

Canim

Generated on Thu Nov 27 2025 16:29:56 for Canim by Doxygen 1.15.0

Thu Nov 27 2025 16:29:56

1 Todo List	1
2 Directory Hierarchy	3
2.1 Directories	3
3 Data Structure Index	5
3.1 Data Structures	5
4 File Index	7
4.1 File List	7
5 Directory Documentation	9
5.1 include/canim Directory Reference	9
5.2 src/core Directory Reference	10
5.3 src/gfx Directory Reference	10
5.4 include Directory Reference	11
5.5 src/loader Directory Reference	11
5.6 src Directory Reference	11
6 Data Structure Documentation	13
6.1 GfxAPI Struct Reference	13
6.1.1 Detailed Description	13
6.1.2 Field Documentation	14
6.1.2.1 gfx_create_device	14
6.1.2.2 gfx_destroy_device	14
6.2 GfxContainer Struct Reference	14
6.2.1 Detailed Description	15
6.2.2 Field Documentation	15
6.2.2.1 api	15
6.2.2.2 backend	15
6.2.2.3 handle	15
6.2.2.4 impl	15
6.3 GfxDevice Struct Reference	16
6.3.1 Detailed Description	16
6.3.2 Field Documentation	16
6.3.2.1 egl_context	16
6.3.2.2 egl_display	16
6.3.2.3 egl_surface	16
6.3.2.4 glctx	16
6.3.2.5 headless	17
6.3.2.6 height	17
6.3.2.7 inst	17
6.3.2.8 width	17
6.3.2.9 win	17

6.4 GfxInitInfo Struct Reference	17
6.4.1 Detailed Description	18
6.4.2 Field Documentation	18
6.4.2.1 headless	18
6.4.2.2 height	18
6.4.2.3 native_window	18
6.4.2.4 width	19
6.5 SVec Struct Reference	19
6.5.1 Detailed Description	19
6.5.2 Field Documentation	19
6.5.2.1 data	19
6.5.2.2 element_size	19
6.5.2.3 list_size	19
7 File Documentation	21
7.1 include/canim/canim.h File Reference	21
7.2 canim.h	22
7.3 include/canim/core.h File Reference	22
7.3.1 Detailed Description	25
7.3.2 Macro Definition Documentation	25
7.3.2.1 BIT_ALIGN	25
7.3.2.2 BIT_IGNORE	25
7.3.2.3 BIT_SIZE	26
7.3.2.4 BUFFER_SIZE	26
7.3.2.5 CANIM_API	26
7.3.2.6 CANIM_PLATFORM_UNKNOWN	26
7.3.2.7 EPSILON	26
7.3.2.8 FATAL	26
7.3.2.9 IS_AN_ERROR	27
7.3.2.10 max	27
7.3.2.11 NONFATAL	27
7.3.2.12 NOOP	27
7.3.2.13 null	27
7.3.2.14 PCLOSE	28
7.3.2.15 POPEN	28
7.3.2.16 REALIGN	28
7.3.2.17 STATUS_TYPE	28
7.3.2.18 STATUS_TYPE_MASK	28
7.3.2.19 STATUS_TYPE_SHIFT	29
7.3.2.20 SUCCESS	29
7.3.2.21 WB	29
7.3.3 Typedef Documentation	29

7.3.3.1 CanimResult	29
7.3.4 Enumeration Type Documentation	29
7.3.4.1 anonymous enum	29
7.3.5 Function Documentation	30
7.3.5.1 print_error()	30
7.3.5.2 read_be_f32()	31
7.3.5.3 read_be_f64()	32
7.3.5.4 read_be_u16()	32
7.3.5.5 read_be_u32()	32
7.3.5.6 read_be_u64()	33
7.3.5.7 read_le_f32()	33
7.3.5.8 read_le_f64()	33
7.3.5.9 read_le_u16()	34
7.3.5.10 read_le_u32()	34
7.3.5.11 read_le_u64()	34
7.3.5.12 svec_free()	35
7.3.5.13 svec_get()	35
7.3.5.14 svec_init()	35
7.3.5.15 svec_pop()	36
7.3.5.16 svec_push()	36
7.3.5.17 svec_set()	36
7.3.5.18 write_be_f32()	37
7.3.5.19 write_be_f64()	37
7.3.5.20 write_be_u16()	37
7.3.5.21 write_be_u32()	38
7.3.5.22 write_be_u64()	38
7.3.5.23 write_le_f32()	38
7.3.5.24 write_le_f64()	38
7.3.5.25 write_le_u16()	39
7.3.5.26 write_le_u32()	39
7.3.5.27 write_le_u64()	39
7.4 core.h	40
7.5 include/canim/gfx.h File Reference	43
7.5.1 Typedef Documentation	45
7.5.1.1 GfxAPI	45
7.5.1.2 GfxContainer	45
7.5.1.3 GfxDevice	45
7.5.2 Enumeration Type Documentation	45
7.5.2.1 GfxBackend	45
7.6 gfx.h	46
7.7 include/canim/loader.h File Reference	46
7.7.1 Function Documentation	48

7.7.1.1 gfx_load_backend()	48
7.7.1.2 gfx_unload_backend()	49
7.8 loader.h	50
7.9 src/core/core.c File Reference	50
7.9.1 Function Documentation	51
7.9.1.1 print_error()	51
7.10 core.c	52
7.11 src/core/endian.c File Reference	53
7.11.1 Function Documentation	54
7.11.1.1 read_be_f32()	54
7.11.1.2 read_be_f64()	54
7.11.1.3 read_be_u16()	55
7.11.1.4 read_be_u32()	55
7.11.1.5 read_be_u64()	55
7.11.1.6 read_le_f32()	56
7.11.1.7 read_le_f64()	56
7.11.1.8 read_le_u16()	56
7.11.1.9 read_le_u32()	57
7.11.1.10 read_le_u64()	57
7.11.1.11 write_be_f32()	58
7.11.1.12 write_be_f64()	58
7.11.1.13 write_be_u16()	58
7.11.1.14 write_be_u32()	59
7.11.1.15 write_be_u64()	59
7.11.1.16 write_le_f32()	60
7.11.1.17 write_le_f64()	60
7.11.1.18 write_le_u16()	60
7.11.1.19 write_le_u32()	61
7.11.1.20 write_le_u64()	61
7.12 endian.c	62
7.13 src/core/svec.c File Reference	63
7.13.1 Function Documentation	64
7.13.1.1 svec_free()	64
7.13.1.2 svec_get()	64
7.13.1.3 svec_init()	65
7.13.1.4 svec_pop()	65
7.13.1.5 svec_push()	65
7.13.1.6 svec_set()	66
7.14 svec.c	66
7.15 src/gfx/gl.c File Reference	67
7.15.1 Function Documentation	68
7.15.1.1 gl_create_device()	68

7.15.1.2 gl_destroy_device()	69
7.15.2 Variable Documentation	69
7.15.2.1 GFX_GL_API	69
7.16 gl.c	70
7.17 src/gfx/vk.c File Reference	72
7.17.1 Function Documentation	72
7.17.1.1 vk_create_device()	72
7.18 vk.c	73
7.19 src/loader/loader.c File Reference	75
7.19.1 Function Documentation	75
7.19.1.1 gfx_backend_libname()	75
7.19.1.2 gfx_load_backend()	76
7.19.1.3 gfx_unload_backend()	77
7.20 loader.c	77
Index	79

Chapter 1

Todo List

Global `vk_create_device (CanimResult *result, GfxContainer *container, const GfxInitInfo *info)`

FIX

FIX

FIX

Chapter 2

Directory Hierarchy

2.1 Directories

canim	9
canim.h	21
core.h	22
gfx.h	43
loader.h	46
core	10
core.c	50
endian.c	53
svec.c	63
gfx	10
gl.c	67
vk.c	72
include	11
canim	9
canim.h	21
core.h	22
gfx.h	43
loader.h	46
loader	11
loader.c	75
src	11
core	10
core.c	50
endian.c	53
svec.c	63
gfx	10
gl.c	67
vk.c	72
loader	11
loader.c	75

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

GfxAPI	
	This struct stores the cross platform API functions This struct has a fptr, but i am not gonna document it, because i believe in freedom and you can't stop me
GfxContainer	13
GfxDevice	14
GfxInitInfo	16
	This struct stores info neccesary to initialize graphics
SVec	17
	Minimal growable byte-vector
	19

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

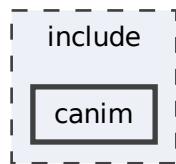
include/canim/ canim.h	21
include/canim/ core.h	This is everything core to Canim that is shared between subsystems	22
include/canim/ gfx.h	43
include/canim/ loader.h	46
src/core/ core.c	50
src/core/ endian.c	53
src/core/ svec.c	63
src/gfx/ gl.c	67
src/gfx/ vk.c	72
src/loader/ loader.c	75

Chapter 5

Directory Documentation

5.1 include/canim Directory Reference

Directory dependency graph for canim:



Files

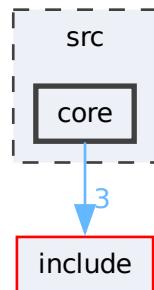
- file [canim.h](#)
- file [core.h](#)

This is everything core to Canim that is shared between subsystems.

- file [gfx.h](#)
- file [loader.h](#)

5.2 src/core Directory Reference

Directory dependency graph for core:

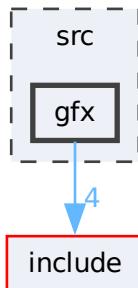


Files

- file [core.c](#)
- file [endian.c](#)
- file [svec.c](#)

5.3 src/gfx Directory Reference

Directory dependency graph for gfx:



Files

- file [gl.c](#)
- file [vk.c](#)

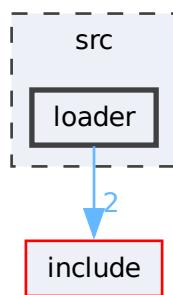
5.4 include Directory Reference

Directories

- directory [canim](#)

5.5 src/loader Directory Reference

Directory dependency graph for loader:

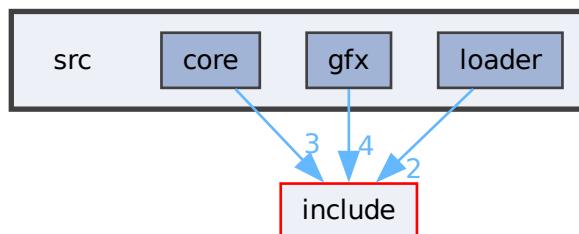


Files

- file [loader.c](#)

5.6 src Directory Reference

Directory dependency graph for src:



Directories

- directory [core](#)
- directory [gfx](#)
- directory [loader](#)

Chapter 6

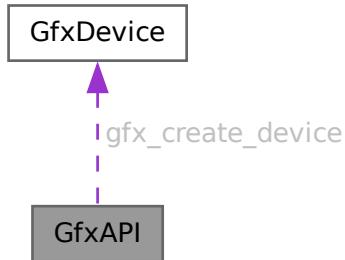
Data Structure Documentation

6.1 GfxAPI Struct Reference

This struct stores the cross platform API functions This struct has a fptr, but i am not gonna document it, because i believe in freedom and you can't stop me.

```
#include <gfx.h>
```

Collaboration diagram for GfxAPI:



Data Fields

- `GfxDevice GfxDevice *(* gfx_create_device)(CanimResult *, GfxContainer *, const GfxInitInfo *)`
- `void(* gfx_destroy_device)(CanimResult *, GfxContainer *)`

6.1.1 Detailed Description

This struct stores the cross platform API functions This struct has a fptr, but i am not gonna document it, because i believe in freedom and you can't stop me.

Definition at line 35 of file [gfx.h](#).

6.1.2 Field Documentation

6.1.2.1 gfx_create_device

```
GfxDevice*(* GfxAPI::gfx_create_device) (CanimResult *, GfxContainer *, const GfxInitInfo *)
```

Definition at line 36 of file [gfx.h](#).

6.1.2.2 gfx_destroy_device

```
void(* GfxAPI::gfx_destroy_device) (CanimResult *, GfxContainer *)
```

Definition at line 38 of file [gfx.h](#).

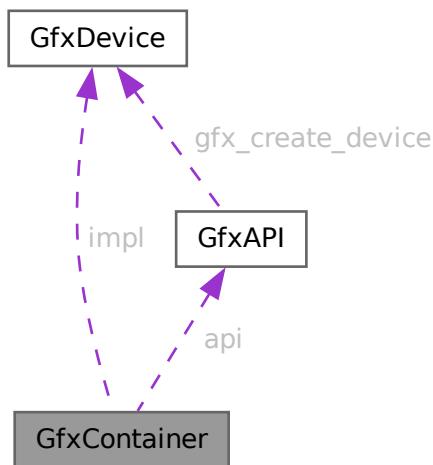
The documentation for this struct was generated from the following file:

- [include/canim/gfx.h](#)

6.2 GfxContainer Struct Reference

```
#include <gfx.h>
```

Collaboration diagram for GfxContainer:



Data Fields

- `GfxAPI api`
The API.
- `GfxDeviceGfxDevice * impl`
The Implementation specific gfxdevice.
- `GfxBackend backend`
The backend.
- `void * handle`
Handle to the dynamic lib.

6.2.1 Detailed Description

Definition at line 42 of file [gfx.h](#).

6.2.2 Field Documentation

6.2.2.1 `api`

`GfxAPI GfxContainer::api`

The API.

Definition at line 43 of file [gfx.h](#).

6.2.2.2 `backend`

`GfxBackend GfxContainer::backend`

The backend.

Definition at line 45 of file [gfx.h](#).

6.2.2.3 `handle`

`void* GfxContainer::handle`

Handle to the dynamic lib.

Definition at line 46 of file [gfx.h](#).

6.2.2.4 `impl`

`GfxDeviceGfxDevice* GfxContainer::impl`

The Implementation specific gfxdevice.

Definition at line 44 of file [gfx.h](#).

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

- `include/canim/gfx.h`

6.3 GfxDevice Struct Reference

Data Fields

- bool `headless`
- int `width`
- int `height`
- `SDL_GLContext` `glctx`
- `SDL_Window *` `win`
- `EGLDisplay` `egl_display`
- `EGLSurface` `egl_surface`
- `EGLContext` `egl_context`
- `VkInstance` `inst`

6.3.1 Detailed Description

Definition at line 11 of file [gl.c](#).

6.3.2 Field Documentation

6.3.2.1 egl_context

`EGLContext GfxDevice::egl_context`

Definition at line 19 of file [gl.c](#).

6.3.2.2 egl_display

`EGLDisplay GfxDevice::egl_display`

Definition at line 17 of file [gl.c](#).

6.3.2.3 egl_surface

`EGLSurface GfxDevice::egl_surface`

Definition at line 18 of file [gl.c](#).

6.3.2.4 glctx

`SDL_GLContext GfxDevice::glctx`

Definition at line 15 of file [gl.c](#).

6.3.2.5 headless

```
bool GfxDevice::headless
```

Definition at line 12 of file [gl.c](#).

6.3.2.6 height

```
int GfxDevice::height
```

Definition at line 14 of file [gl.c](#).

6.3.2.7 inst

```
VkInstance GfxDevice::inst
```

Definition at line 11 of file [vk.c](#).

6.3.2.8 width

```
int GfxDevice::width
```

Definition at line 13 of file [gl.c](#).

6.3.2.9 win

```
SDL_Window * GfxDevice::win
```

Definition at line 16 of file [gl.c](#).

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

- [src/gfx/gl.c](#)
- [src/gfx/vk.c](#)

6.4 GfxInitInfo Struct Reference

This struct stores info neccesary to initialize graphics.

```
#include <gfx.h>
```

Data Fields

- bool `headless`
This stores whether or not the window is headless.
- void * `native_window`
This stores a pointer to a native window.
- int `width`
The width of the window.
- int `height`
The height of it.

6.4.1 Detailed Description

This struct stores info neccesary to initialize graphics.

Definition at line [24](#) of file `gfx.h`.

6.4.2 Field Documentation

6.4.2.1 `headless`

```
bool GfxInitInfo::headless
```

This stores whether or not the window is headless.

Definition at line [25](#) of file `gfx.h`.

6.4.2.2 `height`

```
int GfxInitInfo::height
```

The height of it.

Definition at line [28](#) of file `gfx.h`.

6.4.2.3 `native_window`

```
void* GfxInitInfo::native_window
```

This stores a pointer to a native window.

Definition at line [26](#) of file `gfx.h`.

6.4.2.4 width

```
int GfxInitInfo::width
```

The width of the window.

Definition at line 27 of file [gfx.h](#).

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

- [include/canim/gfx.h](#)

6.5 SVec Struct Reference

Minimal growable byte-vector.

```
#include <core.h>
```

Data Fields

- `size_t element_size`
Size of each element in bytes.
- `size_t list_size`
Number of elements stored.
- `unsigned char * data`
Contiguous raw byte storage.

6.5.1 Detailed Description

Minimal growable byte-vector.

Definition at line 278 of file [core.h](#).

6.5.2 Field Documentation

6.5.2.1 data

```
unsigned char* SVec::data
```

Contiguous raw byte storage.

Definition at line 281 of file [core.h](#).

6.5.2.2 element_size

```
size_t SVec::element_size
```

Size of each element in bytes.

Definition at line 279 of file [core.h](#).

6.5.2.3 list_size

```
size_t SVec::list_size
```

Number of elements stored.

Definition at line 280 of file [core.h](#).

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

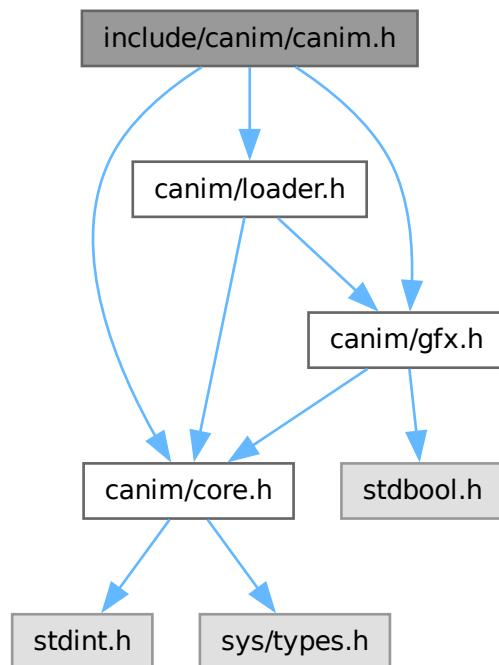
- [include/canim/core.h](#)

Chapter 7

File Documentation

7.1 include/canim/canim.h File Reference

```
#include "canim/core.h"
#include "canim/gfx.h"
#include "canim/loader.h"
Include dependency graph for canim.h:
```



7.2 canim.h

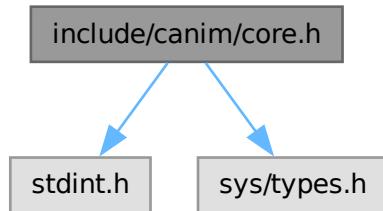
[Go to the documentation of this file.](#)

```
00001 // SPDX-License-Identifier: MIT
00002 #pragma once
00003 #include "canim/core.h"
00004 #include "canim/gfx.h"
00005 #include "canim/loader.h"
```

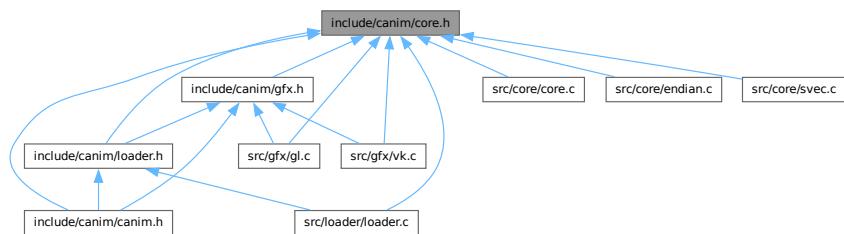
7.3 include/canim/core.h File Reference

This is everything core to Canim that is shared between subsystems.

```
#include <stdint.h>
#include <sys/types.h>
Include dependency graph for core.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [SVec](#)

Minimal growable byte-vector.

Macros

- `#define CANIM_API __attribute__((visibility("default")))`
Meant for a visibility label to all parts of the Canim API.
- `#define STATUS_TYPE_SHIFT 24`
The bit shift applied to the value of an error to get it's type.
- `#define STATUS_TYPE_MASK 0xFF000000`
The mask applied to the an error to eliminate it's subtype.
- `#define SUCCESS 0x00`
This is the status code for success.
- `#define NOOP 0x01`
No operation performed.
- `#define NONFATAL 0x02`
Non-fatal error (unused).
- `#define FATAL 0x03`
Fatal error.
- `#define STATUS_TYPE(code)`
Extracts the status type (SUCCESS/NOOP/NONFATAL/FATAL).
- `#define IS_AN_ERROR(x)`
True if code is an error (NONFATAL or FATAL).
- `#define CANIM_PLATFORM_UNKNOWN 1`
Defined if no known platform could be detected.
- `#define max(a, b)`
A max macro.
- `#define EPSILON 0.000001`
Tolerance for floating-point comparisons.
- `#define null NULL`
I do not want to capitalize NULL.
- `#define BUFFER_SIZE 4096`
The size of a buffer.
- `#define POPEN popen`
A cross platform alias for opening a pipe.
- `#define PCLOSE pclose`
A cross platform alias for closing a pipe.
- `#define WB "w"`
Correct mode for writing in binary files on each platform.
- `#define BIT_IGNORE 4`
Alignment granularity in bits (round up to $2^{\text{BIT_IGNORE}}$).
- `#define BIT_SIZE ((1u << BIT_IGNORE) - 1u)`
 $2^{\text{BIT_IGNORE}} - 1$
- `#define BIT_ALIGN(x)`
Round x up to nearest aligned multiple of $2^{\text{BIT_IGNORE}}$. Always.
- `#define REALIGN(a, b)`
True if a and b land in different aligned capacity buckets.

Typedefs

- `typedef uint32_t CanimResult`
Result code used across Canim operations.

Enumerations

```
• enum {
    GFX_DEVICE_CALLOC_ERROR , EGL_NO_DISPLAY_ERROR , EGL_DISPLAY_INIT_ERROR ,
    EGL_DISPLAY_CONFIGURATION_ERROR ,
    EGL_NO_SURFACE_ERROR , EGL_NO_CONTEXT_ERROR , EGL_MAKE_CURRENT_ERROR ,
    SDL_INIT_VIDEO_ERROR ,
    SDL_WINDOW_CREATION_ERROR , SDL_GL_CONTEXT_CREATION_ERROR , SDL_GLAD_LOAD_ERROR
    , EGL_GLAD_LOAD_ERROR ,
    SVEC_ZERO_ELEMENT_SIZE , SVEC_REALLOC_FAIL }
```

This are all of the error codes.

Functions

- void `print_error (CanimResult error)`

Print out the error.
- static uint16_t `read_be_u16 (const unsigned char *p)`

Read a 16-bit unsigned integer in big endian byte order.
- static uint32_t `read_be_u32 (const unsigned char *p)`

Read a 32-bit unsigned integer in big endian byte order.
- static uint64_t `read_be_u64 (const unsigned char *p)`

Read a 64-bit unsigned integer in big endian byte order.
- static uint16_t `read_le_u16 (const unsigned char *p)`

Read a 16-bit unsigned integer in little endian byte order.
- static uint32_t `read_le_u32 (const unsigned char *p)`

Read a 32-bit unsigned integer in little endian byte order.
- static uint64_t `read_le_u64 (const unsigned char *p)`

Read a 64-bit unsigned integer in little endian byte order.
- static void `write_be_u16 (unsigned char *p, uint16_t v)`

Write a 16-bit unsigned integer in big endian byte order.
- static void `write_be_u32 (unsigned char *p, uint32_t v)`

Write a 32-bit unsigned integer in big endian byte order.
- static void `write_be_u64 (unsigned char *p, uint64_t v)`

Write a 64-bit unsigned integer in big endian byte order.
- static void `write_le_u16 (unsigned char *p, uint16_t v)`

Write a 16-bit unsigned integer in little endian byte order.
- static void `write_le_u32 (unsigned char *p, uint32_t v)`

Write a 32-bit unsigned integer in little endian byte order.
- static void `write_le_u64 (unsigned char *p, uint64_t v)`

Write a 64-bit unsigned integer in little endian byte order.
- static float `read_be_f32 (const unsigned char *p)`

Read a 32-bit float in big endian byte order.
- static double `read_be_f64 (const unsigned char *p)`

Read a 64-bit float in big endian byte order.
- static float `read_le_f32 (const unsigned char *p)`

Read a 32-bit float in little endian byte order.
- static double `read_le_f64 (const unsigned char *p)`

Read a 64-bit float in little endian byte order.
- static void `write_be_f32 (unsigned char *p, float f)`

Write a 32 bit float to a big endian buffer.
- static void `write_be_f64 (unsigned char *p, double f)`

- static void [write_le_f32](#) (unsigned char *p, float f)

Write a 32 bit float to a big endian buffer.
- static void [write_le_f64](#) (unsigned char *p, double f)

Write a 64 bit double to a little endian buffer.
- static void [svec_init](#) (CanimResult *result, SVec *v, size_t elem_size)

Initialize a vector with a fixed element size.
- static void [svec_free](#) (CanimResult *result, SVec *v)

Free all memory owned by the vector.
- static void [svec_push](#) (CanimResult *result, SVec *v, const void *elem)

Append one element, reallocating only when bucket changes.
- static void [svec_pop](#) (CanimResult *result, SVec *v, void *out)

Remove the last element, shrinking bucket only if needed.
- static void * [svec_get](#) (CanimResult *result, SVec *v, size_t i)

Get a pointer to element at index i.
- static void [svec_set](#) (CanimResult *result, SVec *v, size_t i, const void *elem)

Set the element at index i to the value pointed to by elem.

7.3.1 Detailed Description

This is everything core to Canim that is shared between subsystems.

Definition in file [core.h](#).

7.3.2 Macro Definition Documentation

7.3.2.1 BIT_ALIGN

```
#define BIT_ALIGN(
    x)
```

Value:

```
(((((x) + BIT_SIZE) & ~BIT_SIZE)) ? (((x) + BIT_SIZE) & ~BIT_SIZE) : 1)
```

Round x up to nearest aligned multiple of $2^{\text{BIT_IGNORE}}$. Always.

Definition at line 269 of file [core.h](#).

```
00269 #define BIT_ALIGN(x)
00270   (((((x) + BIT_SIZE) & ~BIT_SIZE)) ? (((x) + BIT_SIZE) & ~BIT_SIZE) : 1) \
```

7.3.2.2 BIT_IGNORE

```
#define BIT_IGNORE 4
```

Alignment granularity in bits (round up to $2^{\text{BIT_IGNORE}}$).

Definition at line 261 of file [core.h](#).

7.3.2.3 BIT_SIZE

```
#define BIT_SIZE ((1u << BIT_IGNORE) - 1u)  
2^BIT_IGNORE - 1
```

Definition at line [265](#) of file [core.h](#).

7.3.2.4 BUFFER_SIZE

```
#define BUFFER_SIZE 4096
```

The size of a buffer.

Definition at line [133](#) of file [core.h](#).

7.3.2.5 CANIM_API

```
#define CANIM_API __attribute__((visibility("default")))
```

Meant for a visibility label to all parts of the Canim API.

Definition at line [11](#) of file [core.h](#).

7.3.2.6 CANIM_PLATFORM_UNKNOWN

```
#define CANIM_PLATFORM_UNKNOWN 1
```

Defined if no known platform could be detected.

Definition at line [114](#) of file [core.h](#).

7.3.2.7 EPSILON

```
#define EPSILON 0.000001
```

Tolerance for floating-point comparisons.

Definition at line [125](#) of file [core.h](#).

7.3.2.8 FATAL

```
#define FATAL 0x03
```

Fatal error.

Definition at line [35](#) of file [core.h](#).

7.3.2.9 IS_AN_ERROR

```
#define IS_AN_ERROR(  
    x)
```

Value:

```
((STATUS_TYPE((x)) == FATAL) || (STATUS_TYPE((x)) == NONFATAL))
```

True if code is an error (NONFATAL or FATAL).

Definition at line 43 of file [core.h](#).

```
00043 #define IS_AN_ERROR(x)  
00044     ((STATUS_TYPE((x)) == FATAL) || (STATUS_TYPE((x)) == NONFATAL))\n
```

\

7.3.2.10 max

```
#define max(  
    a,  
    b)
```

Value:

```
((a) > (b)) ? (a) : (b)
```

A max macro.

Definition at line 120 of file [core.h](#).

7.3.2.11 NONFATAL

```
#define NONFATAL 0x02
```

Non-fatal error (unused).

Definition at line 31 of file [core.h](#).

7.3.2.12 NOOP

```
#define NOOP 0x01
```

No operation performed.

Definition at line 27 of file [core.h](#).

7.3.2.13 null

```
#define null NULL
```

I do not want to capitalize NULL.

Definition at line 129 of file [core.h](#).

7.3.2.14 PCLOSE

```
#define PCLOSE pclose
```

A cross platform alias for closing a pipe.

Definition at line [148](#) of file [core.h](#).

7.3.2.15 POPEN

```
#define POPEN popen
```

A cross platform alias for opening a pipe.

Definition at line [140](#) of file [core.h](#).

7.3.2.16 REALIGN

```
#define REALIGN(  
    a,  
    b)
```

Value:

```
(BIT_ALIGN(a) != BIT_ALIGN(b))
```

True if a and b land in different aligned capacity buckets.

Definition at line [274](#) of file [core.h](#).

7.3.2.17 STATUS_TYPE

```
#define STATUS_TYPE(  
    code)
```

Value:

```
((code) & STATUS_TYPE_MASK) >> STATUS_TYPE_SHIFT)
```

Extracts the status type (SUCCESS/NOOP/NONFATAL/FATAL).

Definition at line [39](#) of file [core.h](#).

7.3.2.18 STATUS_TYPE_MASK

```
#define STATUS_TYPE_MASK 0xFF000000
```

The mask applied to the an error to eliminate it's subtype.

Definition at line [19](#) of file [core.h](#).

7.3.2.19 STATUS_TYPE_SHIFT

```
#define STATUS_TYPE_SHIFT 24
```

The bit shift applied to the value of an error to get it's type.

Definition at line 15 of file [core.h](#).

7.3.2.20 SUCCESS

```
#define SUCCESS 0x00
```

This is the status code for success.

Definition at line 23 of file [core.h](#).

7.3.2.21 WB

```
#define WB "w"
```

Correct mode for writing in binary files on each platform.

Definition at line 156 of file [core.h](#).

7.3.3 Typedef Documentation

7.3.3.1 CanimResult

```
typedef uint32_t CanimResult
```

Result code used across Canim operations.

Definition at line 77 of file [core.h](#).

7.3.4 Enumeration Type Documentation

7.3.4.1 anonymous enum

```
anonymous enum
```

This are all of the error codes.

Enumerator

GFX_DEVICE_CALLOC_ERROR	When using calloc to allocate memory for a GfxDevice , calloc failed
-------------------------	--

EGL_NO_DISPLAY_ERROR	When creating an EGL display, no display was created
EGL_DISPLAY_INIT_ERROR	When initializing an EGL display something failed.
EGL_DISPLAY_CONFIGURATION_ERROR	When configuring an EGL display something failed.
EGL_NO_SURFACE_ERROR	When creating an EGL surface, no surface was created
EGL_NO_CONTEXT_ERROR	When creating an EGL context, no context was created.
EGL_MAKE_CURRENT_ERROR	When the making the EGL context, surface and display the current one, something failed.
SDL_INIT_VIDEO_ERROR	When initializing SDL video, something failed.
SDL_WINDOW_CREATION_ERROR	When making a window with SDL2, something failed
SDL_GL_CONTEXT_CREATION_ERROR	When making an OpenGL context with SDL, something failed.
SDL_GLAD_LOAD_ERROR	When loading OpenGL functions with GLAD, using SDL, something failed
EGL_GLAD_LOAD_ERROR	When loading OpenGL functions with GLAD, using EGL, something failed
SVEC_ZERO_ELEMENT_SIZE	When initializing an SVec, the element size was set to zero
SVEC_REALLOC_FAIL	When reallocating space for an SVec, something failed.

Definition at line 47 of file [core.h](#).

```
00047     {
00048     GFX_DEVICE_CALLOC_ERROR =
00049         (uint32_t)0x03000000, //||< When using calloc to allocate memory for a
00050                     //||< GfxDevice, calloc failed
00051     EGL_NO_DISPLAY_ERROR, //||< When creating an EGL display, no display was
00052                     //||< created
00053     EGL_DISPLAY_INIT_ERROR, //||< When initializing an EGL display something failed
00054     EGL_DISPLAY_CONFIGURATION_ERROR, //||< When configuring an EGL display
00055                     //||< something failed.
00056     EGL_NO_SURFACE_ERROR, //||< When creating an EGL surface, no surface was
00057                     //||< created
00058     EGL_NO_CONTEXT_ERROR, //||< When creating an EGL context, no context was
00059                     //||< created.
00060     EGL_MAKE_CURRENT_ERROR, //||< When the making the EGL context, surface and
00061                     //||< display the current one, something failed.
00062     SDL_INIT_VIDEO_ERROR, //||< When initializing SDL video, something failed
00063     SDL_WINDOW_CREATION_ERROR, //||< When making a window with SDL2, something
00064                     //||< failed
00065     SDL_GL_CONTEXT_CREATION_ERROR, //||< When making an OpenGL context with SDL,
00066                     //||< something failed.
00067     SDL_GLAD_LOAD_ERROR, //||< When loading OpenGL functions with GLAD, using SDL,
00068                     //||< something failed
00069     EGL_GLAD_LOAD_ERROR, //||< When loading OpenGL functions with GLAD, using EGL,
00070                     //||< something failed
00071     SVEC_ZERO_ELEMENT_SIZE, //||< When initializing an SVec, the element size was
00072                     //||< set to zero
00073     SVEC_REALLOC_FAIL //||< When reallocating space for an SVec, something failed
00074 };
```

7.3.5 Function Documentation

7.3.5.1 print_error()

```
void print_error (
    CanimResult error)
```

Print out the error.

Parameters

error	The error to be printed.
-------	--------------------------

Definition at line 5 of file **core.c**.

```

00005
00006     if (!IS_AN_ERROR(error)) {
00007         return;
00008     }
00009     switch (error) {
00010     case GFX_DEVICE_CALLOC_ERROR:
00011         fprintf(stderr, "When using calloc to allocate memory for"
00012                 "a GfxDevice, calloc failed\n");
00013         break;
00014     case EGL_NO_DISPLAY_ERROR:
00015         fprintf(stderr, "When creating an EGL display, no display was created\n");
00016         break;
00017     case EGL_DISPLAY_INIT_ERROR:
00018         fprintf(stderr, "When initializing an EGL display something failed.\n");
00019         break;
00020     case EGL_DISPLAY_CONFIGURATION_ERROR:
00021         fprintf(stderr, "When configuring an EGL display something failed.\n");
00022         break;
00023     case EGL_NO_SURFACE_ERROR:
00024         fprintf(stderr, "When creating an EGL surface, no surface was created\n");
00025         break;
00026     case EGL_NO_CONTEXT_ERROR:
00027         fprintf(stderr, "When creating an EGL context, no context was created.\n");
00028         break;
00029     case EGL_MAKE_CURRENT_ERROR:
00030         fprintf(stderr,
00031                 "When the making the EGL context, surface and display the current "
00032                 "one, something failed.\n");
00033         break;
00034     case SDL_INIT_VIDEO_ERROR:
00035         fprintf(stderr, "When initializing SDL video, something failed\n");
00036         break;
00037     case SDL_WINDOW_CREATION_ERROR:
00038         fprintf(stderr, "When making a window with SDL2, something failed\n");
00039         break;
00040     case SDL_GL_CONTEXT_CREATION_ERROR:
00041         fprintf(stderr,
00042                 "When making an OpenGL context with SDL, something failed.\n");
00043         break;
00044     case SDL_GLAD_LOAD_ERROR:
00045         fprintf(stderr, "When loading OpenGL functions with GLAD, using SDL, "
00046                 "something failed\n");
00047         break;
00048     case EGL_GLAD_LOAD_ERROR:
00049         fprintf(stderr, "When loading OpenGL functions with GLAD, using EGL, "
00050                 "something failed\n");
00051         break;
00052     case SVEC_ZERO_ELEMENT_SIZE:
00053         fprintf(stderr,
00054                 "When initializing an SVec, the element size was set to zero\n");
00055         break;
00056     case SVEC_REALLOC_FAIL:
00057         fprintf(stderr, "When reallocating space for an SVec, something failed.\n");
00058         break;
00059
00060     default:
00061         fprintf(stderr, "SOMETHING BAD HAPPENED, WE DON'T KNOW WHAT\n");
00062         break;
00063     }
00064 }
```

7.3.5.2 read_be_f32()

```
float read_be_f32 (
    const unsigned char * p) [inline], [static]
```

Read a 32-bit float in big endian byte order.

Parameters

<i>p</i>	The buffer to be read
----------	-----------------------

Returns

The float

7.3.5.3 `read_be_f64()`

```
double read_be_f64 (
    const unsigned char * p) [inline], [static]
```

Read a 64-bit float in big endian byte order.

Parameters

<i>p</i>	The buffer to be read
----------	-----------------------

Returns

The float

7.3.5.4 `read_be_u16()`

```
uint16_t read_be_u16 (
    const unsigned char * p) [inline], [static]
```

Read a 16-bit unsigned integer in big endian byte order.

Parameters

<i>p</i>	Pointer to a buffer containing at least 2 bytes
----------	---

Returns

The 16-bit unsigned integer

7.3.5.5 `read_be_u32()`

```
uint32_t read_be_u32 (
    const unsigned char * p) [inline], [static]
```

Read a 32-bit unsigned integer in big endian byte order.

Parameters

<i>p</i>	Pointer to a buffer containing at least 4 bytes
----------	---

Returns

The 32-bit unsigned integer

7.3.5.6 read_be_u64()

```
uint64_t read_be_u64 (
    const unsigned char * p) [inline], [static]
```

Read a 64-bit unsigned integer in big endian byte order.

Parameters

<i>p</i>	Pointer to a buffer containing at least 8 bytes
----------	---

Returns

The 64-bit unsigned integer

7.3.5.7 read_le_f32()

```
float read_le_f32 (
    const unsigned char * p) [inline], [static]
```

Read a 32-bit float in little endian byte order.

Parameters

<i>p</i>	The buffer to be read
----------	-----------------------

Returns

The float

7.3.5.8 read_le_f64()

```
double read_le_f64 (
    const unsigned char * p) [inline], [static]
```

Read a 64-bit float in little endian byte order.

Parameters

<i>p</i>	The buffer to be read
----------	-----------------------

Returns

The float

7.3.5.9 `read_le_u16()`

```
uint16_t read_le_u16 (
    const unsigned char * p) [inline], [static]
```

Read a 16-bit unsigned integer in little endian byte order.

Parameters

<i>p</i>	Pointer to a buffer containing at least 2 bytes
----------	---

Returns

The 16-bit unsigned integer

7.3.5.10 `read_le_u32()`

```
uint32_t read_le_u32 (
    const unsigned char * p) [inline], [static]
```

Read a 32-bit unsigned integer in little endian byte order.

Parameters

<i>p</i>	Pointer to a buffer containing at least 4 bytes
----------	---

Returns

The 32-bit unsigned integer

7.3.5.11 `read_le_u64()`

```
uint64_t read_le_u64 (
    const unsigned char * p) [inline], [static]
```

Read a 64-bit unsigned integer in little endian byte order.

Parameters

<i>p</i>	Pointer to a buffer containing at least 8 bytes
----------	---

Returns

The 64-bit unsigned integer

7.3.5.12 svec_free()

```
void svec_free (
    CanimResult * result,
    SVec * v) [inline], [static]
```

Free all memory owned by the vector.

Parameters

<i>out</i>	<i>result</i>	A status code
	<i>v</i>	Pointer to SVec

7.3.5.13 svec_get()

```
void * svec_get (
    CanimResult * result,
    SVec * v,
    size_t i) [inline], [static]
```

Get a pointer to element at index *i*.

Parameters

<i>out</i>	<i>result</i>	A status code
	<i>v</i>	Pointer to SVec
	<i>i</i>	Index to read from

Returns

A pointer to the element

7.3.5.14 svec_init()

```
void svec_init (
    CanimResult * result,
    SVec * v,
    size_t elem_size) [inline], [static]
```

Initialize a vector with a fixed element size.

Parameters

<i>out</i>	<i>result</i>	A status code
	<i>v</i>	Pointer to SVec
	<i>elem_size</i>	Number of bytes per element

7.3.5.15 svec_pop()

```
void svec_pop (
    CanimResult * result,
    SVec * v,
    void * out) [inline], [static]
```

Remove the last element, shrinking bucket only if needed.

Parameters

<i>out</i>	<i>result</i>	A status code
	<i>v</i>	Pointer to SVec
	<i>out</i>	Optional output location

7.3.5.16 svec_push()

```
void svec_push (
    CanimResult * result,
    SVec * v,
    const void * elem) [inline], [static]
```

Append one element, reallocating only when bucket changes.

Parameters

<i>out</i>	<i>result</i>	A status code
	<i>v</i>	Pointer to SVec
	<i>elem</i>	Pointer to element data

7.3.5.17 svec_set()

```
void svec_set (
    CanimResult * result,
    SVec * v,
    size_t i,
    const void * elem) [inline], [static]
```

Set the element at index *i* to the value pointed to by *elem*.

Parameters

<i>out</i>	<i>result</i>	A status code
	<i>v</i>	Pointer to SVec
	<i>i</i>	Index to write to
	<i>elem</i>	Pointer to source data

7.3.5.18 write_be_f32()

```
void write_be_f32 (
    unsigned char * p,
    float f) [inline], [static]
```

Write a 32 bit float to a big endian buffer.

Parameters

<i>The</i>	buffer to be written to
<i>f</i>	The float

7.3.5.19 write_be_f64()

```
void write_be_f64 (
    unsigned char * p,
    double f) [inline], [static]
```

Write a 64 bit ouble to a big endian buffer.

Parameters

<i>The</i>	buffer to be written to
<i>f</i>	The double

7.3.5.20 write_be_u16()

```
void write_be_u16 (
    unsigned char * p,
    uint16_t v) [inline], [static]
```

Write a 16-bit unsigned integer in big endian byte order.

Parameters

<i>p</i>	The buffer to be written to
<i>v</i>	The integer to be converted

7.3.5.21 write_be_u32()

```
void write_be_u32 (
    unsigned char * p,
    uint32_t v) [inline], [static]
```

Write a 32-bit unsigned integer in big endian byte order.

Parameters

<i>p</i>	The buffer to be written to
<i>v</i>	The integer to be converted

7.3.5.22 write_be_u64()

```
void write_be_u64 (
    unsigned char * p,
    uint64_t v) [inline], [static]
```

Write a 64-bit unsigned integer in big endian byte order.

Parameters

<i>p</i>	The buffer to be written to
<i>v</i>	The integer to be converted

7.3.5.23 write_le_f32()

```
void write_le_f32 (
    unsigned char * p,
    float f) [inline], [static]
```

Write a 32 bit float to a little endian buffer.

Parameters

<i>The</i>	buffer to be written to
<i>f</i>	The float

7.3.5.24 write_le_f64()

```
void write_le_f64 (
    unsigned char * p,
    double f) [inline], [static]
```

Write a 64 bit double to a little endian buffer.

Parameters

<i>The</i>	buffer to be written to
<i>f</i>	The double

7.3.5.25 write_le_u16()

```
void write_le_u16 (
    unsigned char * p,
    uint16_t v) [inline], [static]
```

Write a 16-bit unsigned integer in little endian byte order.

Parameters

<i>p</i>	The buffer to be written to
<i>v</i>	The integer to be converted

7.3.5.26 write_le_u32()

```
void write_le_u32 (
    unsigned char * p,
    uint32_t v) [inline], [static]
```

Write a 32-bit unsigned integer in little endian byte order.

Parameters

<i>p</i>	The buffer to be written to
<i>v</i>	The integer to be converted

7.3.5.27 write_le_u64()

```
void write_le_u64 (
    unsigned char * p,
    uint64_t v) [inline], [static]
```

Write a 64-bit unsigned integer in little endian byte order.

Parameters

<i>p</i>	The buffer to be written to
<i>v</i>	The integer to be converted

7.4 core.h

[Go to the documentation of this file.](#)

```

00001 // SPDX-License-Identifier: MIT
00002 #pragma once
00003
00004 /// @file core.h
00005 /// @brief This is everything core to Canim that is shared between subsystems
00006
00007 #include <stdint.h>
00008 #include <sys/types.h>
00009 /// @def CANIM_API
00010 /// @brief Meant for a visibility label to all parts of the Canim API
00011 #define CANIM_API __attribute__((visibility("default")))
00012
00013 /// @def STATUS_TYPE_SHIFT
00014 /// @brief The bit shift applied to the value of an error to get it's type
00015 #define STATUS_TYPE_SHIFT 24
00016
00017 /// @def STATUS_TYPE_MASK
00018 /// @brief The mask applied to the an error to eliminate it's subtype.
00019 #define STATUS_TYPE_MASK 0xFF000000
00020
00021 /// @def SUCCESS
00022 /// @brief This is the status code for success
00023 #define SUCCESS 0x00
00024
00025 /// @def NOOP
00026 /// @brief No operation performed.
00027 #define NOOP 0x01
00028
00029 /// @def NONFATAL
00030 /// @brief Non-fatal error (unused)
00031 #define NONFATAL 0x02
00032
00033 /// @def FATAL
00034 /// @brief Fatal error.
00035 #define FATAL 0x03
00036
00037 /// @def STATUS_TYPE
00038 /// @brief Extracts the status type (SUCCESS/NOOP/NONFATAL/FATAL).
00039 #define STATUS_TYPE(code) (((code) & STATUS_TYPE_MASK) » STATUS_TYPE_SHIFT)
00040
00041 /// @def IS_AN_ERROR
00042 /// @brief True if code is an error (NONFATAL or FATAL).
00043 #define IS_AN_ERROR(x) \
00044     ((STATUS_TYPE((x)) == FATAL) || (STATUS_TYPE((x)) == NONFATAL))
00045
00046 /// @brief This are all of the error codes
00047 enum {
00048     GFX_DEVICE_CALLOC_ERROR =
00049         (uint32_t)0x03000000, //;< When using calloc to allocate memory for a
00050                     //;< GfxDevice, calloc failed
00051     EGL_NO_DISPLAY_ERROR, //;< When creating an EGL display, no display was
00052                     //;< created
00053     EGL_DISPLAY_INIT_ERROR, //;< When initializing an EGL display something failed
00054     EGL_DISPLAY_CONFIGURATION_ERROR, //;< When configuring an EGL display
00055                     //;< something failed.
00056     EGL_NO_SURFACE_ERROR, //;< When creating an EGL surface, no surface was
00057                     //;< created
00058     EGL_NO_CONTEXT_ERROR, //;< When creating an EGL context, no context was
00059                     //;< created.
00060     EGL_MAKE_CURRENT_ERROR, //;< When the making the EGL context, surface and
00061                     //;< display the current one, something failed.
00062     SDL_INIT_VIDEO_ERROR, //;< When initializing SDL video, something failed
00063     SDL_WINDOW_CREATION_ERROR, //;< When making a window with SDL2, something
00064                     //;< failed
00065     SDL_GL_CONTEXT_CREATION_ERROR, //;< When making an OpenGL context with SDL,
00066                     //;< something failed.
00067     SDL_GLAD_LOAD_ERROR, //;< When loading OpenGL functions with GLAD, using SDL,
00068                     //;< something failed
00069     EGL_GLAD_LOAD_ERROR, //;< When loading OpenGL functions with GLAD, using EGL,
00070                     //;< something failed
00071     SVEC_ZERO_ELEMENT_SIZE, //;< When initializing an SVec, the element size was
00072                     //;< set to zero
00073     SVEC_REALLOC_FAIL //;< When reallocating space for an SVec, something failed
00074 };
00075
00076 /// @brief Result code used across Canim operations.
00077 typedef uint32_t CanimResult;
00078
00079 /// @brief Print out the error
00080 /// @param error The error to be printed.
00081 void print_error(CanimResult error);
00082

```

```
00083 /// @def CANIM_PLATFORM_WINDOWS
00084 /// @brief Defined on Windows (_WIN32).
00085 #ifdef _WIN32
00086 #define CANIM_PLATFORM_WINDOWS 1
00087 #define CANIM_PLATFORM_KNOWN 1
00088 #endif
00089
00090 /// @def CANIM_PLATFORM_MAC
00091 /// @brief Defined on macOS (__APPLE__).
00092 #ifdef __APPLE__
00093 #define CANIM_PLATFORM_MAC 1
00094 #define CANIM_PLATFORM_KNOWN 1
00095 #endif
00096
00097 /// @def CANIM_PLATFORM_LINUX
00098 /// @brief Defined on Linux (__linux__).
00099 #ifdef __linux__
00100 #define CANIM_PLATFORM_LINUX 1
00101 #define CANIM_PLATFORM_KNOWN 1
00102 #endif
00103
00104 /// @def CANIM_PLATFORM_POSIX
00105 /// @brief Defined on POSIX platforms (macOS or Linux).
00106 #if defined(CANIM_PLATFORM_LINUX) || defined(CANIM_PLATFORM_MAC)
00107 #define CANIM_PLATFORM_POSIX 1
00108 #define CANIM_PLATFORM_KNOWN 1
00109 #endif
00110
00111 /// @def CANIM_PLATFORM_UNKNOWN
00112 /// @brief Defined if no known platform could be detected.
00113 #ifndef CANIM_PLATFORM_KNOWN
00114 #define CANIM_PLATFORM_UNKNOWN 1
00115 #endif
00116
00117 /// @def max
00118 /// @brief A max macro
00119 #ifndef max
00120 #define max(a, b) (((a) > (b)) ? (a) : (b))
00121 #endif
00122
00123 /// @def EPSILON
00124 /// @brief Tolerance for floating-point comparisons.
00125 #define EPSILON 0.000001
00126
00127 /// @def null
00128 /// @brief I do not want to capitalize NULL
00129 #define null NULL
00130
00131 /// @def BUFFER_SIZE
00132 /// @brief The size of a buffer
00133 #define BUFFER_SIZE 4096
00134
00135 /// @def POPEN
00136 /// @brief A cross platform alias for opening a pipe.
00137 #ifdef CANIM_PLATFORM_WINDOWS
00138 #define POPEN _popen
00139 #else
00140 #define POPEN popen
00141 #endif
00142
00143 /// @def PCLOSE
00144 /// @brief A cross platform alias for closing a pipe.
00145 #ifdef CANIM_PLATFORM_WINDOWS
00146 #define PCLOSE _pclose
00147 #else
00148 #define PCLOSE pclose
00149 #endif
00150
00151 /// @def WB
00152 /// @brief Correct mode for writing in binary files on each platform
00153 #ifdef CANIM_PLATFORM_WINDOWS
00154 #define WB "wb"
00155 #else
00156 #define WB "w"
00157 #endif
00158
00159 /// @brief Read a 16-bit unsigned integer in big endian byte order
00160 /// @param p Pointer to a buffer containing at least 2 bytes
00161 /// @return The 16-bit unsigned integer
00162 static inline uint16_t read_be_u16(const unsigned char *p);
00163
00164 /// @brief Read a 32-bit unsigned integer in big endian byte order
00165 /// @param p Pointer to a buffer containing at least 4 bytes
00166 /// @return The 32-bit unsigned integer
00167 static inline uint32_t read_be_u32(const unsigned char *p);
00168
00169 /// @brief Read a 64-bit unsigned integer in big endian byte order
```

```
00170 /// @param p Pointer to a buffer containing at least 8 bytes
00171 /// @return The 64-bit unsigned integer
00172 static inline uint64_t read_be_u64(const unsigned char *p);
00173
00174 /// @brief Read a 16-bit unsigned integer in little endian byte order
00175 /// @param p Pointer to a buffer containing at least 2 bytes
00176 /// @return The 16-bit unsigned integer
00177 static inline uint16_t read_le_u16(const unsigned char *p);
00178
00179 /// @brief Read a 32-bit unsigned integer in little endian byte order
00180 /// @param p Pointer to a buffer containing at least 4 bytes
00181 /// @return The 32-bit unsigned integer
00182 static inline uint32_t read_le_u32(const unsigned char *p);
00183
00184 /// @brief Read a 64-bit unsigned integer in little endian byte order
00185 /// @param p Pointer to a buffer containing at least 8 bytes
00186 /// @return The 64-bit unsigned integer
00187 static inline uint64_t read_le_u64(const unsigned char *p);
00188
00189 /// @brief Write a 16-bit unsigned integer in big endian byte order
00190 /// @param p The buffer to be written to
00191 /// @param v The integer to be converted
00192 static inline void write_be_u16(unsigned char *p, uint16_t v);
00193
00194 /// @brief Write a 32-bit unsigned integer in big endian byte order
00195 /// @param p The buffer to be written to
00196 /// @param v The integer to be converted
00197 static inline void write_be_u32(unsigned char *p, uint32_t v);
00198
00199 /// @brief Write a 64-bit unsigned integer in big endian byte order
00200 /// @param p The buffer to be written to
00201 /// @param v The integer to be converted
00202 static inline void write_be_u64(unsigned char *p, uint64_t v);
00203
00204 /// @brief Write a 16-bit unsigned integer in little endian byte order
00205 /// @param p The buffer to be written to
00206 /// @param v The integer to be converted
00207 static inline void write_le_u16(unsigned char *p, uint16_t v);
00208
00209 /// @brief Write a 32-bit unsigned integer in little endian byte order
00210 /// @param p The buffer to be written to
00211 /// @param v The integer to be converted
00212 static inline void write_le_u32(unsigned char *p, uint32_t v);
00213
00214 /// @brief Write a 64-bit unsigned integer in little endian byte order
00215 /// @param p The buffer to be written to
00216 /// @param v The integer to be converted
00217 static inline void write_le_u64(unsigned char *p, uint64_t v);
00218
00219 /// @brief Read a 32-bit float in big endian byte order
00220 /// @param p The buffer to be read
00221 /// @return The float
00222 static inline float read_be_f32(const unsigned char *p);
00223
00224 /// @brief Read a 64-bit float in big endian byte order
00225 /// @param p The buffer to be read
00226 /// @return The float
00227 static inline double read_be_f64(const unsigned char *p);
00228
00229 /// @brief Read a 32-bit float in little endian byte order
00230 /// @param p The buffer to be read
00231 /// @return The float
00232 static inline float read_le_f32(const unsigned char *p);
00233
00234 /// @brief Read a 64-bit float in little endian byte order
00235 /// @param p The buffer to be read
00236 /// @return The float
00237 static inline double read_le_f64(const unsigned char *p);
00238
00239 /// @brief Write a 32 bit float to a big endian buffer
00240 /// @param The buffer to be written to
00241 /// @param f The float
00242 static inline void write_be_f32(unsigned char *p, float f);
00243
00244 /// @brief Write a 64 bit ouble to a big endian buffer
00245 /// @param The buffer to be written to
00246 /// @param f The double
00247 static inline void write_be_f64(unsigned char *p, double f);
00248
00249 /// @brief Write a 32 bit float to a little endian buffer
00250 /// @param The buffer to be written to
00251 /// @param f The float
00252 static inline void write_le_f32(unsigned char *p, float f);
00253
00254 /// @brief Write a 64 bit ouble to a little endian buffer
00255 /// @param The buffer to be written to
00256 /// @param f The double
```

```

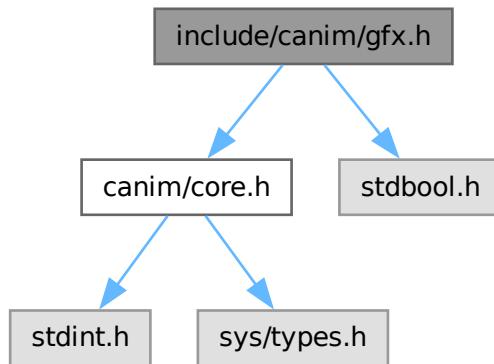
00257 static inline void write_le_f64(unsigned char *p, double f);
00258
00259 /// @def BIT_IGNORE
00260 /// @brief Alignment granularity in bits (round up to 2^BIT_IGNORE).
00261 #define BIT_IGNORE 4
00262
00263 /// @def BIT_SIZE
00264 /// @brief 2^BIT_IGNORE - 1
00265 #define BIT_SIZE ((lu << BIT_IGNORE) - lu)
00266
00267 /// @def BIT_ALIGN(x)
00268 /// @brief Round x up to nearest aligned multiple of 2^BIT_IGNORE. Always
00269 #define BIT_ALIGN(x) \
00270   (((x) + BIT_SIZE) & ~BIT_SIZE)) ? (((x) + BIT_SIZE) & ~BIT_SIZE) : 1) \
00271
00272 /// @def REALIGN(a, b)
00273 /// @brief True if a and b land in different aligned capacity buckets.
00274 #define REALIGN(a, b) (BIT_ALIGN(a) != BIT_ALIGN(b))
00275
00276 /// @struct SVec
00277 /// @brief Minimal growable byte-vector.
00278 typedef struct {
00279   size_t element_size; /////< Size of each element in bytes.
00280   size_t list_size; /////< Number of elements stored.
00281   unsigned char *data; /////< Contiguous raw byte storage
00282 } SVec;
00283
00284 /// @brief Initialize a vector with a fixed element size.
00285 /// @param[out] result A status code
00286 /// @param v Pointer to SVec
00287 /// @param elem_size Number of bytes per element
00288 static inline void svec_init(CanimResult *result, SVec *v, size_t elem_size);
00289
00290 /// @brief Free all memory owned by the vector.
00291 /// @param[out] result A status code
00292 /// @param v Pointer to SVec
00293 static inline void svec_free(CanimResult *result, SVec *v);
00294
00295 /// @brief Append one element, reallocating only when bucket changes.
00296 /// @param[out] result A status code
00297 /// @param v Pointer to SVec
00298 /// @param elem Pointer to element data
00299 static inline void svec_push(CanimResult *result, SVec *v, const void *elem);
00300
00301 /// @brief Remove the last element, shrinking bucket only if needed.
00302 /// @param[out] result A status code
00303 /// @param v Pointer to SVec
00304 /// @param out Optional output location
00305 static inline void svec_pop(CanimResult *result, SVec *v, void *out);
00306
00307 /// @brief Get a pointer to element at index i
00308 /// @param[out] result A status code
00309 /// @param v Pointer to SVec
00310 /// @param i Index to read from
00311 /// @return A pointer to the element
00312 static inline void *svec_get(CanimResult *result, SVec *v, size_t i);
00313
00314 /// @brief Set the element at index i to the value pointed to by elem.
00315 /// @param[out] result A status code
00316 /// @param v Pointer to SVec
00317 /// @param i Index to write to
00318 /// @param elem Pointer to source data
00319 static inline void svec_set(CanimResult *result, SVec *v, size_t i,
00320                           const void *elem);

```

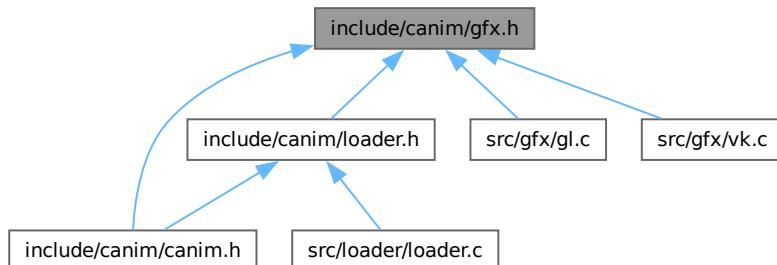
7.5 include/canim/gfx.h File Reference

```
#include "canim/core.h"
#include <stdbool.h>
```

Include dependency graph for gfx.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [GfxInitInfo](#)
This struct stores info necessary to initialize graphics.
- struct [GfxAPI](#)
This struct stores the cross platform API functions. This struct has a fptr, but I am not gonna document it, because I believe in freedom and you can't stop me.
- struct [GfxContainer](#)

TypeDefs

- typedef struct GfxDevice [GfxDevice](#)
GFX API Specific struct.
- typedef struct GfxContainer [GfxContainer](#)
This container contains the graphics stuff.
- typedef struct GfxAPI [GfxAPI](#)
This stores the function pointers for implementation of gfx api's.

Enumerations

- enum [GfxBackend](#) { [CANIM_GFX_NONE](#) = 0 , [CANIM_GFX_GL](#) = 1 }
- This is a list of available backends.*

7.5.1 Typedef Documentation

7.5.1.1 GfxAPI

```
typedef struct GfxAPI GfxAPI
```

This stores the function pointers for implementation of gfx api's.

Definition at line [20](#) of file [gfx.h](#).

7.5.1.2 GfxContainer

```
typedef struct GfxContainer GfxContainer
```

This container contains the graphics stuff.

Definition at line [17](#) of file [gfx.h](#).

7.5.1.3 GfxDevice

```
typedef struct GfxDevice GfxDevice
```

GFX API Specific struct.

Definition at line [14](#) of file [gfx.h](#).

7.5.2 Enumeration Type Documentation

7.5.2.1 GfxBackend

```
enum GfxBackend
```

This is a list of available backends.

Enumerator

CANIM_GFX_NONE	No Graphics API used.
CANIM_GFX_GL	OpenGL in use.

Definition at line [8](#) of file [gfx.h](#).

```
00008 {
00009   CANIM\_GFX\_NONE = 0, ///< No Graphics API used
0010   CANIM\_GFX\_GL = 1, ///< OpenGL in use
0011 } GfxBackend;
```

7.6 gfx.h

[Go to the documentation of this file.](#)

```

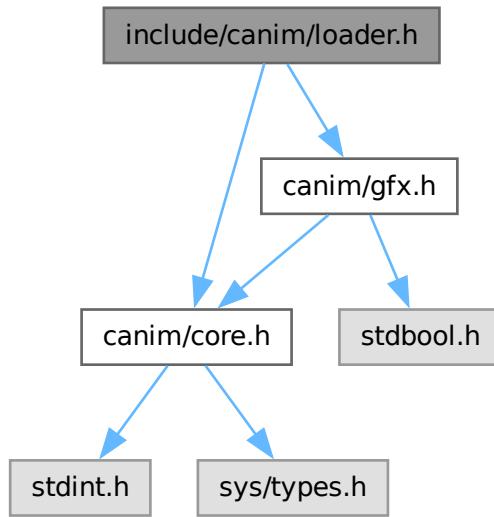
00001 // SPDX-License-Identifier: MIT
00002 #pragma once
00003 #include "canim/core.h"
00004 #include <stdbool.h>
00005
00006 /// @enum GfxBackend
00007 /// @brief This is a list of available backends
00008 typedef enum {
00009     CANIM_GFX_NONE = 0, ///< No Graphics API used
0010     CANIM_GFX_GL = 1,   ///< OpenGL in use
0011 } GfxBackend;
0012
0013 /// @brief GFX API Specific struct
0014 typedef struct GfxDevice GfxDevice;
0015
0016 /// @brief This container contains the graphics stuff
0017 typedef struct GfxContainer GfxContainer;
0018
0019 /// @brief This stores the function pointers for implementation of gfx api's
0020 typedef struct GfxAPI GfxAPI;
0021
0022 /// @struct GfxInitInfo
0023 /// @brief This struct stores info neccesary to initialize graphics
0024 typedef struct {
0025     bool headless;      ///< This stores whether or not the window is headless
0026     void *native_window; ///< This stores a pointer to a native window
0027     int width;          ///< The width of the window
0028     int height;         ///< The height of it
0029 } GfxInitInfo;
0030
0031 /// @struct GfxAPI
0032 /// @brief This struct stores the cross platform API functions
0033 /// This struct has a fptr, but i am not gonna document it, because i believe in
0034 /// freedom and you can't stop me
0035 struct GfxAPI {
0036     GfxDevice *(*gfx_create_device)(CanimResult *, GfxContainer *,
0037                                     const GfxInitInfo *);
0038     void (*gfx_destroy_device)(CanimResult *, GfxContainer *);
0039 };
0040
0041 /// @struct GfxContainer
0042 struct GfxContainer {
0043     GfxAPI api;          ///< The API
0044     GfxDevice *impl;     ///< The Implementation specfic gfxdevice
0045     GfxBackend backend;  ///< The backend
0046     void *handle;        ///< Handle to the dynamic lib
0047 };

```

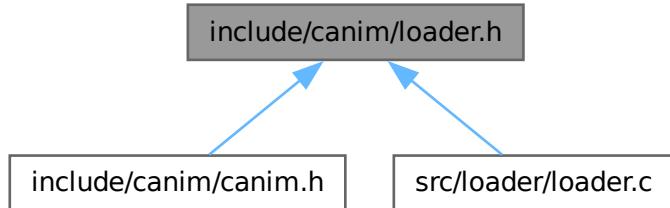
7.7 include/canim/loader.h File Reference

```
#include "canim/core.h"
#include "canim/gfx.h"
```

Include dependency graph for loader.h:



This graph shows which files directly or indirectly include this file:



Functions

- **CANIM_API GfxContainer * gfx_load_backend** (CanimResult *result, GfxBackend backend, const GfxInitInfo *info)

This loads a backend and makes a graphics container.
- **CANIM_API void gfx_unload_backend** (CanimResult *result, GfxContainer *gfx)

This unloads a backend.

7.7.1 Function Documentation

7.7.1.1 gfx_load_backend()

```
CANIM_API GfxContainer * gfx_load_backend (
    CanimResult * result,
    GfxBackend backend,
    const GfxInitInfo * info)
```

This loads a backend and makes a graphics container.

Parameters

<i>result</i>	This is the result of the output
<i>backend</i>	This is the gfx api
<i>info</i>	The relevant info to create the container

Returns

A gfx container

Definition at line 36 of file [loader.c](#).

```
00037
00038     const char *libname = gfx_backend_libname(backend);
00039     LIB_HANDLE handle = LIB_LOAD(libname);
00040     const GfxAPI *const *entry =
00041         (const GfxAPI *const *)LIB_SYM(handle, "GFX_API_ENTRY");
00042     GfxContainer *gfx = calloc(1, sizeof(*gfx));
00043     const GfxAPI *api = *entry;
00044     gfx->api = api;
00045     gfx->impl = NULL;
00046     gfx->backend = backend;
00047     GfxDevice *dev = gfx->api.gfx_create_device(result, gfx, info);
00048     if (IS_AN_ERROR(*result)) {
00049         return NULL;
00050     }
00051     gfx->handle = (void *)handle;
00052     gfx->impl = dev;
00053     *result = SUCCESS;
00054     return gfx;
00055 }
```

Here is the call graph for this function:



7.7.1.2 `gfx_unload_backend()`

```
CANIM_API void gfx_unload_backend (
    CanimResult * result,
    GfxContainer * gfx)
```

This unloads a backend.

Parameters

<i>result</i>	This is the result of the output
<i>gfx</i>	This is the container

Definition at line 57 of file [loader.c](#).

```
00057
00058     LIB_CLOSE((LIB_HANDLE)gfx->handle);
00059     *result = SUCCESS;
00060 }
```

7.8 loader.h

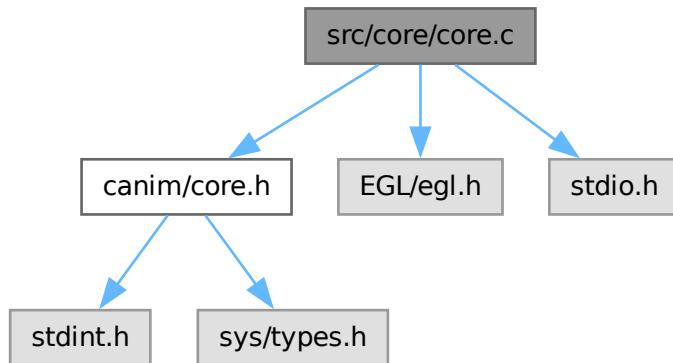
[Go to the documentation of this file.](#)

```
00001 // SPDX-License-Identifier: MIT
00002 #pragma once
00003 #include "canim/core.h"
00004 #include "canim/gfx.h"
00005
00006 /// @brief This loads a backend and makes a graphics container
00007 /// @param result This is the result of the output
00008 /// @param backend This is the gfx api
00009 /// @param info The relevant info to create the container
00010 /// @return A gfx container
00011 CANIM_API GfxContainer *gfx_load_backend(CanimResult *result,
00012                                             GfxBackend backend,
00013                                             const GfxInitInfo *info);
00014
00015 /// @brief This unloads a backend
00016 /// @param result This is the result of the output
00017 /// @param gfx This is the container
00018 CANIM_API void gfx_unload_backend(CanimResult *result, GfxContainer *gfx);
```

7.9 src/core/core.c File Reference

```
#include "canim/core.h"
#include <EGL/egl.h>
#include <stdio.h>
```

Include dependency graph for core.c:



Functions

- void **print_error** (**CanimResult** error)

Print out the error.

7.9.1 Function Documentation

7.9.1.1 print_error()

```
void print_error (
    CanimResult error)
```

Print out the error.

Parameters

error	The error to be printed.
--------------	--------------------------

Definition at line 5 of file **core.c**.

```
00005
00006     if (!IS_AN_ERROR(error)) {
00007         return;
00008     }
00009     switch (error) {
00010     case GFX_DEVICE_CALLOC_ERROR:
00011         fprintf(stderr, "When using calloc to allocate memory for"
00012                 "a GfxDevice, calloc failed\n");
00013         break;
00014     case EGL_NO_DISPLAY_ERROR:
00015         fprintf(stderr, "When creating an EGL display, no display was created\n");
00016         break;
00017     case EGL_DISPLAY_INIT_ERROR:
00018         fprintf(stderr, "When initializing an EGL display something failed.\n");
00019         break;
00020     case EGL_DISPLAY_CONFIGURATION_ERROR:
00021         fprintf(stderr, "When configuring an EGL display something failed.\n");
00022         break;
00023     case EGL_NO_SURFACE_ERROR:
00024         fprintf(stderr, "When creating an EGL surface, no surface was created\n");
00025         break;
00026     case EGL_NO_CONTEXT_ERROR:
00027         fprintf(stderr, "When creating an EGL context, no context was created.\n");
00028         break;
00029     case EGL_MAKE_CURRENT_ERROR:
00030         fprintf(stderr,
00031                 "When the making the EGL context, surface and display the current "
00032                 "one, something failed.\n");
00033         break;
00034     case SDL_INIT_VIDEO_ERROR:
00035         fprintf(stderr, "When initializing SDL video, something failed\n");
00036         break;
00037     case SDL_WINDOW_CREATION_ERROR:
00038         fprintf(stderr, "When making a window with SDL2, something failed\n");
00039         break;
00040     case SDL_GL_CONTEXT_CREATION_ERROR:
00041         fprintf(stderr,
00042                 "When making an OpenGL context with SDL, something failed.\n");
00043         break;
00044     case SDL_GLAD_LOAD_ERROR:
00045         fprintf(stderr, "When loading OpenGL functions with GLAD, using SDL, "
00046                 "something failed\n");
00047         break;
00048     case EGL_GLAD_LOAD_ERROR:
00049         fprintf(stderr, "When loading OpenGL functions with GLAD, using EGL, "
00050                 "something failed\n");
00051         break;
00052     case SVEC_ZERO_ELEMENT_SIZE:
00053         fprintf(stderr,
00054                 "When initializing an SVec, the element size was set to zero\n");
00055         break;
```

```

00056     case SVEC_REALLOC_FAIL:
00057         fprintf(stderr, "When reallocating space for an SVec, something failed.\n");
00058         break;
00059     default:
00060         fprintf(stderr, "SOMETHING BAD HAPPENED, WE DON'T KNOW WHAT\n");
00061         break;
00062     }
00063 }
00064 }
```

7.10 core.c

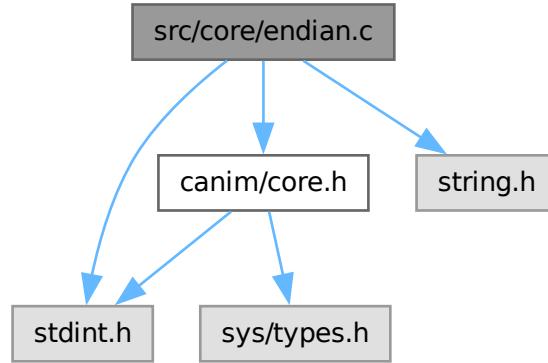
[Go to the documentation of this file.](#)

```

00001 // SPDX-License-Identifier: MIT
00002 #include "canim/core.h"
00003 #include <EGL/egl.h>
00004 #include <stdio.h>
00005 void print_error(CanimResult error) {
00006     if (!IS_AN_ERROR(error)) {
00007         return;
00008     }
00009     switch (error) {
00010         case GFX_DEVICE_CALLOC_ERROR:
00011             fprintf(stderr, "When using calloc to allocate memory for"
00012                     "a GfxDevice, calloc failed\n");
00013             break;
00014         case EGL_NO_DISPLAY_ERROR:
00015             fprintf(stderr, "When creating an EGL display, no display was created\n");
00016             break;
00017         case EGL_DISPLAY_INIT_ERROR:
00018             fprintf(stderr, "When initializing an EGL display something failed.\n");
00019             break;
00020         case EGL_DISPLAY_CONFIGURATION_ERROR:
00021             fprintf(stderr, "When configuring an EGL display something failed.\n");
00022             break;
00023         case EGL_NO_SURFACE_ERROR:
00024             fprintf(stderr, "When creating an EGL surface, no surface was created\n");
00025             break;
00026         case EGL_NO_CONTEXT_ERROR:
00027             fprintf(stderr, "When creating an EGL context, no context was created.\n");
00028             break;
00029         case EGL_MAKE_CURRENT_ERROR:
00030             fprintf(stderr,
00031                     "When the making the EGL context, surface and display the current "
00032                     "one, something failed.\n");
00033             break;
00034         case SDL_INIT_VIDEO_ERROR:
00035             fprintf(stderr, "When initializing SDL video, something failed\n");
00036             break;
00037         case SDL_WINDOW_CREATION_ERROR:
00038             fprintf(stderr, "When making a window with SDL2, something failed\n");
00039             break;
00040         case SDL_GL_CONTEXT_CREATION_ERROR:
00041             fprintf(stderr,
00042                     "When making an OpenGL context with SDL, something failed.\n");
00043             break;
00044         case SDL_GLAD_LOAD_ERROR:
00045             fprintf(stderr, "When loading OpenGL functions with GLAD, using SDL, "
00046                     "something failed\n");
00047             break;
00048         case EGL_GLAD_LOAD_ERROR:
00049             fprintf(stderr, "When loading OpenGL functions with GLAD, using EGL, "
00050                     "something failed\n");
00051             break;
00052         case SVEC_ZERO_ELEMENT_SIZE:
00053             fprintf(stderr,
00054                     "When initializing an SVec, the element size was set to zero\n");
00055             break;
00056         case SVEC_REALLOC_FAIL:
00057             fprintf(stderr, "When reallocating space for an SVec, something failed.\n");
00058             break;
00059     default:
00060         fprintf(stderr, "SOMETHING BAD HAPPENED, WE DON'T KNOW WHAT\n");
00061         break;
00062     }
00063 }
00064 }
```

7.11 src/core/endian.c File Reference

```
#include "canim/core.h"
#include <stdint.h>
#include <string.h>
Include dependency graph for endian.c:
```



Functions

- static uint16_t `read_be_u16` (const unsigned char *p)
- static uint32_t `read_be_u32` (const unsigned char *p)
- static uint64_t `read_be_u64` (const unsigned char *p)
- static uint16_t `read_le_u16` (const unsigned char *p)
- static uint32_t `read_le_u32` (const unsigned char *p)
- static uint64_t `read_le_u64` (const unsigned char *p)
- static void `write_be_u16` (unsigned char *p, uint16_t v)
- static void `write_be_u32` (unsigned char *p, uint32_t v)
- static void `write_be_u64` (unsigned char *p, uint64_t v)
- static void `write_le_u16` (unsigned char *p, uint16_t v)
- static void `write_le_u32` (unsigned char *p, uint32_t v)
- static void `write_le_u64` (unsigned char *p, uint64_t v)
- static float `read_be_f32` (const unsigned char *p)
- static float `read_le_f32` (const unsigned char *p)
- static void `write_be_f32` (unsigned char *p, float f)
- static void `write_le_f32` (unsigned char *p, float f)
- static double `read_be_f64` (const unsigned char *p)
- static double `read_le_f64` (const unsigned char *p)
- static void `write_be_f64` (unsigned char *p, double d)
- static void `write_le_f64` (unsigned char *p, double d)

7.11.1 Function Documentation

7.11.1.1 read_be_f32()

```
float read_be_f32 (
    const unsigned char * p) [inline], [static]
```

Definition at line 84 of file [endian.c](#).

```
00084
00085     uint32_t u = read\_be\_u32(p);
00086     float f;
00087     memcpy(&f, &u, sizeof(f));
00088     return f;
00089 }
```

Here is the call graph for this function:



7.11.1.2 read_be_f64()

```
double read_be_f64 (
    const unsigned char * p) [inline], [static]
```

Definition at line 110 of file [endian.c](#).

```
00110
00111     uint64_t u = read\_be\_u64(p);
00112     double d;
00113     memcpy(&d, &u, sizeof(d));
00114     return d;
00115 }
```

Here is the call graph for this function:



7.11.1.3 `read_be_u16()`

```
uint16_t read_be_u16 (
    const unsigned char * p) [inline], [static]
```

Definition at line 6 of file [endian.c](#).

```
00006
00007     return ((uint16_t)p[0] << 010) | ((uint16_t)p[1] << 000);
00008 }
```

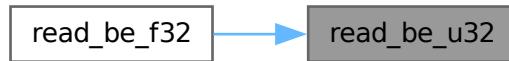
7.11.1.4 `read_be_u32()`

```
uint32_t read_be_u32 (
    const unsigned char * p) [inline], [static]
```

Definition at line 10 of file [endian.c](#).

```
00010
00011     return ((uint32_t)p[0] << 030) | ((uint32_t)p[1] << 020) |
00012         ((uint32_t)p[2] << 010) | ((uint32_t)p[3] << 000);
00013 }
```

Here is the caller graph for this function:



7.11.1.5 `read_be_u64()`

```
uint64_t read_be_u64 (
    const unsigned char * p) [inline], [static]
```

Definition at line 15 of file [endian.c](#).

```
00015
00016     return ((uint64_t)p[0] << 070) | ((uint64_t)p[1] << 060) |
00017         ((uint64_t)p[2] << 050) | ((uint64_t)p[3] << 040) |
00018         ((uint64_t)p[4] << 030) | ((uint64_t)p[5] << 020) |
00019         ((uint64_t)p[6] << 010) | ((uint64_t)p[7] << 000);
00020 }
```

Here is the caller graph for this function:



7.11.1.6 `read_le_f32()`

```
float read_le_f32 (
    const unsigned char * p) [inline], [static]
```

Definition at line 91 of file [endian.c](#).

```
00091
00092     uint32_t u = read\_le\_u32(p);
00093     float f;
00094     memcpy(&f, &u, sizeof(f));
00095     return f;
00096 }
```

Here is the call graph for this function:



7.11.1.7 `read_le_f64()`

```
double read_le_f64 (
    const unsigned char * p) [inline], [static]
```

Definition at line 117 of file [endian.c](#).

```
00117
00118     uint64_t u = read\_le\_u64(p);
00119     double d;
00120     memcpy(&d, &u, sizeof(d));
00121     return d;
00122 }
```

Here is the call graph for this function:



7.11.1.8 `read_le_u16()`

```
uint16_t read_le_u16 (
    const unsigned char * p) [inline], [static]
```

Definition at line 22 of file [endian.c](#).

```
00022
00023     return ((uint16_t)p[1] << 010) | ((uint16_t)p[0] << 000);
00024 }
```

7.11.1.9 read_le_u32()

```
uint32_t read_le_u32 (
    const unsigned char * p) [inline], [static]
```

Definition at line 26 of file [endian.c](#).

```
00026
00027     return ((uint32_t)p[3] << 030) | ((uint32_t)p[2] << 020) |
00028         ((uint32_t)p[1] << 010) | ((uint32_t)p[0] << 000);
00029 }
```

Here is the caller graph for this function:



7.11.1.10 read_le_u64()

```
uint64_t read_le_u64 (
    const unsigned char * p) [inline], [static]
```

Definition at line 31 of file [endian.c](#).

```
00031
00032     return ((uint64_t)p[7] << 070) | ((uint64_t)p[6] << 060) |
00033         ((uint64_t)p[5] << 050) | ((uint64_t)p[4] << 040) |
00034         ((uint64_t)p[3] << 030) | ((uint64_t)p[2] << 020) |
00035         ((uint64_t)p[1] << 010) | ((uint64_t)p[0] << 000);
00036 }
```

Here is the caller graph for this function:



7.11.1.11 write_be_f32()

```
void write_be_f32 (
    unsigned char * p,
    float f) [inline], [static]
```

Definition at line 98 of file [endian.c](#).

```
00098
00099     uint32_t u;
00100     memcpy(&u, &f, sizeof(u));
00101     write_be_u32(p, u);
00102 }
```

Here is the call graph for this function:



7.11.1.12 write_be_f64()

```
void write_be_f64 (
    unsigned char * p,
    double d) [inline], [static]
```

Definition at line 124 of file [endian.c](#).

```
00124
00125     uint64_t u;
00126     memcpy(&u, &d, sizeof(u));
00127     write_be_u64(p, u);
00128 }
```

Here is the call graph for this function:



7.11.1.13 write_be_u16()

```
void write_be_u16 (
    unsigned char * p,
    uint16_t v) [inline], [static]
```

Definition at line 38 of file [endian.c](#).

```
00038
00039     p[0] = (unsigned char)(v >> 010);
00040     p[1] = (unsigned char)(v >> 000);
00041 }
```

7.11.1.14 write_be_u32()

```
void write_be_u32 (
    unsigned char * p,
    uint32_t v) [inline], [static]
```

Definition at line 43 of file [endian.c](#).

```
00043
00044     p[0] = (unsigned char)(v >> 030);
00045     p[1] = (unsigned char)(v >> 020);
00046     p[2] = (unsigned char)(v >> 010);
00047     p[3] = (unsigned char)(v >> 000);
00048 }
```

Here is the caller graph for this function:



7.11.1.15 write_be_u64()

```
void write_be_u64 (
    unsigned char * p,
    uint64_t v) [inline], [static]
```

Definition at line 50 of file [endian.c](#).

```
00050
00051     p[0] = (unsigned char)(v >> 070);
00052     p[1] = (unsigned char)(v >> 060);
00053     p[2] = (unsigned char)(v >> 050);
00054     p[3] = (unsigned char)(v >> 040);
00055     p[4] = (unsigned char)(v >> 030);
00056     p[5] = (unsigned char)(v >> 020);
00057     p[6] = (unsigned char)(v >> 010);
00058     p[7] = (unsigned char)(v >> 000);
00059 }
```

Here is the caller graph for this function:



7.11.1.16 write_le_f32()

```
void write_le_f32 (
    unsigned char * p,
    float f) [inline], [static]
```

Definition at line 104 of file [endian.c](#).

```
00104
00105     uint32_t u;
00106     memcpy(&u, &f, sizeof(u));
00107     write_le_u32(p, u);
00108 }
```

Here is the call graph for this function:



7.11.1.17 write_le_f64()

```
void write_le_f64 (
    unsigned char * p,
    double d) [inline], [static]
```

Definition at line 130 of file [endian.c](#).

```
00130
00131     uint64_t u;
00132     memcpy(&u, &d, sizeof(u));
00133     write_le_u64(p, u);
00134 }
```

Here is the call graph for this function:



7.11.1.18 write_le_u16()

```
void write_le_u16 (
    unsigned char * p,
    uint16_t v) [inline], [static]
```

Definition at line 61 of file [endian.c](#).

```
00061
00062     p[0] = (unsigned char)(v >> 000);
00063     p[1] = (unsigned char)(v >> 010);
00064 }
```

7.11.1.19 write_le_u32()

```
void write_le_u32 (
    unsigned char * p,
    uint32_t v) [inline], [static]
```

Definition at line 66 of file [endian.c](#).

```
00066
00067     p[0] = (unsigned char)(v >> 000);
00068     p[1] = (unsigned char)(v >> 010);
00069     p[2] = (unsigned char)(v >> 020);
00070     p[3] = (unsigned char)(v >> 030);
00071 }
```

Here is the caller graph for this function:



7.11.1.20 write_le_u64()

```
void write_le_u64 (
    unsigned char * p,
    uint64_t v) [inline], [static]
```

Definition at line 73 of file [endian.c](#).

```
00073
00074     p[0] = (unsigned char)(v >> 000);
00075     p[1] = (unsigned char)(v >> 010);
00076     p[2] = (unsigned char)(v >> 020);
00077     p[3] = (unsigned char)(v >> 030);
00078     p[4] = (unsigned char)(v >> 040);
00079     p[5] = (unsigned char)(v >> 050);
00080     p[6] = (unsigned char)(v >> 060);
00081     p[7] = (unsigned char)(v >> 070);
00082 }
```

Here is the caller graph for this function:



7.12 endian.c

[Go to the documentation of this file.](#)

```

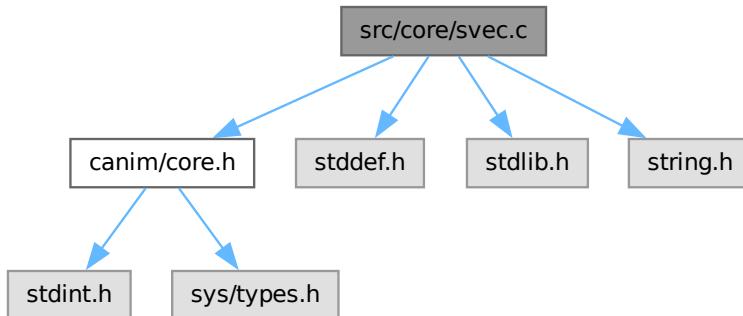
00001 // SPDX-License-Identifier: MIT
00002 #include "canim/core.h"
00003 #include <stdint.h>
00004 #include <string.h>
00005
00006 static inline uint16_t read_be_u16(const unsigned char *p) {
00007     return ((uint16_t)p[0] « 010) | ((uint16_t)p[1] « 000);
00008 }
00009
00010 static inline uint32_t read_be_u32(const unsigned char *p) {
00011     return ((uint32_t)p[0] « 030) | ((uint32_t)p[1] « 020) |
00012         ((uint32_t)p[2] « 010) | ((uint32_t)p[3] « 000);
00013 }
00014
00015 static inline uint64_t read_be_u64(const unsigned char *p) {
00016     return ((uint64_t)p[0] « 070) | ((uint64_t)p[1] « 060) |
00017         ((uint64_t)p[2] « 050) | ((uint64_t)p[3] « 040) |
00018         ((uint64_t)p[4] « 030) | ((uint64_t)p[5] « 020) |
00019         ((uint64_t)p[6] « 010) | ((uint64_t)p[7] « 000);
00020 }
00021
00022 static inline uint16_t read_le_u16(const unsigned char *p) {
00023     return ((uint16_t)p[1] « 010) | ((uint16_t)p[0] « 000);
00024 }
00025
00026 static inline uint32_t read_le_u32(const unsigned char *p) {
00027     return ((uint32_t)p[3] « 030) | ((uint32_t)p[2] « 020) |
00028         ((uint32_t)p[1] « 010) | ((uint32_t)p[0] « 000);
00029 }
00030
00031 static inline uint64_t read_le_u64(const unsigned char *p) {
00032     return ((uint64_t)p[7] « 070) | ((uint64_t)p[6] « 060) |
00033         ((uint64_t)p[5] « 050) | ((uint64_t)p[4] « 040) |
00034         ((uint64_t)p[3] « 030) | ((uint64_t)p[2] « 020) |
00035         ((uint64_t)p[1] « 010) | ((uint64_t)p[0] « 000);
00036 }
00037
00038 static inline void write_be_u16(unsigned char *p, uint16_t v) {
00039     p[0] = (unsigned char)(v » 010);
00040     p[1] = (unsigned char)(v » 000);
00041 }
00042
00043 static inline void write_be_u32(unsigned char *p, uint32_t v) {
00044     p[0] = (unsigned char)(v » 030);
00045     p[1] = (unsigned char)(v » 020);
00046     p[2] = (unsigned char)(v » 010);
00047     p[3] = (unsigned char)(v » 000);
00048 }
00049
00050 static inline void write_be_u64(unsigned char *p, uint64_t v) {
00051     p[0] = (unsigned char)(v » 070);
00052     p[1] = (unsigned char)(v » 060);
00053     p[2] = (unsigned char)(v » 050);
00054     p[3] = (unsigned char)(v » 040);
00055     p[4] = (unsigned char)(v » 030);
00056     p[5] = (unsigned char)(v » 020);
00057     p[6] = (unsigned char)(v » 010);
00058     p[7] = (unsigned char)(v » 000);
00059 }
00060
00061 static inline void write_le_u16(unsigned char *p, uint16_t v) {
00062     p[0] = (unsigned char)(v » 000);
00063     p[1] = (unsigned char)(v » 010);
00064 }
00065
00066 static inline void write_le_u32(unsigned char *p, uint32_t v) {
00067     p[0] = (unsigned char)(v » 000);
00068     p[1] = (unsigned char)(v » 010);
00069     p[2] = (unsigned char)(v » 020);
00070     p[3] = (unsigned char)(v » 030);
00071 }
00072
00073 static inline void write_le_u64(unsigned char *p, uint64_t v) {
00074     p[0] = (unsigned char)(v » 000);
00075     p[1] = (unsigned char)(v » 010);
00076     p[2] = (unsigned char)(v » 020);
00077     p[3] = (unsigned char)(v » 030);
00078     p[4] = (unsigned char)(v » 040);
00079     p[5] = (unsigned char)(v » 050);
00080     p[6] = (unsigned char)(v » 060);
00081     p[7] = (unsigned char)(v » 070);
00082 }
```

```
00083
00084 static inline float read_be_f32(const unsigned char *p) {
00085     uint32_t u = read_be_u32(p);
00086     float f;
00087     memcpy(&f, &u, sizeof(f));
00088     return f;
00089 }
00090
00091 static inline float read_le_f32(const unsigned char *p) {
00092     uint32_t u = read_le_u32(p);
00093     float f;
00094     memcpy(&f, &u, sizeof(f));
00095     return f;
00096 }
00097
00098 static inline void write_be_f32(unsigned char *p, float f) {
00099     uint32_t u;
00100     memcpy(&u, &f, sizeof(u));
00101     write_be_u32(p, u);
00102 }
00103
00104 static inline void write_le_f32(unsigned char *p, float f) {
00105     uint32_t u;
00106     memcpy(&u, &f, sizeof(u));
00107     write_le_u32(p, u);
00108 }
00109
00110 static inline double read_be_f64(const unsigned char *p) {
00111     uint64_t u = read_be_u64(p);
00112     double d;
00113     memcpy(&d, &u, sizeof(d));
00114     return d;
00115 }
00116
00117 static inline double read_le_f64(const unsigned char *p) {
00118     uint64_t u = read_le_u64(p);
00119     double d;
00120     memcpy(&d, &u, sizeof(d));
00121     return d;
00122 }
00123
00124 static inline void write_be_f64(unsigned char *p, double d) {
00125     uint64_t u;
00126     memcpy(&u, &d, sizeof(u));
00127     write_be_u64(p, u);
00128 }
00129
00130 static inline void write_le_f64(unsigned char *p, double d) {
00131     uint64_t u;
00132     memcpy(&u, &d, sizeof(u));
00133     write_le_u64(p, u);
00134 }
```

7.13 src/core/svec.c File Reference

```
#include "canim/core.h"
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for svec.c:



Functions

- static void `svec_init` (CanimResult *result, SVec *v, size_t elem_size)
- static void `svec_free` (CanimResult *result, SVec *v)
- static void `svec_push` (CanimResult *result, SVec *v, const void *elem)
- static void `svec_pop` (CanimResult *result, SVec *v, void *out)
- static void `svec_set` (CanimResult *result, SVec *v, size_t i, const void *elem)
- static void * `svec_get` (CanimResult *result, SVec *v, size_t i)

7.13.1 Function Documentation

7.13.1.1 `svec_free()`

```
void svec_free (
    CanimResult * result,
    SVec * v) [inline], [static]
```

Definition at line 18 of file [svec.c](#).

```
00018
00019     free(v->data);
00020     v->data = NULL;
00021     v->list_size = 0;
00022
00023     *result = SUCCESS;
00024 }
```

7.13.1.2 `svec_get()`

```
void * svec_get (
    CanimResult * result,
    SVec * v,
    size_t i) [inline], [static]
```

Definition at line 74 of file [svec.c](#).

```
00074
00075     *result = SUCCESS;
00076     return v->data + i * v->element_size;
00077 }
```

7.13.1.3 svec_init()

```
void svec_init (
    CanimResult * result,
    SVec * v,
    size_t elem_size) [inline], [static]
```

Definition at line 7 of file **svec.c**.

```
00007
00008     if (elem_size == 0) {
00009         *result = SVEC_ZERO_ELEMENT_SIZE;
00010         return;
00011     }
00012     v->element_size = elem_size;
00013     v->list_size = 0;
00014     v->data = NULL;
00015     *result = SUCCESS;
00016 }
```

7.13.1.4 svec_pop()

```
void svec_pop (
    CanimResult * result,
    SVec * v,
    void * out) [inline], [static]
```

Definition at line 48 of file **svec.c**.

```
00048
00049     size_t old_size = v->list_size;
00050     size_t new_size = old_size - 1;
00051     if (out) {
00052         memcpy(out, v->data + (old_size - 1) * v->element_size, v->element_size);
00053     }
00054     if (REALIGN(old_size, new_size)) {
00055         size_t new_cap = BIT_ALIGN(new_size);
00056         void *new_data = realloc(v->data, new_cap * v->element_size);
00057
00058         if (!new_data) {
00059             *result = SVEC_REALLOC_FAIL;
00060             return;
00061         }
00062         v->data = new_data;
00063     }
00064     v->list_size--;
00065     *result = SUCCESS;
00066 }
```

7.13.1.5 svec_push()

```
void svec_push (
    CanimResult * result,
    SVec * v,
    const void * elem) [inline], [static]
```

Definition at line 26 of file **svec.c**.

```
00026
00027
00028     size_t old_size = v->list_size;
00029     size_t new_size = old_size + 1;
00030
00031     if (REALIGN(old_size, new_size)) {
00032         size_t new_cap = BIT_ALIGN(new_size);
00033         void *new_data = realloc(v->data, new_cap * v->element_size);
00034
00035         if (!new_data) {
00036             *result = SVEC_REALLOC_FAIL;
00037             return;
```

```

00038      }
00039      v->data = new_data;
00040  }
00041
00042  memcpy(v->data + old_size * v->element_size, elem, v->element_size);
00043  v->list_size++;
00044
00045  *result = SUCCESS;
00046 }
```

7.13.1.6 svec_set()

```

void svec_set (
    CanimResult * result,
    SVec * v,
    size_t i,
    const void * elem) [inline], [static]
```

Definition at line 68 of file [svec.c](#).

```

00069                                     {
00070  memcpy(v->data + i * v->element_size, elem, v->element_size);
00071  *result = SUCCESS;
00072 }
```

7.14 svec.c

[Go to the documentation of this file.](#)

```

00001 // SPDX-License-Identifier: MIT
00002 #include "canim/core.h"
00003 #include <stddef.h>
00004 #include <stdlib.h>
00005 #include <string.h>
00006
00007 static inline void svec_init(CanimResult *result, SVec *v, size_t elem_size) {
00008     if (elem_size == 0) {
00009         *result = SVEC_ZERO_ELEMENT_SIZE;
00010         return;
00011     }
00012     v->element_size = elem_size;
00013     v->list_size = 0;
00014     v->data = NULL;
00015     *result = SUCCESS;
00016 }
00017
00018 static inline void svec_free(CanimResult *result, SVec *v) {
00019     free(v->data);
00020     v->data = NULL;
00021     v->list_size = 0;
00022
00023     *result = SUCCESS;
00024 }
00025
00026 static inline void svec_push(CanimResult *result, SVec *v, const void *elem) {
00027
00028     size_t old_size = v->list_size;
00029     size_t new_size = old_size + 1;
00030
00031     if (REALIGN(old_size, new_size)) {
00032         size_t new_cap = BIT_ALIGN(new_size);
00033         void *new_data = realloc(v->data, new_cap * v->element_size);
00034
00035         if (!new_data) {
00036             *result = SVEC_REALLOC_FAIL;
00037             return;
00038         }
00039         v->data = new_data;
00040     }
00041
00042     memcpy(v->data + old_size * v->element_size, elem, v->element_size);
00043     v->list_size++;
00044
00045     *result = SUCCESS;
00046 }
```

```

00047
00048 static inline void svec_pop(CanimResult *result, SVec *v, void *out) {
00049     size_t old_size = v->list_size;
00050     size_t new_size = old_size - 1;
00051     if (out) {
00052         memcpy(out, v->data + (old_size - 1) * v->element_size, v->element_size);
00053     }
00054     if (REALIGN(old_size, new_size)) {
00055         size_t new_cap = BIT_ALIGN(new_size);
00056         void *new_data = realloc(v->data, new_cap * v->element_size);
00057         if (!new_data) {
00058             *result = SVEC_REALLOC_FAIL;
00059             return;
00060         }
00061         v->data = new_data;
00062     }
00063     v->list_size--;
00064     *result = SUCCESS;
00065 }
00066
00067
00068 static inline void svec_set(CanimResult *result, SVec *v, size_t i,
00069                             const void *elem) {
00070     memcpy(v->data + i * v->element_size, elem, v->element_size);
00071     *result = SUCCESS;
00072 }
00073
00074 static inline void *svec_get(CanimResult *result, SVec *v, size_t i) {
00075     *result = SUCCESS;
00076     return v->data + i * v->element_size;
00077 }

```

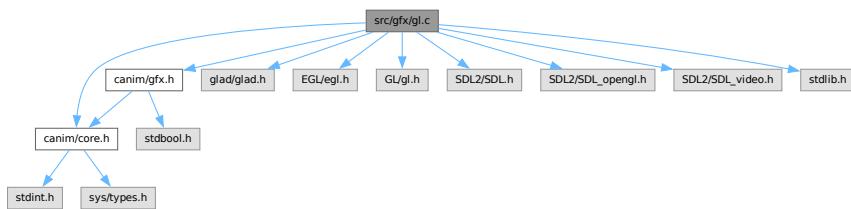
7.15 src/gfx/gl.c File Reference

```

#include "canim/core.h"
#include "canim/gfx.h"
#include "glad/glad.h"
#include <EGL/egl.h>
#include <GL/gl.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_opengl.h>
#include <SDL2/SDL_video.h>
#include <stdlib.h>

```

Include dependency graph for gl.c:



Data Structures

- struct [GfxDevice](#)

Functions

- [GfxDevice * gl_create_device \(CanimResult *result, GfxContainer *container, const GfxInitInfo *info\)](#)
- [void gl_destroy_device \(CanimResult *result, GfxContainer *container\)](#)

Variables

- const GfxAPI GFX_GL_API

7.15.1 Function Documentation

7.15.1.1 gl_create_device()

```
GfxDevice * gl_create_device (
    CanimResult * result,
    GfxContainer * container,
    const GfxInitInfo * info)
```

Definition at line 21 of file gl.c.

```
00022
00023     GfxDevice *dev = (GfxDevice *)calloc(1, sizeof(GfxDevice));
00024     if (!dev) {
00025         *result = GFX_DEVICE_CALLOC_ERROR;
00026         return NULL;
00027     }
00028     dev->headless = info->headless;
00029     dev->height = info->height;
00030     dev->width = info->width;
00031     if (dev->headless) {
00032         dev->egl_display = eglGetDisplay(EGL_DEFAULT_DISPLAY);
00033         if (dev->egl_display == EGL_NO_DISPLAY) {
00034             free(dev);
00035             *result = EGL_NO_DISPLAY_ERROR;
00036             return NULL;
00037         }
00038         if (!eglInitialize(dev->egl_display, 0, 0)) {
00039             free(dev);
00040             *result = EGL_DISPLAY_INIT_ERROR;
00041             return NULL;
00042         }
00043     EGLint cfg_attr[] = {EGL_SURFACE_TYPE, EGL_PBUFFER_BIT, EGL_RENDERABLE_TYPE,
00044                          EGL_OPENGL_BIT, EGL_NONE};
00045     EGLConfig cfg;
00046     EGLint N;
00047     if (!eglChooseConfig(dev->egl_display, cfg_attr, &cfg, 1, &N) || N == 0) {
00048         eglTerminate(dev->egl_display);
00049         free(dev);
00050         *result = EGL_DISPLAY_CONFIGURATION_ERROR;
00051         return NULL;
00052     }
00053     EGLint pb_attr[] = {EGL_WIDTH, info->width, EGL_HEIGHT, info->height,
00054                         EGL_NONE};
00055     dev->egl_surface = eglCreatePbufferSurface(dev->egl_display, cfg, pb_attr);
00056     if (dev->egl_surface == EGL_NO_SURFACE) {
00057         *result = EGL_NO_SURFACE_ERROR;
00058         eglTerminate(dev->egl_display);
00059         free(dev);
00060         return NULL;
00061     }
00062     eglBindAPI(EGL_OPENGL_API);
00063     dev->egl_context =
00064         eglCreateContext(dev->egl_display, cfg, EGL_NO_CONTEXT, NULL);
00065     if (dev->egl_context == EGL_NO_CONTEXT) {
00066         *result = EGL_NO_CONTEXT_ERROR;
00067         eglDestroySurface(dev->egl_display, dev->egl_surface);
00068         eglTerminate(dev->egl_display);
00069         free(dev);
00070         return NULL;
00071     }
00072     if (!eglMakeCurrent(dev->egl_display, dev->egl_surface, dev->egl_surface,
00073                         dev->egl_context)) {
00074         *result = EGL_MAKE_CURRENT_ERROR;
00075         eglDestroySurface(dev->egl_display, dev->egl_surface);
00076         eglDestroyContext(dev->egl_display, dev->egl_context);
00077         eglTerminate(dev->egl_display);
00078         free(dev);
00079         return NULL;
00080     }
00081 } else {
```

```

00083     if (SDL_Init(SDL_INIT_VIDEO) < 0) {
00084         *result = SDL_INIT_VIDEO_ERROR;
00085         free(dev);
00086         return NULL;
00087     }
00088     dev->win = info->native_window
00089             ? (SDL_Window *)info->native_window
00090             : SDL_CreateWindow("Canim", SDL_WINDOWPOS_CENTERED,
00091                                 SDL_WINDOWPOS_CENTERED, info->width,
00092                                 info->height,
00093                                 SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
00094     if (!dev->win) {
00095         *result = SDL_WINDOW_CREATION_ERROR;
00096         SDL_QuitSubSystem(SDL_INIT_VIDEO);
00097         free(dev);
00098         return NULL;
00099     }
00100     dev->glctx = SDL_GL_CreateContext(dev->win);
00101     if (!dev->glctx) {
00102         *result = SDL_GL_CONTEXT_CREATION_ERROR;
00103         SDL_DestroyWindow(dev->win);
00104         SDL_QuitSubSystem(SDL_INIT_VIDEO);
00105         free(dev);
00106         return NULL;
00107     }
00108     SDL_GL_MakeCurrent(dev->win, dev->glctx);
00109 }
00110 glViewport(0, 0, dev->width, dev->height);
00111 glClearColor(1.0f, 0.0f, 1.0f, 1.0f);
00112 glClear(GL_COLOR_BUFFER_BIT);
00113 if (dev->headless) {
00114     eglSwapBuffers(dev->egl_display, dev->egl_surface);
00115     gladLoadGLLoader((GLADloadproc)eglGetProcAddress);
00116 } else {
00117     SDL_GL_SwapWindow(dev->win);
00118     gladLoadGLLoader((GLADloadproc)SDL_GL_GetProcAddress);
00119 }
00120 }
00121 return dev;
00122 }
00123 }
```

7.15.1.2 gl_destroy_device()

```
void gl_destroy_device (
    CanimResult * result,
    GfxContainer * container)
```

Definition at line 124 of file [gl.c](#).

```

00124
00125     GfxDevice *device = container->impl;
00126     if (!device) {
00127         return;
00128     }
00129     if (device->headless) {
00130         eglDestroySurface(device->egl_display, device->egl_surface);
00131         eglDestroyContext(device->egl_display, device->egl_context);
00132         eglTerminate(device->egl_display);
00133     } else {
00134         if (device->glctx) {
00135             SDL_GL_DeleteContext(device->glctx);
00136         }
00137         if (device->win) {
00138             SDL_DestroyWindow(device->win);
00139         }
00140         SDL_QuitSubSystem(SDL_INIT_VIDEO);
00141     }
00142     free(device);
00143 };
```

7.15.2 Variable Documentation

7.15.2.1 GFX_GL_API

```
const GfxAPI GFX_GL_API
```

Initial value:

```
= {.gfx_create_device = gl_create_device,
     .gfx_destroy_device = gl_destroy_device}
```

Definition at line 145 of file [gl.c](#).

```
00145             {.gfx_create_device = gl_create_device,
00146              .gfx_destroy_device = gl_destroy_device};
```

7.16 gl.c

[Go to the documentation of this file.](#)

```
00001 // SPDX-License-Identifier: MIT
00002 #include "canim/core.h"
00003 #include "canim/gfx.h"
00004 #include "glad/glad.h"
00005 #include <EGL/egl.h>
00006 #include <GL/gl.h>
00007 #include <SDL2/SDL.h>
00008 #include <SDL2/SDL_opengl.h>
00009 #include <SDL2/SDL_video.h>
00010 #include <stdlib.h>
00011 struct GfxDevice {
00012     bool headless;
00013     int width;
00014     int height;
00015     SDL_GLContext glctx;
00016     SDL_Window *win;
00017     EGLDisplay egl_display;
00018     EGLSurface egl_surface;
00019     EGLContext egl_context;
00020 };
00021 GfxDevice *gl_create_device(CanimResult *result, GfxContainer *container,
00022                             const GfxInitInfo *info) {
00023     GfxDevice *dev = (GfxDevice *)calloc(1, sizeof(GfxDevice));
00024     if (!dev) {
00025         *result = GFX_DEVICE_CALLOC_ERROR;
00026         return NULL;
00027     }
00028     dev->headless = info->headless;
00029     dev->height = info->height;
00030     dev->width = info->width;
00031     if (dev->headless) {
00032         dev->egl_display = eglGetDisplay(EGL_DEFAULT_DISPLAY);
00033         if (dev->egl_display == EGL_NO_DISPLAY) {
00034             free(dev);
00035             *result = EGL_NO_DISPLAY_ERROR;
00036             return NULL;
00037         }
00038         if (!eglInitialize(dev->egl_display, 0, 0)) {
00039             free(dev);
00040             *result = EGL_DISPLAY_INIT_ERROR;
00041             return NULL;
00042         }
00043     }
00044     EGLint cfg_attr[] = {EGL_SURFACE_TYPE, EGL_PBUFFER_BIT, EGL_RENDERABLE_TYPE,
00045                          EGL_OPENGL_BIT, EGL_NONE};
00046     EGLConfig cfg;
00047     EGLint N;
00048     if (!eglChooseConfig(dev->egl_display, cfg_attr, &cfg, 1, &N) || N == 0) {
00049         eglTerminate(dev->egl_display);
00050         free(dev);
00051         *result = EGL_DISPLAY_CONFIGURATION_ERROR;
00052         return NULL;
00053     }
00054     EGLint pb_attr[] = {EGL_WIDTH, info->width, EGL_HEIGHT, info->height,
00055                         EGL_NONE};
00056     dev->egl_surface = eglCreatePbufferSurface(dev->egl_display, cfg, pb_attr);
00057     if (dev->egl_surface == EGL_NO_DISPLAY) {
00058         *result = EGL_NO_SURFACE_ERROR;
00059         eglTerminate(dev->egl_display);
00060         free(dev);
00061         return NULL;
00062     }
00063     eglBindAPI(EGL_OPENGL_API);
00064     dev->egl_context =
00065         eglCreateContext(dev->egl_display, cfg, EGL_NO_CONTEXT, NULL);
00066     if (dev->egl_context == EGL_NO_CONTEXT) {
00067         *result = EGL_NO_CONTEXT_ERROR;
00068         eglDestroySurface(dev->egl_display, dev->egl_surface);
```

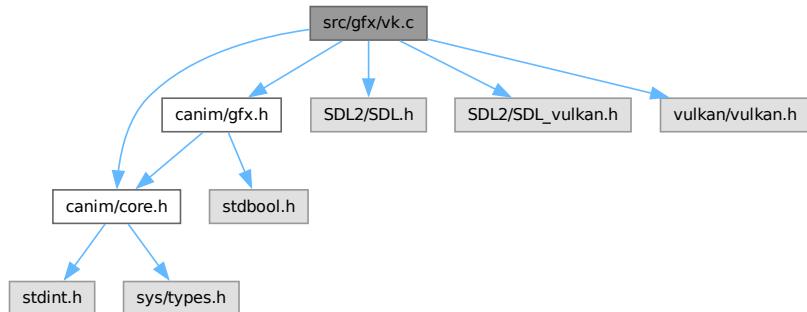
```

00069     eglTerminate(dev->egl_display);
00070     free(dev);
00071     return NULL;
00072 }
00073 if (!eglMakeCurrent(dev->egl_display, dev->egl_surface, dev->egl_surface,
00074                      dev->egl_context)) {
00075     *result = EGL_MAKE_CURRENT_ERROR;
00076     eglDestroySurface(dev->egl_display, dev->egl_surface);
00077     eglDestroyContext(dev->egl_display, dev->egl_context);
00078     eglTerminate(dev->egl_display);
00079     free(dev);
00080     return NULL;
00081 }
00082 } else {
00083     if (SDL_Init(SDL_INIT_VIDEO) < 0) {
00084         *result = SDL_INIT_VIDEO_ERROR;
00085         free(dev);
00086         return NULL;
00087     }
00088     dev->win = info->native_window
00089     ? (SDL_Window *)info->native_window
00090     : SDL_CreateWindow("Canim", SDL_WINDOWPOS_CENTERED,
00091                         SDL_WINDOWPOS_CENTERED, info->width,
00092                         info->height,
00093                         SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
00094     if (!dev->win) {
00095         *result = SDL_WINDOW_CREATION_ERROR;
00096         SDL_QuitSubSystem(SDL_INIT_VIDEO);
00097         free(dev);
00098         return NULL;
00099     }
00100    dev->glctx = SDL_GL_CreateContext(dev->win);
00101    if (!dev->glctx) {
00102        *result = SDL_GL_CONTEXT_CREATION_ERROR;
00103        SDL_DestroyWindow(dev->win);
00104        SDL_QuitSubSystem(SDL_INIT_VIDEO);
00105        free(dev);
00106        return NULL;
00107    }
00108    SDL_GL_MakeCurrent(dev->win, dev->glctx);
00109 }
00110 glViewport(0, 0, dev->width, dev->height);
00111 glClearColor(1.0f, 0.0f, 1.0f, 1.0f);
00112 glClear(GL_COLOR_BUFFER_BIT);
00113 if (dev->headless) {
00114     eglSwapBuffers(dev->egl_display, dev->egl_surface);
00115     gladLoadGLLoader((GLADloadproc)eglGetProcAddress);
00116 } else {
00117     SDL_GL_SwapWindow(dev->win);
00118     gladLoadGLLoader((GLADloadproc)SDL_GL_GetProcAddress);
00119 }
00120 }
00121
00122 return dev;
00123 }
00124 void gl_destroy_device(CanimResult *result, GfxContainer *container) {
00125     GfxDevice *device = container->impl;
00126     if (!device) {
00127         return;
00128     }
00129     if (device->headless) {
00130         eglDestroySurface(device->egl_display, device->egl_surface);
00131         eglDestroyContext(device->egl_display, device->egl_context);
00132         eglTerminate(device->egl_display);
00133     } else {
00134         if (device->glctx) {
00135             SDL_GL_DeleteContext(device->glctx);
00136         }
00137         if (device->win) {
00138             SDL_DestroyWindow(device->win);
00139         }
00140         SDL_QuitSubSystem(SDL_INIT_VIDEO);
00141     }
00142     free(device);
00143 };
00144
00145 const GfxAPI GFX_GL_API = {.gfx_create_device = gl_create_device,
00146                             .gfx_destroy_device = gl_destroy_device};
00147
00148 __attribute__((visibility("default"))) const GfxAPI *GFX_API_ENTRY =
00149     &GFX_GL_API;

```

7.17 src/gfx/vk.c File Reference

```
#include "canim/core.h"
#include "canim/gfx.h"
#include <SDL2/SDL.h>
#include <SDL2/SDL_vulkan.h>
#include <vulkan/vulkan.h>
Include dependency graph for vk.c:
```



Data Structures

- struct [GfxDevice](#)

Functions

- [GfxDevice * vk_create_device \(CanimResult *result, GfxContainer *container, const GfxInitInfo *info\)](#)

7.17.1 Function Documentation

7.17.1.1 [vk_create_device\(\)](#)

```
GfxDevice * vk_create_device (
    CanimResult * result,
    GfxContainer * container,
    const GfxInitInfo * info)
```

Todo FIX

Todo FIX

Todo FIX

Definition at line 15 of file [vk.c](#).

```

00016
00017     GfxDevice *dev = calloc(1, sizeof(GfxDevice));
00018     if (!dev) {
00019         *result = GFX_DEVICE_CALLOC_ERROR;
00020         return NULL;
00021     }
00022
00023     dev->headless = info->headless;
00024     dev->width = info->width;
00025     dev->height = info->height;
00026     dev->win = NULL;
00027     if (!info->headless) {
00028         dev->win = SDL_CreateWindow(
00029             "Canim Vulkan", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
00030             info->width, info->height, SDL_WINDOW_VULKAN | SDL_WINDOW_SHOWN);
00031
00032         if (!dev->win) {
00033             free(dev);
00034             *result = SDL_WINDOW_CREATION_ERROR;
00035             return NULL;
00036         }
00037     }
00038     uint32_t extCount = 0;
00039     const char **extNames = NULL;
00040
00041     if (!info->headless) {
00042         if (!SDL_Vulkan_GetInstanceExtensions(dev->win, &extCount, NULL)) {
00043             SDL_DestroyWindow(dev->win);
00044             free(dev);
00045             *result = 0x03ffff; /// @todo FIX
00046             return NULL;
00047         }
00048
00049         extNames = malloc(sizeof(char *) * extCount);
00050         if (!extNames) {
00051             SDL_DestroyWindow(dev->win);
00052             free(dev);
00053             *result = 0x03ffff; /// @todo FIX
00054
00055             return NULL;
00056         }
00057
00058         SDL_Vulkan_GetInstanceExtensions(dev->win, &extCount, extNames);
00059     }
00060
00061     VkApplicationInfo app = {.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO,
00062                             .pApplicationName = "Canim",
00063                             .applicationVersion = VK_MAKE_VERSION(1, 0, 0),
00064                             .pEngineName = "Canim",
00065                             .engineVersion = VK_MAKE_VERSION(1, 0, 0),
00066                             .apiVersion = VK_API_VERSION_1_3};
00067
00068     VkInstanceCreateInfo ci = {.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO,
00069                             .pApplicationInfo = &app,
00070                             .enabledLayerCount = 0,
00071                             .ppEnabledLayerNames = NULL,
00072                             .enabledExtensionCount = extCount,
00073                             .ppEnabledExtensionNames = extNames};
00074
00075     VkResult vkres = vkCreateInstance(&ci, NULL, &dev->inst);
00076
00077     free(extNames);
00078     if (vkres != VK_SUCCESS) {
00079         if (dev->win)
00080             SDL_DestroyWindow(dev->win);
00081         free(dev);
00082         *result = 0x03ffff; /// @todo FIX
00083
00084         return NULL;
00085     }
00086
00087     *result = SUCCESS;
00088     return dev;
00089 }
```

7.18 vk.c

[Go to the documentation of this file.](#)

00001 // SPDX-License-Identifier: MIT

```

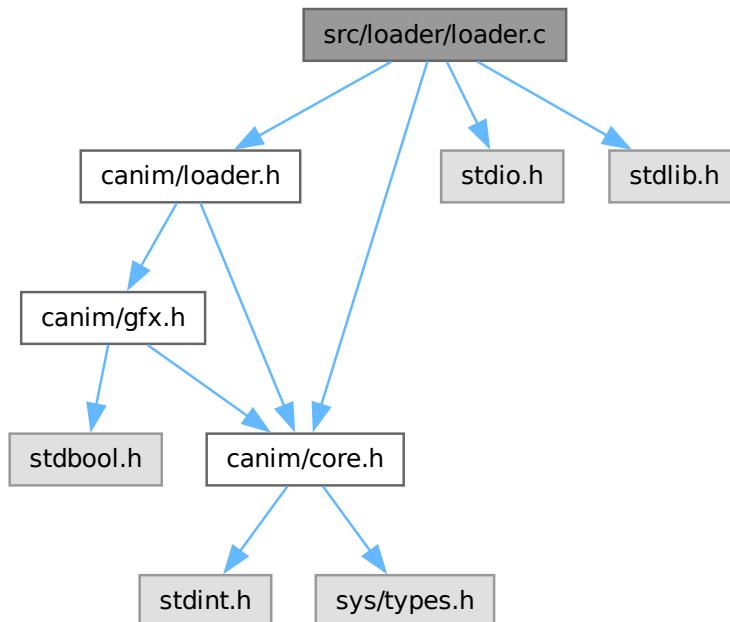
00002 #include "canim/core.h"
00003 #include "canim/gfx.h"
00004 #include <SDL2/SDL.h>
00005 #include <SDL2/SDL_vulkan.h>
00006 #include <vulkan/vulkan.h>
00007 struct GfxDevice {
00008     bool headless;
00009     int width;
0010     int height;
0011     VkInstance inst;
0012     SDL_Window *win;
0013 };
0014
0015 GfxDevice *vk_create_device(CanimResult *result, GfxContainer *container,
0016                             const GfxInitInfo *info) {
0017     GfxDevice *dev = calloc(1, sizeof(GfxDevice));
0018     if (!dev) {
0019         *result = GFX_DEVICE_CALLOC_ERROR;
0020         return NULL;
0021     }
0022
0023     dev->headless = info->headless;
0024     dev->width = info->width;
0025     dev->height = info->height;
0026     dev->win = NULL;
0027     if (!info->headless) {
0028         dev->win = SDL_CreateWindow(
0029             "Canim Vulkan", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
0030             info->width, info->height, SDL_WINDOW_VULKAN | SDL_WINDOW_SHOWN);
0031
0032         if (!dev->win) {
0033             free(dev);
0034             *result = SDL_WINDOW_CREATION_ERROR;
0035             return NULL;
0036         }
0037     }
0038     uint32_t extCount = 0;
0039     const char **extNames = NULL;
0040
0041     if (!info->headless) {
0042         if (!SDL_Vulkan_GetInstanceExtensions(dev->win, &extCount, NULL)) {
0043             SDL_DestroyWindow(dev->win);
0044             free(dev);
0045             *result = 0x03ffff; // @todo FIX
0046             return NULL;
0047         }
0048
0049         extNames = malloc(sizeof(char *) * extCount);
0050         if (!extNames) {
0051             SDL_DestroyWindow(dev->win);
0052             free(dev);
0053             *result = 0x03ffff; // @todo FIX
0054
0055             return NULL;
0056         }
0057
0058         SDL_Vulkan_GetInstanceExtensions(dev->win, &extCount, extNames);
0059     }
0060
0061     VkApplicationInfo app = {.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO,
0062                             .pApplicationName = "Canim",
0063                             .applicationVersion = VK_MAKE_VERSION(1, 0, 0),
0064                             .pEngineName = "Canim",
0065                             .engineVersion = VK_MAKE_VERSION(1, 0, 0),
0066                             .apiVersion = VK_API_VERSION_1_3};
0067
0068     VkInstanceCreateInfo ci = {.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO,
0069                             .pApplicationInfo = &app,
0070                             .enabledLayerCount = 0,
0071                             .ppEnabledLayerNames = NULL,
0072                             .enabledExtensionCount = extCount,
0073                             .ppEnabledExtensionNames = extNames};
0074
0075     VkResult vkres = vkCreateInstance(&ci, NULL, &dev->inst);
0076
0077     free(extNames);
0078     if (vkres != VK_SUCCESS) {
0079         if (dev->win)
0080             SDL_DestroyWindow(dev->win);
0081         free(dev);
0082         *result = 0x03ffff; // @todo FIX
0083
0084         return NULL;
0085     }
0086
0087     *result = SUCCESS;
0088     return dev;

```

```
00089 }
```

7.19 src/loader/loader.c File Reference

```
#include "canim/loader.h"
#include "canim/core.h"
#include <stdio.h>
#include <stdlib.h>
Include dependency graph for loader.c:
```



Functions

- `const char * gfx_backend_libname (GfxBackend backend)`
- `GfxContainer * gfx_load_backend (CanimResult *result, GfxBackend backend, const GfxInitInfo *info)`
This loads a backend and makes a graphics container.
- `void gfx_unload_backend (CanimResult *result, GfxContainer *gfx)`
This unloads a backend.

7.19.1 Function Documentation

7.19.1.1 gfx_backend_libname()

```
const char * gfx_backend_libname (
    GfxBackend backend)
```

Definition at line 27 of file [loader.c](#).

```
00027
00028     switch (backend) {
00029         case CANIM_GFX_GL:
00030             return "libgl" LIB_EXT;
00031     default:
00032         return NULL;
00033     }
00034 }
```

Here is the caller graph for this function:



7.19.1.2 gfx_load_backend()

```
GfxContainer * gfx_load_backend (
    CanimResult * result,
    GfxBackend backend,
    const GfxInitInfo * info)
```

This loads a backend and makes a graphics container.

Parameters

<i>result</i>	This is the result of the output
<i>backend</i>	This is the gfx api
<i>info</i>	The relevant info to create the container

Returns

A gfx container

Definition at line 36 of file [loader.c](#).

```
00037
00038     const char *libname = gfx_backend_libname(backend);
00039     LIB_HANDLE handle = LIB_LOAD(libname);
00040     const GfxAPI *const *entry =
00041         (const GfxAPI *const *)LIB_SYM(handle, "GFX_API_ENTRY");
00042     GfxContainer *gfx = calloc(1, sizeof(*gfx));
00043     const GfxAPI *api = *entry;
00044     gfx->api = *api;
00045     gfx->impl = NULL;
00046     gfx->backend = backend;
00047     GfxDevice *dev = gfx->api.gfx_create_device(result, gfx, info);
00048     if (IS_AN_ERROR(*result)) {
00049         return NULL;
00050     }
00051     gfx->handle = (void *)handle;
00052     gfx->impl = dev;
00053     *result = SUCCESS;
```

```
00054     return gfx;
00055 }
```

Here is the call graph for this function:



7.19.1.3 gfx_unload_backend()

```
void gfx_unload_backend (
    CanimResult * result,
    GfxContainer * gfx)
```

This unloads a backend.

Parameters

<i>result</i>	This is the result of the output
<i>gfx</i>	This is the container

Definition at line 57 of file [loader.c](#).

```
00057
00058     LIB_CLOSE((LIB_HANDLE)gfx->handle);
00059     *result = SUCCESS;
00060 }
```

7.20 loader.c

[Go to the documentation of this file.](#)

```
00001 // SPDX-License-Identifier: MIT
00002 #include "canim/loader.h"
00003 #include "canim/core.h"
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #ifdef CANIM_PLATFORM_POSIX
00007 #include <dlopen.h>
00008 #define LIB_HANDLE void *
00009 #define LIB_LOAD(name) dlopen(name, RTLD_NOW | RTLD_LOCAL)
00010 #define LIB_SYM(h, sym) dlsym(h, sym)
00011 #define LIB_CLOSE(h) dlclose(h)
00012 #endif
00013 #ifdef CANIM_PLATFORM_LINUX
00014 #define LIB_EXT ".so"
00015 #endif
00016 #ifdef CANIM_PLATFORM_MAC
00017 #define LIB_EXT ".dylib"
00018 #endif
00019 #ifdef CANIM_PLATFORM_WINDOWS
00020 #include <windows.h>
00021 #define LIB_HANDLE HMODULE
00022 #define LIB_LOAD(name) LoadLibraryA(name)
```

```
00023 #define LIB_SYM(h, sym) GetProcAddress(h, sym)
00024 #define LIB_CLOSE(h) FreeLibrary(h)
00025 #define LIB_EXT ".dll"
00026 #endif
00027 const char *gfx_backend_libname(GfxBackend backend) {
00028     switch (backend) {
00029     case CANIM_GFX_GL:
00030         return "libgl" LIB_EXT;
00031     default:
00032         return NULL;
00033     }
00034 }
00035 }
00036 GfxContainer *gfx_load_backend(CanimResult *result, GfxBackend backend,
00037                                     const GfxInitInfo *info) {
00038     const char *libname = gfx_backend_libname(backend);
00039     LIB_HANDLE handle = LIB_LOAD(libname);
00040     const GfxAPI *const *entry =
00041         (const GfxAPI *const *)LIB_SYM(handle, "GFX_API_ENTRY");
00042     GfxContainer *gfx = calloc(1, sizeof(*gfx));
00043     const GfxAPI *api = *entry;
00044     gfx->api = *api;
00045     gfx->impl = NULL;
00046     gfx->backend = backend;
00047     GfxDevice *dev = gfx->api.gfx_create_device(result, gfx, info);
00048     if (IS_AN_ERROR(*result)) {
00049         return NULL;
00050     }
00051     gfx->handle = (void *)handle;
00052     gfx->impl = dev;
00053     *result = SUCCESS;
00054     return gfx;
00055 }
00056
00057 void gfx_unload_backend(CanimResult *result, GfxContainer *gfx) {
00058     LIB_CLOSE((LIB_HANDLE)gfx->handle);
00059     *result = SUCCESS;
00060 }
```

Index

api
 GfxContainer, 15

backend
 GfxContainer, 15

BIT_ALIGN
 core.h, 25

BIT_IGNORE
 core.h, 25

BIT_SIZE
 core.h, 25

BUFFER_SIZE
 core.h, 26

CANIM_API
 core.h, 26

CANIM_GFX_GL
 gfx.h, 45

CANIM_GFX_NONE
 gfx.h, 45

CANIM_PLATFORM_UNKNOWN
 core.h, 26

CanimResult
 core.h, 29

core.c
 print_error, 51

core.h
 BIT_ALIGN, 25
 BIT_IGNORE, 25
 BIT_SIZE, 25
 BUFFER_SIZE, 26
 CANIM_API, 26
 CANIM_PLATFORM_UNKNOWN, 26
 CanimResult, 29
 EGL_DISPLAY_CONFIGURATION_ERROR, 30
 EGL_DISPLAY_INIT_ERROR, 30
 EGL_GLAD_LOAD_ERROR, 30
 EGL_MAKE_CURRENT_ERROR, 30
 EGL_NO_CONTEXT_ERROR, 30
 EGL_NO_DISPLAY_ERROR, 30
 EGL_NO_SURFACE_ERROR, 30
 EPSILON, 26
 FATAL, 26
 GFX_DEVICE_CALLOC_ERROR, 29
 IS_AN_ERROR, 26
 max, 27
 NONFATAL, 27
 NOOP, 27
 null, 27
 PCLOSE, 27

 POPEN, 28
 print_error, 30
 read_be_f32, 31
 read_be_f64, 32
 read_be_u16, 32
 read_be_u32, 32
 read_be_u64, 33
 read_le_f32, 33
 read_le_f64, 33
 read_le_u16, 34
 read_le_u32, 34
 read_le_u64, 34
 REALIGN, 28
 SDL_GL_CONTEXT_CREATION_ERROR, 30
 SDL_GLAD_LOAD_ERROR, 30
 SDL_INIT_VIDEO_ERROR, 30
 SDL_WINDOW_CREATION_ERROR, 30
 STATUS_TYPE, 28
 STATUS_TYPE_MASK, 28
 STATUS_TYPE_SHIFT, 28
 SUCCESS, 29
 svec_free, 35
 svec_get, 35
 svec_init, 35
 svec_pop, 36
 svec_push, 36
 SVEC_REALLOC_FAIL, 30
 svec_set, 36
 SVEC_ZERO_ELEMENT_SIZE, 30
 WB, 29
 write_be_f32, 37
 write_be_f64, 37
 write_be_u16, 37
 write_be_u32, 37
 write_be_u64, 38
 write_le_f32, 38
 write_le_f64, 38
 write_le_u16, 39
 write_le_u32, 39
 write_le_u64, 39

data
 SVec, 19

egl_context
 GfxDevice, 16

egl_display
 GfxDevice, 16

 EGL_DISPLAY_CONFIGURATION_ERROR
 core.h, 30

EGL_DISPLAY_INIT_ERROR
 core.h, 30

EGL_GLAD_LOAD_ERROR
 core.h, 30

EGL_MAKE_CURRENT_ERROR
 core.h, 30

EGL_NO_CONTEXT_ERROR
 core.h, 30

EGL_NO_DISPLAY_ERROR
 core.h, 30

EGL_NO_SURFACE_ERROR
 core.h, 30

egl_surface
 GfxDevice, 16

element_size
 SVec, 19

Endian.c
 read_be_f32, 54
 read_be_f64, 54
 read_be_u16, 54
 read_be_u32, 55
 read_be_u64, 55
 read_le_f32, 55
 read_le_f64, 56
 read_le_u16, 56
 read_le_u32, 56
 read_le_u64, 57
 write_be_f32, 57
 write_be_f64, 58
 write_be_u16, 58
 write_be_u32, 58
 write_be_u64, 59
 write_le_f32, 59
 write_le_f64, 60
 write_le_u16, 60
 write_le_u32, 60
 write_le_u64, 61

EPSILON
 core.h, 26

FATAL
 core.h, 26

gfx.h
 CANIM_GFX_GL, 45
 CANIM_GFX_NONE, 45
 GfxAPI, 45
 GfxBackend, 45
 GfxContainer, 45
 GfxDevice, 45
 gfx_backend_libname
 loader.c, 75

gfx_create_device
 GfxAPI, 14

gfx_destroy_device
 GfxAPI, 14

GFX_DEVICE_CALLOC_ERROR
 core.h, 29

GFX_GL_API

gl.c, 69
 gfx_load_backend
 loader.c, 76
 loader.h, 48
 gfx_unload_backend
 loader.c, 77
 loader.h, 48

GfxAPI, 13
 gfx.h, 45
 gfx_create_device, 14
 gfx_destroy_device, 14

GfxBackend
 gfx.h, 45

GfxContainer, 14
 api, 15
 backend, 15
 gfx.h, 45
 handle, 15
 impl, 15

GfxDevice, 16
 egl_context, 16
 egl_display, 16
 egl_surface, 16
 gfx.h, 45
 glctx, 16
 headless, 16
 height, 17
 inst, 17
 width, 17
 win, 17

GfxInitInfo, 17
 headless, 18
 height, 18
 native_window, 18
 width, 18

gl.c
 GFX_GL_API, 69
 gl_create_device, 68
 gl_destroy_device, 69

gl_create_device
 gl.c, 68

gl_destroy_device
 gl.c, 69

glctx
 GfxDevice, 16

handle
 GfxContainer, 15

headless
 GfxDevice, 16
 GfxInitInfo, 18

height
 GfxDevice, 17
 GfxInitInfo, 18

impl
 GfxContainer, 15

include Directory Reference, 11

include/canim Directory Reference, 9

include/canim/canim.h, 21, 22
include/canim/core.h, 22, 40
include/canim/gfx.h, 43, 46
include/canim/loader.h, 46, 50
inst
 GfxDevice, 17
IS_AN_ERROR
 core.h, 26

list_size
 SVec, 19
loader.c
 gfx_backend_libname, 75
 gfx_load_backend, 76
 gfx_unload_backend, 77
loader.h
 gfx_load_backend, 48
 gfx_unload_backend, 48

max
 core.h, 27

native_window
 GfxInitInfo, 18
NONFATAL
 core.h, 27
NOOP
 core.h, 27
null
 core.h, 27

PCLOSE
 core.h, 27
POOPEN
 core.h, 28
print_error
 core.c, 51
 core.h, 30

read_be_f32
 core.h, 31
 endian.c, 54
read_be_f64
 core.h, 32
 endian.c, 54
read_be_u16
 core.h, 32
 endian.c, 54
read_be_u32
 core.h, 32
 endian.c, 55
read_be_u64
 core.h, 33
 endian.c, 55
read_le_f32
 core.h, 33
 endian.c, 55
read_le_f64
 core.h, 33
 endian.c, 56
 endian.c, 56
read_le_u16
 core.h, 34
 endian.c, 56
read_le_u32
 core.h, 34
 endian.c, 56
read_le_u64
 core.h, 34
 endian.c, 57
REALIGN
 core.h, 28

SDL_GL_CONTEXT_CREATION_ERROR
 core.h, 30
SDL_GLAD_LOAD_ERROR
 core.h, 30
SDL_INIT_VIDEO_ERROR
 core.h, 30
SDL_WINDOW_CREATION_ERROR
 core.h, 30
src Directory Reference, 11
src/core Directory Reference, 10
src/core/core.c, 50, 52
src/core/endian.c, 53, 62
src/core/svec.c, 63, 66
src/gfx Directory Reference, 10
src/gfx/gl.c, 67, 70
src/gfx/vk.c, 72, 73
src/loader Directory Reference, 11
src/loader/loader.c, 75, 77
STATUS_TYPE
 core.h, 28
STATUS_TYPE_MASK
 core.h, 28
STATUS_TYPE_SHIFT
 core.h, 28
SUCCESS
 core.h, 29
SVec, 19
 data, 19
 element_size, 19
 list_size, 19
svec.c
 svec_free, 64
 svec_get, 64
 svec_init, 64
 svec_pop, 65
 svec_push, 65
 svec_set, 66
svec_free
 core.h, 35
 svec.c, 64
svec_get
 core.h, 35
 svec.c, 64
svec_init
 core.h, 35
 svec.c, 64

svec_pop
 core.h, 36
 svec.c, 65
svec_push
 core.h, 36
 svec.c, 65
SVEC_REALLOC_FAIL
 core.h, 30
svec_set
 core.h, 36
 svec.c, 66
SVEC_ZERO_ELEMENT_SIZE
 core.h, 30

Todo List, 1

vk.c
 vk_create_device, 72
vk_create_device
 vk.c, 72

WB
 core.h, 29
width
 GfxDevice, 17
 GfxInitInfo, 18
win
 GfxDevice, 17
write_be_f32
 core.h, 37
 endian.c, 57
write_be_f64
 core.h, 37
 endian.c, 58
write_be_u16
 core.h, 37
 endian.c, 58
write_be_u32
 core.h, 37
 endian.c, 58
write_be_u64
 core.h, 38
 endian.c, 59
write_le_f32
 core.h, 38
 endian.c, 59
write_le_f64
 core.h, 38
 endian.c, 60
write_le_u16
 core.h, 39
 endian.c, 60
write_le_u32
 core.h, 39
 endian.c, 60
write_le_u64
 core.h, 39
 endian.c, 61