         CLAY_TEXT(Clay__IntToString(selectedItem->boundingBox.x), infoTextConfig);
03353                         CLAY_TEXT(CLAY_STRING(", y: "), infoTextConfig);
03354                         CLAY_TEXT(Clay__IntToString(selectedItem->boundingBox.y), infoTextConfig);
03355                         CLAY_TEXT(CLAY_STRING(", width: "), infoTextConfig);
03356                         CLAY_TEXT(Clay__IntToString(selectedItem->boundingBox.width), infoTextConfig);
03357                         CLAY_TEXT(CLAY_STRING(", height: "), infoTextConfig);
03358                         CLAY_TEXT(Clay__IntToString(selectedItem->boundingBox.height),
      infoTextConfig);
03359                         CLAY_TEXT(CLAY_STRING(" }"), infoTextConfig);
03360                     }
03361                     // .layoutDirection
03362                     CLAY_TEXT(CLAY_STRING("Layout Direction"), infoTitleConfig);
03363                     Clay_LayoutConfig *layoutConfig = selectedItem->layoutElement->layoutConfig;
03364                     CLAY_TEXT(layoutConfig->layoutDirection == CLAY_TOP_TO_BOTTOM ?
      CLAY_STRING("TOP_TO_BOTTOM") : CLAY_STRING("LEFT_TO_RIGHT"), infoTextConfig);
03365                     // .sizing
03366                     CLAY_TEXT(CLAY_STRING("Sizing"), infoTitleConfig);
03367                     CLAY(CLAY_LAYOUT(CLAY__DEFAULT_STRUCT)) {
03368                         CLAY_TEXT(CLAY_STRING("width: "), infoTextConfig);
03369                         Clay__RenderDebugLayoutSizing(layoutConfig->sizing.width, infoTextConfig);
03370                     }
03371                     CLAY(CLAY_LAYOUT(CLAY__DEFAULT_STRUCT)) {
03372                         CLAY_TEXT(CLAY_STRING("height: "), infoTextConfig);
03373                         Clay__RenderDebugLayoutSizing(layoutConfig->sizing.height, infoTextConfig);
03374                     }
03375                     // .padding
03376                     CLAY_TEXT(CLAY_STRING("Padding"), infoTitleConfig);
03377                     CLAY(CLAY_ID("Clay__DebugViewElementInfoPadding")) {
03378                         CLAY_TEXT(CLAY_STRING("{ left: "), infoTextConfig);
03379                         CLAY_TEXT(Clay__IntToString(layoutConfig->padding.left), infoTextConfig);
03380                         CLAY_TEXT(CLAY_STRING(", right: "), infoTextConfig);
03381                         CLAY_TEXT(Clay__IntToString(layoutConfig->padding.right), infoTextConfig);
03382                         CLAY_TEXT(CLAY_STRING(", top: "), infoTextConfig);
03383                         CLAY_TEXT(Clay__IntToString(layoutConfig->padding.top), infoTextConfig);
03384                         CLAY_TEXT(CLAY_STRING(", bottom: "), infoTextConfig);
03385                         CLAY_TEXT(Clay__IntToString(layoutConfig->padding.bottom), infoTextConfig);
03386                         CLAY_TEXT(CLAY_STRING(" }"), infoTextConfig);
03387                     }
03388                     // .childGap
03389                     CLAY_TEXT(CLAY_STRING("Child Gap"), infoTitleConfig);
03390                     CLAY_TEXT(Clay__IntToString(layoutConfig->childGap), infoTextConfig);
03391                     // .childAlignment
03392                     CLAY_TEXT(CLAY_STRING("Child Alignment"), infoTitleConfig);
03393                     CLAY(CLAY_LAYOUT(CLAY__DEFAULT_STRUCT)) {
03394                         CLAY_TEXT(CLAY_STRING("{ x: "), infoTextConfig);
03395                         Clay_String alignX = CLAY_STRING("LEFT");
03396                         if (layoutConfig->childAlignment.x == CLAY_ALIGN_X_CENTER) {
03397                             alignX = CLAY_STRING("CENTER");
03398                         } else if (layoutConfig->childAlignment.x == CLAY_ALIGN_X_RIGHT) {
03399                             alignX = CLAY_STRING("RIGHT");
03400                         }
03401                         CLAY_TEXT(alignX, infoTextConfig);
03402                         CLAY_TEXT(CLAY_STRING(", y: "), infoTextConfig);
03403                         Clay_String alignY = CLAY_STRING("TOP");
03404                         if (layoutConfig->childAlignment.y == CLAY_ALIGN_Y_CENTER) {
03405                             alignY = CLAY_STRING("CENTER");
03406                         } else if (layoutConfig->childAlignment.y == CLAY_ALIGN_Y_BOTTOM) {
03407                             alignY = CLAY_STRING("BOTTOM");
03408                         }
03409                         CLAY_TEXT(alignY, infoTextConfig);
03410                         CLAY_TEXT(CLAY_STRING(" }"), infoTextConfig);
03411                     }
03412                 }
03413                 for (int32_t elementConfigIndex = 0; elementConfigIndex <
      selectedItem->layoutElement->elementConfigs.length; ++elementConfigIndex) {
03414                     Clay_ElementConfig *elementConfig =
      Clay__ElementConfigArraySlice_Get(&selectedItem->layoutElement->elementConfigs, elementConfigIndex);
03415                     Clay__RenderDebugViewElementConfigHeader(selectedItem->elementId.stringId,
      elementConfig->type);
03416                     switch (elementConfig->type) {
03417                         case CLAY__ELEMENT_CONFIG_TYPE_RECTANGLE: {
03418                             Clay_RectangleElementConfig *rectangleConfig =
      elementConfig->config.rectangleElementConfig;
03419                             CLAY(CLAY_LAYOUT({ .padding = attributeConfigPadding, .childGap = 8,
      .layoutDirection = CLAY_TOP_TO_BOTTOM })) {
03420                                 // .color
03421                                 CLAY_TEXT(CLAY_STRING("Color"), infoTitleConfig);
03422                                 Clay__RenderDebugViewColor(rectangleConfig->color, infoTextConfig);
03423                                 // .cornerRadius
03424                                 CLAY_TEXT(CLAY_STRING("Corner Radius"), infoTitleConfig);
03425                                 Clay__RenderDebugViewCornerRadius(rectangleConfig->cornerRadius,
      infoTextConfig);
03426                             }
03427                             break;
03428                         }
```

```
03429                         case CLAY__ELEMENT_CONFIG_TYPE_TEXT: {
03430                             Clay_TextElementConfig *textConfig =
    elementConfig->config.textElementConfig;
03431                             CLAY(CLAY_LAYOUT({ .padding = attributeConfigPadding, .childGap = 8,
    .layoutDirection = CLAY_TOP_TO_BOTTOM })) {
03432                                 // .fontSize
03433                                 CLAY_TEXT(CLAY_STRING("Font Size"), infoTitleConfig);
03434                                 CLAY_TEXT(Clay__IntToString(textConfig->fontSize), infoTextConfig);
03435                                 // .fontId
03436                                 CLAY_TEXT(CLAY_STRING("Font ID"), infoTitleConfig);
03437                                 CLAY_TEXT(Clay__IntToString(textConfig->fontId), infoTextConfig);
03438                                 // .lineHeight
03439                                 CLAY_TEXT(CLAY_STRING("Line Height"), infoTitleConfig);
03440                                 CLAY_TEXT(textConfig->lineHeight == 0 ? CLAY_STRING("auto") :
    Clay__IntToString(textConfig->lineHeight), infoTextConfig);
03441                                 // .letterSpacing
03442                                 CLAY_TEXT(CLAY_STRING("Letter Spacing"), infoTitleConfig);
03443                                 CLAY_TEXT(Clay__IntToString(textConfig->letterSpacing),
    infoTextConfig);
03444                                 // .lineSpacing
03445                                 CLAY_TEXT(CLAY_STRING("Wrap Mode"), infoTitleConfig);
03446                                 Clay_String wrapMode = CLAY_STRING("WORDS");
03447                                 if (textConfig->wrapMode == CLAY_TEXT_WRAP_NONE) {
03448                                     wrapMode = CLAY_STRING("NONE");
03449                                 } else if (textConfig->wrapMode == CLAY_TEXT_WRAP_NEWLINES) {
03450                                     wrapMode = CLAY_STRING("NEWLINES");
03451                                 }
03452                                 CLAY_TEXT(wrapMode, infoTextConfig);
03453                                 // .textColor
03454                                 CLAY_TEXT(CLAY_STRING("Text Color"), infoTitleConfig);
03455                                 Clay__RenderDebugViewColor(textConfig->textColor, infoTextConfig);
03456                             }
03457                             break;
03458                         }
03459                         case CLAY__ELEMENT_CONFIG_TYPE_IMAGE: {
03460                             Clay_ImageElementConfig *imageConfig =
    elementConfig->config.imageElementConfig;
03461                             CLAY(CLAY_ID("Clay__DebugViewElementInfoImageBody"), CLAY_LAYOUT({
    .padding = attributeConfigPadding, .childGap = 8, .layoutDirection = CLAY_TOP_TO_BOTTOM })) {
03462                                 // .sourceDimensions
03463                                 CLAY_TEXT(CLAY_STRING("Source Dimensions"), infoTitleConfig);
03464                                 CLAY(CLAY_ID("Clay__DebugViewElementInfoImageDimensions")) {
03465                                     CLAY_TEXT(CLAY_STRING("{ width: "), infoTextConfig);
03466                                     CLAY_TEXT(Clay__IntToString(imageConfig->sourceDimensions.width),
    infoTextConfig);
03467                                     CLAY_TEXT(CLAY_STRING(", height: "), infoTextConfig);
03468                                     CLAY_TEXT(Clay__IntToString(imageConfig->sourceDimensions.height),
    infoTextConfig);
03469                                     CLAY_TEXT(CLAY_STRING(" }"), infoTextConfig);
03470                                 }
03471                                 // Image Preview
03472                                 CLAY_TEXT(CLAY_STRING("Preview"), infoTitleConfig);
03473                                 CLAY(CLAY_LAYOUT({ .sizing = { CLAY_SIZING_GROW(0,
    imageConfig->sourceDimensions.width) }}),
    Clay__AttachElementConfig(CLAY__INIT(Clay_ElementConfigUnion) { .imageElementConfig = imageConfig },
    CLAY__ELEMENT_CONFIG_TYPE_IMAGE)) {}
03474                             }
03475                             break;
03476                         }
03477                         case CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER: {
03478                             Clay_ScrollElementConfig *scrollConfig =
    elementConfig->config.scrollElementConfig;
03479                             CLAY(CLAY_LAYOUT({ .padding = attributeConfigPadding, .childGap = 8,
    .layoutDirection = CLAY_TOP_TO_BOTTOM })) {
03480                                 // .vertical
03481                                 CLAY_TEXT(CLAY_STRING("Vertical"), infoTitleConfig);
03482                                 CLAY_TEXT(scrollConfig->vertical ? CLAY_STRING("true") :
    CLAY_STRING("false") , infoTextConfig);
03483                                 // .horizontal
03484                                 CLAY_TEXT(CLAY_STRING("Horizontal"), infoTitleConfig);
03485                                 CLAY_TEXT(scrollConfig->horizontal ? CLAY_STRING("true") :
    CLAY_STRING("false") , infoTextConfig);
03486                             }
03487                             break;
03488                         }
03489                         case CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER: {
03490                             Clay_FloatingElementConfig *floatingConfig =
    elementConfig->config.floatingElementConfig;
03491                             CLAY(CLAY_LAYOUT({ .padding = attributeConfigPadding, .childGap = 8,
    .layoutDirection = CLAY_TOP_TO_BOTTOM })) {
03492                                 // .offset
03493                                 CLAY_TEXT(CLAY_STRING("Offset"), infoTitleConfig);
03494                                 CLAY(CLAY_LAYOUT(CLAY__DEFAULT_STRUCT)) {
03495                                     CLAY_TEXT(CLAY_STRING("{ x: "), infoTextConfig);
03496                                     CLAY_TEXT(Clay__IntToString(floatingConfig->offset.x),
    infoTextConfig);
03497                                     CLAY_TEXT(CLAY_STRING(", y: "), infoTextConfig);
```

```
03498                                   CLAY_TEXT(Clay__IntToString(floatingConfig->offset.y),
      infoTextConfig);
03499                                   CLAY_TEXT(CLAY_STRING(" }"), infoTextConfig);
03500                               }
03501                               // .expand
03502                               CLAY_TEXT(CLAY_STRING("Expand"), infoTitleConfig);
03503                               CLAY(CLAY_LAYOUT(CLAY__DEFAULT_STRUCT)) {
03504                                   CLAY_TEXT(CLAY_STRING("{ width: "), infoTextConfig);
03505                                   CLAY_TEXT(Clay__IntToString(floatingConfig->expand.width),
      infoTextConfig);
03506                                   CLAY_TEXT(CLAY_STRING(", height: "), infoTextConfig);
03507                                   CLAY_TEXT(Clay__IntToString(floatingConfig->expand.height),
      infoTextConfig);
03508                                   CLAY_TEXT(CLAY_STRING(" }"), infoTextConfig);
03509                               }
03510                               // .zIndex
03511                               CLAY_TEXT(CLAY_STRING("z-index"), infoTitleConfig);
03512                               CLAY_TEXT(Clay__IntToString(floatingConfig->zIndex), infoTextConfig);
03513                               // .parentId
03514                               CLAY_TEXT(CLAY_STRING("Parent"), infoTitleConfig);
03515                               Clay_LayoutElementHashMapItem *hashItem =
      Clay__GetHashMapItem(floatingConfig->parentId);
03516                               CLAY_TEXT(hashItem->elementId.stringId, infoTextConfig);
03517                           }
03518                           break;
03519                       }
03520                       case CLAY__ELEMENT_CONFIG_TYPE_BORDER_CONTAINER: {
03521                           Clay_BorderElementConfig *borderConfig =
      elementConfig->config.borderElementConfig;
03522                           CLAY(CLAY_ID("Clay__DebugViewElementInfoBorderBody"), CLAY_LAYOUT({
      .padding = attributeConfigPadding, .childGap = 8, .layoutDirection = CLAY_TOP_TO_BOTTOM })) {
03523                               // .left
03524                               CLAY_TEXT(CLAY_STRING("Left Border"), infoTitleConfig);
03525                               Clay__RenderDebugViewBorder(1, borderConfig->left, infoTextConfig);
03526                               // .right
03527                               CLAY_TEXT(CLAY_STRING("Right Border"), infoTitleConfig);
03528                               Clay__RenderDebugViewBorder(2, borderConfig->right, infoTextConfig);
03529                               // .top
03530                               CLAY_TEXT(CLAY_STRING("Top Border"), infoTitleConfig);
03531                               Clay__RenderDebugViewBorder(3, borderConfig->top, infoTextConfig);
03532                               // .bottom
03533                               CLAY_TEXT(CLAY_STRING("Bottom Border"), infoTitleConfig);
03534                               Clay__RenderDebugViewBorder(4, borderConfig->bottom, infoTextConfig);
03535                               // .betweenChildren
03536                               CLAY_TEXT(CLAY_STRING("Border Between Children"), infoTitleConfig);
03537                               Clay__RenderDebugViewBorder(5, borderConfig->betweenChildren,
      infoTextConfig);
03538                               // .cornerRadius
03539                               CLAY_TEXT(CLAY_STRING("Corner Radius"), infoTitleConfig);
03540                               Clay__RenderDebugViewCornerRadius(borderConfig->cornerRadius,
      infoTextConfig);
03541                           }
03542                           break;
03543                       }
03544                       case CLAY__ELEMENT_CONFIG_TYPE_CUSTOM:
03545                       default: break;
03546                   }
03547               }
03548           }
03549       } else {
03550           CLAY(CLAY_ID("Clay__DebugViewWarningsScrollPane"), CLAY_LAYOUT({ .sizing =
      {CLAY_SIZING_GROW(0), CLAY_SIZING_FIXED(300)}, .childGap = 6, .layoutDirection = CLAY_TOP_TO_BOTTOM
      }), CLAY_SCROLL({ .horizontal = true, .vertical = true }), CLAY_RECTANGLE({ .color =
      CLAY__DEBUGVIEW_COLOR_2 })) {
03551               Clay_TextElementConfig *warningConfig = CLAY_TEXT_CONFIG({ .textColor =
      CLAY__DEBUGVIEW_COLOR_4, .fontSize = 16, .wrapMode = CLAY_TEXT_WRAP_NONE });
03552               CLAY(CLAY_ID("Clay__DebugViewWarningItemHeader"), CLAY_LAYOUT({ .sizing = {.height =
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT)}, .padding = {CLAY__DEBUGVIEW_OUTER_PADDING,
      CLAY__DEBUGVIEW_OUTER_PADDING}, .childGap = 8, .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} })) {
03553                   CLAY_TEXT(CLAY_STRING("Warnings"), warningConfig);
03554               }
03555               CLAY(CLAY_ID("Clay__DebugViewWarningsTopBorder"), CLAY_LAYOUT({ .sizing = { .width =
      CLAY_SIZING_GROW(0), .height = CLAY_SIZING_FIXED(1)} }), CLAY_RECTANGLE({ .color = {200, 200, 200,
      255} })) {}
03556               int32_t previousWarningsLength = context->warnings.length;
03557               for (int32_t i = 0; i < previousWarningsLength; i++) {
03558                   Clay__Warning warning = context->warnings.internalArray[i];
03559                   CLAY(CLAY_IDI("Clay__DebugViewWarningItem", i), CLAY_LAYOUT({ .sizing = {.height =
      CLAY_SIZING_FIXED(CLAY__DEBUGVIEW_ROW_HEIGHT)}, .padding = {CLAY__DEBUGVIEW_OUTER_PADDING,
      CLAY__DEBUGVIEW_OUTER_PADDING}, .childGap = 8, .childAlignment = {.y = CLAY_ALIGN_Y_CENTER} })) {
03560                       CLAY_TEXT(warning.baseMessage, warningConfig);
03561                       if (warning.dynamicMessage.length > 0) {
03562                           CLAY_TEXT(warning.dynamicMessage, warningConfig);
03563                       }
03564                   }
03565               }
03566           }
```

```
03567         }
03568     }
03569 }
03570 #pragma endregion
03571
03572 uint32_t Clay__debugViewWidth = 400;
03573 Clay_Color Clay__debugViewHighlightColor = { 168, 66, 28, 100 };
03574
03575 Clay__WarningArray Clay__WarningArray_Allocate_Arena(int32_t capacity, Clay_Arena *arena) {
03576     size_t totalSizeBytes = capacity * sizeof(Clay_String);
03577     Clay__WarningArray array = {.capacity = capacity, .length = 0};
03578     uintptr_t nextAllocAddress = arena->nextAllocation + (uintptr_t)arena->memory;
03579     uintptr_t arenaOffsetAligned = nextAllocAddress + (CLAY__ALIGNMENT(Clay_String) -
      (nextAllocAddress % CLAY__ALIGNMENT(Clay_String)));
03580     arenaOffsetAligned -= (uintptr_t)arena->memory;
03581     if (arenaOffsetAligned + totalSizeBytes <= arena->capacity) {
03582         array.internalArray = (Clay__Warning*)((uintptr_t)arena->memory +
      (uintptr_t)arenaOffsetAligned);
03583         arena->nextAllocation = arenaOffsetAligned + totalSizeBytes;
03584     }
03585     else {
03586         Clay__currentContext->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
03587             .errorType = CLAY_ERROR_TYPE_ARENA_CAPACITY_EXCEEDED,
03588             .errorText = CLAY_STRING("Clay attempted to allocate memory in its arena, but ran out of
      capacity. Try increasing the capacity of the arena passed to Clay_Initialize()"),
03589             .userData = Clay__currentContext->errorHandler.userData });
03590     }
03591     return array;
03592 }
03593
03594 Clay__Warning *Clay__WarningArray_Add(Clay__WarningArray *array, Clay__Warning item)
03595 {
03596     if (array->length < array->capacity) {
03597         array->internalArray[array->length++] = item;
03598         return &array->internalArray[array->length - 1];
03599     }
03600     return &CLAY__WARNING_DEFAULT;
03601 }
03602
03603 void* Clay__Array_Allocate_Arena(int32_t capacity, uint32_t itemSize, uint32_t alignment, Clay_Arena
      *arena)
03604 {
03605     size_t totalSizeBytes = capacity * itemSize;
03606     uintptr_t nextAllocAddress = arena->nextAllocation + (uintptr_t)arena->memory;
03607     uintptr_t arenaOffsetAligned = nextAllocAddress + (alignment - (nextAllocAddress % alignment));
03608     arenaOffsetAligned -= (uintptr_t)arena->memory;
03609     if (arenaOffsetAligned + totalSizeBytes <= arena->capacity) {
03610         arena->nextAllocation = arenaOffsetAligned + totalSizeBytes;
03611         return (void*)((uintptr_t)arena->memory + (uintptr_t)arenaOffsetAligned);
03612     }
03613     else {
03614         Clay__currentContext->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
03615                 .errorType = CLAY_ERROR_TYPE_ARENA_CAPACITY_EXCEEDED,
03616                 .errorText = CLAY_STRING("Clay attempted to allocate memory in its arena, but ran out
      of capacity. Try increasing the capacity of the arena passed to Clay_Initialize()"),
03617                 .userData = Clay__currentContext->errorHandler.userData });
03618     }
03619     return CLAY__NULL;
03620 }
03621
03622 bool Clay__Array_RangeCheck(int32_t index, int32_t length)
03623 {
03624     if (index < length && index >= 0) {
03625         return true;
03626     }
03627     Clay_Context* context = Clay_GetCurrentContext();
03628     context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
03629             .errorType = CLAY_ERROR_TYPE_INTERNAL_ERROR,
03630             .errorText = CLAY_STRING("Clay attempted to make an out of bounds array access. This is an
      internal error and is likely a bug."),
03631             .userData = context->errorHandler.userData });
03632     return false;
03633 }
03634
03635 bool Clay__Array_AddCapacityCheck(int32_t length, int32_t capacity)
03636 {
03637     if (length < capacity) {
03638         return true;
03639     }
03640     Clay_Context* context = Clay_GetCurrentContext();
03641     context->errorHandler.errorHandlerFunction(CLAY__INIT(Clay_ErrorData) {
03642         .errorType = CLAY_ERROR_TYPE_INTERNAL_ERROR,
03643         .errorText = CLAY_STRING("Clay attempted to make an out of bounds array access. This is an
      internal error and is likely a bug."),
03644         .userData = context->errorHandler.userData });
03645     return false;
03646 }
```

```
03647
03648 // PUBLIC API FROM HERE ----------------------------------------
03649
03650 CLAY_WASM_EXPORT("Clay_MinMemorySize")
03651 uint32_t Clay_MinMemorySize(void) {
03652     Clay_Context fakeContext = {
03653         .maxElementCount = Clay__defaultMaxElementCount,
03654         .maxMeasureTextCacheWordCount = Clay__defaultMaxMeasureTextWordCacheCount,
03655         .internalArena = {
03656             .capacity = SIZE_MAX,
03657             .memory = NULL,
03658         }
03659     };
03660     Clay_Context* currentContext = Clay_GetCurrentContext();
03661     if (currentContext) {
03662         fakeContext.maxElementCount = currentContext->maxElementCount;
03663         fakeContext.maxMeasureTextCacheWordCount = currentContext->maxElementCount;
03664     }
03665     // Reserve space in the arena for the context, important for calculating min memory size correctly
03666     Clay__Context_Allocate_Arena(&fakeContext.internalArena);
03667     Clay__InitializePersistentMemory(&fakeContext);
03668     Clay__InitializeEphemeralMemory(&fakeContext);
03669     return fakeContext.internalArena.nextAllocation;
03670 }
03671
03672 CLAY_WASM_EXPORT("Clay_CreateArenaWithCapacityAndMemory")
03673 Clay_Arena Clay_CreateArenaWithCapacityAndMemory(uint32_t capacity, void *offset) {
03674     Clay_Arena arena = {
03675         .capacity = capacity,
03676         .memory = (char *)offset
03677     };
03678     return arena;
03679 }
03680
03681 #ifndef CLAY_WASM
03682 void Clay_SetMeasureTextFunction(Clay_Dimensions (*measureTextFunction)(Clay_StringSlice text,
    Clay_TextElementConfig *config, uintptr_t userData), uintptr_t userData) {
03683     Clay_Context* context = Clay_GetCurrentContext();
03684     Clay__MeasureText = measureTextFunction;
03685     context->mesureTextUserData = userData;
03686 }
03687 void Clay_SetQueryScrollOffsetFunction(Clay_Vector2 (*queryScrollOffsetFunction)(uint32_t elementId,
    uintptr_t userData), uintptr_t userData) {
03688     Clay_Context* context = Clay_GetCurrentContext();
03689     Clay__QueryScrollOffset = queryScrollOffsetFunction;
03690     context->queryScrollOffsetUserData = userData;
03691 }
03692 #endif
03693
03694 CLAY_WASM_EXPORT("Clay_SetLayoutDimensions")
03695 void Clay_SetLayoutDimensions(Clay_Dimensions dimensions) {
03696     Clay_GetCurrentContext()->layoutDimensions = dimensions;
03697 }
03698
03699 CLAY_WASM_EXPORT("Clay_SetPointerState")
03700 void Clay_SetPointerState(Clay_Vector2 position, bool isPointerDown) {
03701     Clay_Context* context = Clay_GetCurrentContext();
03702     if (context->booleanWarnings.maxElementsExceeded) {
03703         return;
03704     }
03705     context->pointerInfo.position = position;
03706     context->pointerOverIds.length = 0;
03707     Clay__int32_tArray dfsBuffer = context->layoutElementChildrenBuffer;
03708     for (int32_t rootIndex = context->layoutElementTreeRoots.length - 1; rootIndex >= 0; --rootIndex)
    {
03709         dfsBuffer.length = 0;
03710         Clay__LayoutElementTreeRoot *root =
    Clay__LayoutElementTreeRootArray_Get(&context->layoutElementTreeRoots, rootIndex);
03711         Clay__int32_tArray_Add(&dfsBuffer, (int32_t)root->layoutElementIndex);
03712         context->treeNodeVisited.internalArray[0] = false;
03713         bool found = false;
03714         while (dfsBuffer.length > 0) {
03715             if (context->treeNodeVisited.internalArray[dfsBuffer.length - 1]) {
03716                 dfsBuffer.length--;
03717                 continue;
03718             }
03719             context->treeNodeVisited.internalArray[dfsBuffer.length - 1] = true;
03720             Clay_LayoutElement *currentElement = Clay_LayoutElementArray_Get(&context->layoutElements,
    Clay__int32_tArray_Get(&dfsBuffer, (int)dfsBuffer.length - 1));
03721             Clay_LayoutElementHashMapItem *mapItem = Clay__GetHashMapItem(currentElement->id); // TODO
    think of a way around this, maybe the fact that it's essentially a binary tree limits the cost, but
    the worst case is not great
03722             Clay_BoundingBox elementBox = mapItem->boundingBox;
03723             elementBox.x -= root->pointerOffset.x;
03724             elementBox.y -= root->pointerOffset.y;
03725             if (mapItem) {
03726                 if ((Clay__PointIsInsideRect(position, elementBox))) {
```

```
03727                     if (mapItem->onHoverFunction) {
03728                         mapItem->onHoverFunction(mapItem->elementId, context->pointerInfo,
      mapItem->hoverFunctionUserData);
03729                     }
03730                     Clay__ElementIdArray_Add(&context->pointerOverIds, mapItem->elementId);
03731                     found = true;
03732                 }
03733                 if (Clay__ElementHasConfig(currentElement, CLAY__ELEMENT_CONFIG_TYPE_TEXT)) {
03734                     dfsBuffer.length--;
03735                     continue;
03736                 }
03737                 for (int32_t i = currentElement->childrenOrTextContent.children.length - 1; i >= 0;
      --i) {
03738                     Clay__int32_tArray_Add(&dfsBuffer,
      currentElement->childrenOrTextContent.children.elements[i]);
03739                     context->treeNodeVisited.internalArray[dfsBuffer.length - 1] = false; // TODO
      needs to be ranged checked
03740                 }
03741             } else {
03742                 dfsBuffer.length--;
03743             }
03744         }
03745
03746         Clay_LayoutElement *rootElement = Clay_LayoutElementArray_Get(&context->layoutElements,
      root->layoutElementIndex);
03747         if (found && Clay__ElementHasConfig(rootElement, CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER)
      &&
03748                 Clay__FindElementConfigWithType(rootElement,
      CLAY__ELEMENT_CONFIG_TYPE_FLOATING_CONTAINER).floatingElementConfig->pointerCaptureMode ==
      CLAY_POINTER_CAPTURE_MODE_CAPTURE) {
03749             break;
03750         }
03751     }
03752
03753     if (isPointerDown) {
03754         if (context->pointerInfo.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME) {
03755             context->pointerInfo.state = CLAY_POINTER_DATA_PRESSED;
03756         } else if (context->pointerInfo.state != CLAY_POINTER_DATA_PRESSED) {
03757             context->pointerInfo.state = CLAY_POINTER_DATA_PRESSED_THIS_FRAME;
03758         }
03759     } else {
03760         if (context->pointerInfo.state == CLAY_POINTER_DATA_RELEASED_THIS_FRAME) {
03761             context->pointerInfo.state = CLAY_POINTER_DATA_RELEASED;
03762         } else if (context->pointerInfo.state != CLAY_POINTER_DATA_RELEASED)  {
03763             context->pointerInfo.state = CLAY_POINTER_DATA_RELEASED_THIS_FRAME;
03764         }
03765     }
03766 }
03767
03768 CLAY_WASM_EXPORT("Clay_Initialize")
03769 Clay_Context* Clay_Initialize(Clay_Arena arena, Clay_Dimensions layoutDimensions, Clay_ErrorHandler
      errorHandler) {
03770     Clay_Context *context = Clay__Context_Allocate_Arena(&arena);
03771     if (context == NULL) return NULL;
03772     // DEFAULTS
03773     Clay_Context *oldContext = Clay_GetCurrentContext();
03774     *context = CLAY__INIT(Clay_Context) {
03775         .maxElementCount = oldContext ? oldContext->maxElementCount : Clay__defaultMaxElementCount,
03776         .maxMeasureTextCacheWordCount = oldContext ? oldContext->maxMeasureTextCacheWordCount :
      Clay__defaultMaxMeasureTextWordCacheCount,
03777         .errorHandler = errorHandler.errorHandlerFunction ? errorHandler :
      CLAY__INIT(Clay_ErrorHandler) { Clay__ErrorHandlerFunctionDefault },
03778         .layoutDimensions = layoutDimensions,
03779         .internalArena = arena,
03780     };
03781     Clay_SetCurrentContext(context);
03782     Clay__InitializePersistentMemory(context);
03783     Clay__InitializeEphemeralMemory(context);
03784     for (int32_t i = 0; i < context->layoutElementsHashMap.capacity; ++i) {
03785         context->layoutElementsHashMap.internalArray[i] = -1;
03786     }
03787     for (int32_t i = 0; i < context->measureTextHashMap.capacity; ++i) {
03788         context->measureTextHashMap.internalArray[i] = 0;
03789     }
03790     context->measureTextHashMapInternal.length = 1; // Reserve the 0 value to mean "no next element"
03791     context->layoutDimensions = layoutDimensions;
03792     return context;
03793 }
03794
03795 CLAY_WASM_EXPORT("Clay_GetCurrentContext")
03796 Clay_Context* Clay_GetCurrentContext(void) {
03797     return Clay__currentContext;
03798 }
03799
03800 CLAY_WASM_EXPORT("Clay_SetCurrentContext")
03801 void Clay_SetCurrentContext(Clay_Context* context) {
03802     Clay__currentContext = context;
```

```
03803 }
03804
03805 CLAY_WASM_EXPORT("Clay_UpdateScrollContainers")
03806 void Clay_UpdateScrollContainers(bool enableDragScrolling, Clay_Vector2 scrollDelta, float deltaTime)
      {
03807     Clay_Context* context = Clay_GetCurrentContext();
03808     bool isPointerActive = enableDragScrolling && (context->pointerInfo.state ==
      CLAY_POINTER_DATA_PRESSED || context->pointerInfo.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME);
03809     // Don't apply scroll events to ancestors of the inner element
03810     int32_t highestPriorityElementIndex = -1;
03811     Clay__ScrollContainerDataInternal *highestPriorityScrollData = CLAY__NULL;
03812     for (int32_t i = 0; i < context->scrollContainerDatas.length; i++) {
03813         Clay__ScrollContainerDataInternal *scrollData =
      Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
03814         if (!scrollData->openThisFrame) {
03815             Clay__ScrollContainerDataInternalArray_RemoveSwapback(&context->scrollContainerDatas, i);
03816             continue;
03817         }
03818         scrollData->openThisFrame = false;
03819         Clay_LayoutElementHashMapItem *hashMapItem = Clay__GetHashMapItem(scrollData->elementId);
03820         // Element isn't rendered this frame but scroll offset has been retained
03821         if (!hashMapItem) {
03822             Clay__ScrollContainerDataInternalArray_RemoveSwapback(&context->scrollContainerDatas, i);
03823             continue;
03824         }
03825
03826         // Touch / click is released
03827         if (!isPointerActive && scrollData->pointerScrollActive) {
03828             float xDiff = scrollData->scrollPosition.x - scrollData->scrollOrigin.x;
03829             if (xDiff < -10 || xDiff > 10) {
03830                 scrollData->scrollMomentum.x = (scrollData->scrollPosition.x -
      scrollData->scrollOrigin.x) / (scrollData->momentumTime * 25);
03831             }
03832             float yDiff = scrollData->scrollPosition.y - scrollData->scrollOrigin.y;
03833             if (yDiff < -10 || yDiff > 10) {
03834                 scrollData->scrollMomentum.y = (scrollData->scrollPosition.y -
      scrollData->scrollOrigin.y) / (scrollData->momentumTime * 25);
03835             }
03836             scrollData->pointerScrollActive = false;
03837
03838             scrollData->pointerOrigin = CLAY__INIT(Clay_Vector2){0,0};
03839             scrollData->scrollOrigin = CLAY__INIT(Clay_Vector2){0,0};
03840             scrollData->momentumTime = 0;
03841         }
03842
03843         // Apply existing momentum
03844         scrollData->scrollPosition.x += scrollData->scrollMomentum.x;
03845         scrollData->scrollMomentum.x *= 0.95f;
03846         bool scrollOccurred = scrollDelta.x != 0 || scrollDelta.y != 0;
03847         if ((scrollData->scrollMomentum.x > -0.1f && scrollData->scrollMomentum.x < 0.1f) ||
      scrollOccurred) {
03848             scrollData->scrollMomentum.x = 0;
03849         }
03850         scrollData->scrollPosition.x = CLAY__MIN(CLAY__MAX(scrollData->scrollPosition.x,
      -(CLAY__MAX(scrollData->contentSize.width - scrollData->layoutElement->dimensions.width, 0))), 0);
03851
03852         scrollData->scrollPosition.y += scrollData->scrollMomentum.y;
03853         scrollData->scrollMomentum.y *= 0.95f;
03854         if ((scrollData->scrollMomentum.y > -0.1f && scrollData->scrollMomentum.y < 0.1f) ||
      scrollOccurred) {
03855             scrollData->scrollMomentum.y = 0;
03856         }
03857         scrollData->scrollPosition.y = CLAY__MIN(CLAY__MAX(scrollData->scrollPosition.y,
      -(CLAY__MAX(scrollData->contentSize.height - scrollData->layoutElement->dimensions.height, 0))), 0);
03858
03859         for (int32_t j = 0; j < context->pointerOverIds.length; ++j) { // TODO n & m are small here
      but this being n*m gives me the creeps
03860             if (scrollData->layoutElement->id == Clay__ElementIdArray_Get(&context->pointerOverIds,
      j)->id) {
03861                 highestPriorityElementIndex = j;
03862                 highestPriorityScrollData = scrollData;
03863             }
03864         }
03865     }
03866
03867     if (highestPriorityElementIndex > -1 && highestPriorityScrollData) {
03868         Clay_LayoutElement *scrollElement = highestPriorityScrollData->layoutElement;
03869         Clay_ScrollElementConfig *scrollConfig = Clay__FindElementConfigWithType(scrollElement,
      CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig;
03870         bool canScrollVertically = scrollConfig->vertical &&
      highestPriorityScrollData->contentSize.height > scrollElement->dimensions.height;
03871         bool canScrollHorizontally = scrollConfig->horizontal &&
      highestPriorityScrollData->contentSize.width > scrollElement->dimensions.width;
03872         // Handle wheel scroll
03873         if (canScrollVertically) {
03874             highestPriorityScrollData->scrollPosition.y = highestPriorityScrollData->scrollPosition.y
      + scrollDelta.y * 10;
```

```
03875         }
03876         if (canScrollHorizontally) {
03877             highestPriorityScrollData->scrollPosition.x = highestPriorityScrollData->scrollPosition.x
      + scrollDelta.x * 10;
03878         }
03879         // Handle click / touch scroll
03880         if (isPointerActive) {
03881             highestPriorityScrollData->scrollMomentum = CLAY__INIT(Clay_Vector2)CLAY__DEFAULT_STRUCT;
03882             if (!highestPriorityScrollData->pointerScrollActive) {
03883                 highestPriorityScrollData->pointerOrigin = context->pointerInfo.position;
03884                 highestPriorityScrollData->scrollOrigin = highestPriorityScrollData->scrollPosition;
03885                 highestPriorityScrollData->pointerScrollActive = true;
03886             } else {
03887                 float scrollDeltaX = 0, scrollDeltaY = 0;
03888                 if (canScrollHorizontally) {
03889                     float oldXScrollPosition = highestPriorityScrollData->scrollPosition.x;
03890                     highestPriorityScrollData->scrollPosition.x =
      highestPriorityScrollData->scrollOrigin.x + (context->pointerInfo.position.x -
      highestPriorityScrollData->pointerOrigin.x);
03891                     highestPriorityScrollData->scrollPosition.x =
      CLAY__MAX(CLAY__MIN(highestPriorityScrollData->scrollPosition.x, 0),
      -(highestPriorityScrollData->contentSize.width - highestPriorityScrollData->boundingBox.width));
03892                     scrollDeltaX = highestPriorityScrollData->scrollPosition.x - oldXScrollPosition;
03893                 }
03894                 if (canScrollVertically) {
03895                     float oldYScrollPosition = highestPriorityScrollData->scrollPosition.y;
03896                     highestPriorityScrollData->scrollPosition.y =
      highestPriorityScrollData->scrollOrigin.y + (context->pointerInfo.position.y -
      highestPriorityScrollData->pointerOrigin.y);
03897                     highestPriorityScrollData->scrollPosition.y =
      CLAY__MAX(CLAY__MIN(highestPriorityScrollData->scrollPosition.y, 0),
      -(highestPriorityScrollData->contentSize.height - highestPriorityScrollData->boundingBox.height));
03898                     scrollDeltaY = highestPriorityScrollData->scrollPosition.y - oldYScrollPosition;
03899                 }
03900                 if (scrollDeltaX > -0.1f && scrollDeltaX < 0.1f && scrollDeltaY > -0.1f &&
      scrollDeltaY < 0.1f && highestPriorityScrollData->momentumTime > 0.15f) {
03901                     highestPriorityScrollData->momentumTime = 0;
03902                     highestPriorityScrollData->pointerOrigin = context->pointerInfo.position;
03903                     highestPriorityScrollData->scrollOrigin =
      highestPriorityScrollData->scrollPosition;
03904                 } else {
03905                     highestPriorityScrollData->momentumTime += deltaTime;
03906                 }
03907             }
03908         }
03909         // Clamp any changes to scroll position to the maximum size of the contents
03910         if (canScrollVertically) {
03911             highestPriorityScrollData->scrollPosition.y =
      CLAY__MAX(CLAY__MIN(highestPriorityScrollData->scrollPosition.y, 0),
      -(highestPriorityScrollData->contentSize.height - scrollElement->dimensions.height));
03912         }
03913         if (canScrollHorizontally) {
03914             highestPriorityScrollData->scrollPosition.x =
      CLAY__MAX(CLAY__MIN(highestPriorityScrollData->scrollPosition.x, 0),
      -(highestPriorityScrollData->contentSize.width - scrollElement->dimensions.width));
03915         }
03916     }
03917 }
03918
03919 CLAY_WASM_EXPORT("Clay_BeginLayout")
03920 void Clay_BeginLayout(void) {
03921     Clay_Context* context = Clay_GetCurrentContext();
03922     Clay__InitializeEphemeralMemory(context);
03923     context->generation++;
03924     context->dynamicElementIndex = 0;
03925     // Set up the root container that covers the entire window
03926     Clay_Dimensions rootDimensions = {context->layoutDimensions.width,
      context->layoutDimensions.height};
03927     if (context->debugModeEnabled) {
03928         rootDimensions.width -= (float)Clay__debugViewWidth;
03929     }
03930     context->booleanWarnings = CLAY__INIT(Clay_BooleanWarnings) CLAY__DEFAULT_STRUCT;
03931     Clay__OpenElement();
03932     CLAY_ID("Clay__RootContainer");
03933     CLAY_LAYOUT({ .sizing = {CLAY_SIZING_FIXED((rootDimensions.width)),
      CLAY_SIZING_FIXED(rootDimensions.height)} });
03934     Clay__ElementPostConfiguration();
03935     Clay__int32_tArray_Add(&context->openLayoutElementStack, 0);
03936     Clay__LayoutElementTreeRootArray_Add(&context->layoutElementTreeRoots,
      CLAY__INIT(Clay__LayoutElementTreeRoot) { .layoutElementIndex = 0 });
03937 }
03938
03939 Clay_TextElementConfig Clay__DebugView_ErrorTextConfig = {.textColor = {255, 0, 0, 255}, .fontSize =
      16, .wrapMode = CLAY_TEXT_WRAP_NONE };
03940
03941 CLAY_WASM_EXPORT("Clay_EndLayout")
03942 Clay_RenderCommandArray Clay_EndLayout() {
```

```
03943     Clay_Context* context = Clay_GetCurrentContext();
03944     Clay__CloseElement();
03945     if (context->debugModeEnabled) {
03946         context->warningsEnabled = false;
03947         Clay__RenderDebugView();
03948         context->warningsEnabled = true;
03949     }
03950     if (context->booleanWarnings.maxElementsExceeded) {
03951         Clay_String message = CLAY_STRING("Clay Error: Layout elements exceeded
      Clay__maxElementCount");
03952         Clay__AddRenderCommand(CLAY__INIT(Clay_RenderCommand ) { .boundingBox = {
      context->layoutDimensions.width / 2 - 59 * 4, context->layoutDimensions.height / 2, 0, 0 },  .config =
      { .textElementConfig = &Clay__DebugView_ErrorTextConfig }, .text = CLAY__INIT(Clay_StringSlice) {
      .length = message.length, .chars = message.chars, .baseChars = message.chars }, .commandType =
      CLAY_RENDER_COMMAND_TYPE_TEXT });
03953     } else {
03954         Clay__CalculateFinalLayout();
03955     }
03956     return context->renderCommands;
03957 }
03958
03959 CLAY_WASM_EXPORT("Clay_GetElementId")
03960 Clay_ElementId Clay_GetElementId(Clay_String idString) {
03961     return Clay__HashString(idString, 0, 0);
03962 }
03963
03964 CLAY_WASM_EXPORT("Clay_GetElementIdWithIndex")
03965 Clay_ElementId Clay_GetElementIdWithIndex(Clay_String idString, uint32_t index) {
03966     return Clay__HashString(idString, index, 0);
03967 }
03968
03969 bool Clay_Hovered(void) {
03970     Clay_Context* context = Clay_GetCurrentContext();
03971     if (context->booleanWarnings.maxElementsExceeded) {
03972         return false;
03973     }
03974     Clay_LayoutElement *openLayoutElement = Clay__GetOpenLayoutElement();
03975     // If the element has no id attached at this point, we need to generate one
03976     if (openLayoutElement->id == 0) {
03977         Clay__GenerateIdForAnonymousElement(openLayoutElement);
03978     }
03979     for (int32_t i = 0; i < context->pointerOverIds.length; ++i) {
03980         if (Clay__ElementIdArray_Get(&context->pointerOverIds, i)->id == openLayoutElement->id) {
03981             return true;
03982         }
03983     }
03984     return false;
03985 }
03986
03987 void Clay_OnHover(void (*onHoverFunction)(Clay_ElementId elementId, Clay_PointerData pointerInfo,
      intptr_t userData), intptr_t userData) {
03988     Clay_Context* context = Clay_GetCurrentContext();
03989     if (context->booleanWarnings.maxElementsExceeded) {
03990         return;
03991     }
03992     Clay_LayoutElement *openLayoutElement = Clay__GetOpenLayoutElement();
03993     if (openLayoutElement->id == 0) {
03994         Clay__GenerateIdForAnonymousElement(openLayoutElement);
03995     }
03996     Clay_LayoutElementHashMapItem *hashMapItem = Clay__GetHashMapItem(openLayoutElement->id);
03997     hashMapItem->onHoverFunction = onHoverFunction;
03998     hashMapItem->hoverFunctionUserData = userData;
03999 }
04000
04001 CLAY_WASM_EXPORT("Clay_PointerOver")
04002 bool Clay_PointerOver(Clay_ElementId elementId) { // TODO return priority for separating multiple
      results
04003     Clay_Context* context = Clay_GetCurrentContext();
04004     for (int32_t i = 0; i < context->pointerOverIds.length; ++i) {
04005         if (Clay__ElementIdArray_Get(&context->pointerOverIds, i)->id == elementId.id) {
04006             return true;
04007         }
04008     }
04009     return false;
04010 }
04011
04012 CLAY_WASM_EXPORT("Clay_GetScrollContainerData")
04013 Clay_ScrollContainerData Clay_GetScrollContainerData(Clay_ElementId id) {
04014     Clay_Context* context = Clay_GetCurrentContext();
04015     for (int32_t i = 0; i < context->scrollContainerDatas.length; ++i) {
04016         Clay__ScrollContainerDataInternal *scrollContainerData =
      Clay__ScrollContainerDataInternalArray_Get(&context->scrollContainerDatas, i);
04017         if (scrollContainerData->elementId == id.id) {
04018             return CLAY__INIT(Clay_ScrollContainerData) {
04019                 .scrollPosition = &scrollContainerData->scrollPosition,
04020                 .scrollContainerDimensions = { scrollContainerData->boundingBox.width,
      scrollContainerData->boundingBox.height },
```

```
04021                 .contentDimensions = scrollContainerData->contentSize,
04022                 .config = *Clay__FindElementConfigWithType(scrollContainerData->layoutElement,
    CLAY__ELEMENT_CONFIG_TYPE_SCROLL_CONTAINER).scrollElementConfig,
04023                 .found = true
04024             };
04025         }
04026     }
04027     return CLAY__INIT(Clay_ScrollContainerData) CLAY__DEFAULT_STRUCT;
04028 }
04029
04030 CLAY_WASM_EXPORT("Clay_GetElementData")
04031 Clay_ElementData Clay_GetElementData(Clay_ElementId id){
04032     Clay_LayoutElementHashMapItem * item = Clay__GetHashMapItem(id.id);
04033     if(item == &CLAY__LAYOUT_ELEMENT_HASH_MAP_ITEM_DEFAULT) {
04034         return CLAY__INIT(Clay_ElementData) CLAY__DEFAULT_STRUCT;
04035     }
04036
04037     return CLAY__INIT(Clay_ElementData){
04038         .boundingBox = item->boundingBox,
04039         .found = true
04040     };
04041 }
04042
04043 CLAY_WASM_EXPORT("Clay_SetDebugModeEnabled")
04044 void Clay_SetDebugModeEnabled(bool enabled) {
04045     Clay_Context* context = Clay_GetCurrentContext();
04046     context->debugModeEnabled = enabled;
04047 }
04048
04049 CLAY_WASM_EXPORT("Clay_IsDebugModeEnabled")
04050 bool Clay_IsDebugModeEnabled(void) {
04051     Clay_Context* context = Clay_GetCurrentContext();
04052     return context->debugModeEnabled;
04053 }
04054
04055 CLAY_WASM_EXPORT("Clay_SetCullingEnabled")
04056 void Clay_SetCullingEnabled(bool enabled) {
04057     Clay_Context* context = Clay_GetCurrentContext();
04058     context->disableCulling = !enabled;
04059 }
04060
04061 CLAY_WASM_EXPORT("Clay_SetExternalScrollHandlingEnabled")
04062 void Clay_SetExternalScrollHandlingEnabled(bool enabled) {
04063     Clay_Context* context = Clay_GetCurrentContext();
04064     context->externalScrollHandlingEnabled = enabled;
04065 }
04066
04067 CLAY_WASM_EXPORT("Clay_GetMaxElementCount")
04068 int32_t Clay_GetMaxElementCount(void) {
04069     Clay_Context* context = Clay_GetCurrentContext();
04070     return context->maxElementCount;
04071 }
04072
04073 CLAY_WASM_EXPORT("Clay_SetMaxElementCount")
04074 void Clay_SetMaxElementCount(int32_t maxElementCount) {
04075     Clay_Context* context = Clay_GetCurrentContext();
04076     if (context) {
04077         context->maxElementCount = maxElementCount;
04078     } else {
04079         Clay__defaultMaxElementCount = maxElementCount; // TODO: Fix this
04080     Clay__defaultMaxMeasureTextWordCacheCount = maxElementCount * 2;
04081     }
04082 }
04083
04084 CLAY_WASM_EXPORT("Clay_GetMaxMeasureTextCacheWordCount")
04085 int32_t Clay_GetMaxMeasureTextCacheWordCount(void) {
04086     Clay_Context* context = Clay_GetCurrentContext();
04087     return context->maxMeasureTextCacheWordCount;
04088 }
04089
04090 CLAY_WASM_EXPORT("Clay_SetMaxMeasureTextCacheWordCount")
04091 void Clay_SetMaxMeasureTextCacheWordCount(int32_t maxMeasureTextCacheWordCount) {
04092     Clay_Context* context = Clay_GetCurrentContext();
04093     if (context) {
04094         Clay__currentContext->maxMeasureTextCacheWordCount = maxMeasureTextCacheWordCount;
04095     } else {
04096         Clay__defaultMaxMeasureTextWordCacheCount = maxMeasureTextCacheWordCount; // TODO: Fix this
04097     }
04098 }
04099
04100 CLAY_WASM_EXPORT("Clay_ResetMeasureTextCache")
04101 void Clay_ResetMeasureTextCache(void) {
04102     Clay_Context* context = Clay_GetCurrentContext();
04103     context->measureTextHashMapInternal.length = 0;
04104     context->measureTextHashMapInternalFreeList.length = 0;
04105     context->measureTextHashMap.length = 0;
04106     context->measuredWords.length = 0;
```

```
04107     context->measuredWordsFreeList.length = 0;
04108
04109     for (int32_t i = 0; i < context->measureTextHashMap.capacity; ++i) {
04110         context->measureTextHashMap.internalArray[i] = 0;
04111     }
04112     context->measureTextHashMapInternal.length = 1; // Reserve the 0 value to mean "no next element"
04113 }
04114
04115 #endif // CLAY_IMPLEMENTATION
04116
04117 /*
04118 LICENSE
04119 zlib/libpng license
04120
04121 Copyright (c) 2024 Nic Barker
04122
04123 This software is provided 'as-is', without any express or implied warranty.
04124 In no event will the authors be held liable for any damages arising from the
04125 use of this software.
04126
04127 Permission is granted to anyone to use this software for any purpose,
04128 including commercial applications, and to alter it and redistribute it
04129 freely, subject to the following restrictions:
04130
04131     1. The origin of this software must not be misrepresented; you must not
04132     claim that you wrote the original software. If you use this software in a
04133     product, an acknowledgment in the product documentation would be
04134     appreciated but is not required.
04135
04136     2. Altered source versions must be plainly marked as such, and must not
04137     be misrepresented as being the original software.
04138
04139     3. This notice may not be removed or altered from any source
04140     distribution.
04141 */
```

8.16 moduleLib.h

```
00001 #pragma once
00002
00003 #ifdef _WIN32
00004 #include <windows.h>
00005 #else
00006 #include <dlfcn.h>
00007 #include <unistd.h>
00008 #include <sys/types.h>
00009 #include <sys/stat.h>
00010 #include <fcntl.h>
00011 #include <errno.h>
00012
00013 #endif
00014 #include <cstdio>
00015 #include <fstream>
00016 #include <string>
00017
00018 #ifndef SPECTRAL_PLATFORM
00019 #define SPECTRAL_PLATFORM "unknown"
00020 #endif
00021
00022 #ifdef _WIN32
00023 #define CEXPORT extern "C" __declspec(dllexport)
00024 #define EXPORT __declspec(dllexport)
00025 #else
00026 #define CEXPORT extern "C"
00027 #define EXPORT
00028 #endif
00029
00030 #include <filesystem>
00031
00032 #include <cstring>
00033
00034 std::filesystem::path getexepath() {
00035 #ifdef _WIN32
00036     wchar_t path[MAX_PATH] = { 0 };
00037     GetModuleFileNameW(NULL, path, MAX_PATH);
00038     return path;
00039 #else
00040 #define PATH_MAX 4096
00041     char result[PATH_MAX];
00042     ssize_t count = readlink("/proc/self/exe", result, PATH_MAX);
00043     return std::string(result, (count > 0) ? count : 0);
00044 #endif
00045 }
```

```
00046
00047 std::filesystem::path getexedir() {
00048     return getexepath().parent_path();
00049 }
00050
00051 bool readFile(const char* path, std::string& out) {
00052     std::ifstream file(path);
00053     if (!file.is_open()) {
00054         return false;
00055     }
00056     out = std::string((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
00057     return true;
00058 }
00059
00060 #ifdef _WIN32
00061 const char* outputSuffix = ".dll";
00062 #else
00063 const char* outputSuffix = ".so";
00064 #endif
00065
00066 const char* spectralSuffix = ".splmod";
00067
00068 struct DynamicLibrary {
00069 #ifdef _WIN32
00070     HINSTANCE handle;
00071 #else
00072     void* handle;
00073 #endif
00074 public:
00075     const char* mod_name;
00076     const char* mod_imp;
00077
00078 private:
00079     void load(const char* path_in) {
00080 #ifdef _WIN32
00081         char* path = (char*)malloc(strlen(path_in) + 2);
00082         strcpy(path, path_in);
00083         strcat(path, ".");
00084         char* setdlldir = (char*)malloc(strlen(SPECTRAL_PLATFORM) + 7);
00085         strcpy(setdlldir, "./lib/");
00086         strcat(setdlldir, SPECTRAL_PLATFORM);
00087         if (!SetDllDirectoryA(setdlldir)) {
00088             printf("Error setting DLL directory\n");
00089             printf("Error code: %d\n", GetLastError());
00090         }
00091 #else
00092         char* path = (char*)malloc(strlen(path_in) + 1);
00093         strcpy(path, path_in);
00094 #endif
00095 #ifdef _WIN32
00096         handle = LoadLibraryA(path);
00097 #else
00098         handle = dlopen(path, RTLD_LAZY);
00099 #endif
00100         if (!handle) {
00101             printf("Error loading library %s\n", path);
00102             handle = NULL;
00103 #ifdef _WIN32
00104             printf("Error code: %d\n", GetLastError());
00105             printf("libdir: %s\n", setdlldir);
00106 #else
00107             printf("Error: %s\n", dlerror());
00108 #endif
00109         }
00110         free(path);
00111         free(setdlldir);
00112     }
00113 public:
00114
00115     DynamicLibrary() {
00116         mod_name = nullptr;
00117         mod_imp = nullptr;
00118         handle = NULL;
00119     }
00120
00121     DynamicLibrary(const char* path, const char* ident) {
00122         mod_name = ident;
00123         mod_imp = path;
00124         char* p = makePath(path, ident);
00125         load(p);
00126         free(p);
00127     }
00128
00129     static char* makePath(const char* path, const char* ident) {
00130
00131         char* fullPath =
      (char*)malloc(10+strlen(ident)*2+strlen(path)+1+strlen(SPECTRAL_PLATFORM)+strlen(spectralSuffix)+10);
```

```
00132         sprintf(fullPath, "modules/%s/%s/%s_%s%s", SPECTRAL_PLATFORM, ident, ident, path,
      spectralSuffix);
00133         return fullPath;
00134     }
00135
00136     ~DynamicLibrary() {
00137 #ifdef _WIN32
00138         FreeLibrary(handle);
00139 #else
00140         dlclose(handle);
00141 #endif
00142     }
00143
00144     void* getSymbol(const char* name) {
00145         void* sym;
00146         if (!handle) {
00147             printf("Error: Library not loaded\n");
00148             return NULL;
00149         }
00150 #ifdef _WIN32
00151         sym = (void*)GetProcAddress(handle, name);
00152 #else
00153         sym = dlsym(handle, name);
00154 #endif
00155         if (!sym) {
00156             printf("Error loading symbol %s\n", name);
00157 #ifdef _WIN32
00158             printf("Error code: %ld\n", GetLastError());
00159 #else
00160             printf("Error: %s\n", dlerror());
00161 #endif
00162         }
00163         return sym;
00164     }
00165
00166     bool valid() {
00167         return handle != NULL;
00168     }
00169 };
00170
00171 struct Module {
00172     DynamicLibrary lib;
00173     explicit Module(const char* path, const char* ident) : lib(path, ident) {
00174         if (!lib.valid()) {
00175             printf("Error loading module %s\n", path);
00176             return;
00177         }
00178         std::filesystem::path p = getexedir();
00179         char* pth = DynamicLibrary::makePath(path, ident);
00180         if (!std::filesystem::exists(p / pth)) {
00181             printf("Error: Module %s.%s not found\n", ident, path);
00182             return;
00183         } else {
00184             printf("Module %s.%s loaded\n", ident, path);
00185         }
00186     }
00187 };
```

Index