

2P11

Generated by Doxygen 1.9.1

<b>1 PII 2</b>	<b>1</b>
1.1 Doxygen . . . . .	1
<b>2 resources</b>	<b>2</b>
<b>3 Data Structure Index</b>	<b>2</b>
3.1 Data Structures . . . . .	2
<b>4 File Index</b>	<b>3</b>
4.1 File List . . . . .	3
<b>5 Data Structure Documentation</b>	<b>4</b>
5.1 Background Struct Reference . . . . .	4
5.1.1 Detailed Description . . . . .	4
5.2 Button Struct Reference . . . . .	4
5.2.1 Detailed Description . . . . .	5
5.3 Camera Struct Reference . . . . .	5
5.3.1 Detailed Description . . . . .	5
5.3.2 Field Documentation . . . . .	5
5.4 Clickable Struct Reference . . . . .	6
5.4.1 Detailed Description . . . . .	6
5.5 ComponentWrapper Struct Reference . . . . .	6
5.5.1 Detailed Description . . . . .	7
5.6 Entity Struct Reference . . . . .	7
5.6.1 Detailed Description . . . . .	7
5.7 HashMap Struct Reference . . . . .	7
5.7.1 Detailed Description . . . . .	8
5.8 HashMapEntry Struct Reference . . . . .	8
5.8.1 Detailed Description . . . . .	8
5.9 Hoverable Struct Reference . . . . .	8
5.9.1 Detailed Description . . . . .	9
5.10 Inputs Struct Reference . . . . .	9
5.10.1 Detailed Description . . . . .	9
5.11 LinkedList Struct Reference . . . . .	9
5.11.1 Detailed Description . . . . .	10
5.12 LinkedListLink Struct Reference . . . . .	10
5.12.1 Detailed Description . . . . .	10
5.12.2 Field Documentation . . . . .	10
5.13 Minimap Struct Reference . . . . .	11
5.13.1 Detailed Description . . . . .	11
5.14 Position Struct Reference . . . . .	11
5.14.1 Detailed Description . . . . .	12
5.15 Rc Struct Reference . . . . .	12
5.16 Sprite Struct Reference . . . . .	12

5.17 World Struct Reference . . . . .	12
5.17.1 Detailed Description . . . . .	13
5.17.2 Member Function Documentation . . . . .	13
5.17.3 Field Documentation . . . . .	13
<b>6 File Documentation</b>	<b>14</b>
6.1 asset_manager.h File Reference . . . . .	14
6.1.1 Function Documentation . . . . .	15
6.2 bitflag.h File Reference . . . . .	15
6.3 camera.h File Reference . . . . .	16
6.3.1 Function Documentation . . . . .	18
6.4 ecs.h File Reference . . . . .	18
6.4.1 Macro Definition Documentation . . . . .	20
6.4.2 Typedef Documentation . . . . .	21
6.4.3 Function Documentation . . . . .	21
6.5 hash_map.h File Reference . . . . .	22
6.5.1 Function Documentation . . . . .	24
6.6 input.h File Reference . . . . .	25
6.6.1 Macro Definition Documentation . . . . .	26
6.6.2 Typedef Documentation . . . . .	27
6.6.3 Function Documentation . . . . .	27
6.7 linked_list.h File Reference . . . . .	28
6.7.1 Function Documentation . . . . .	29
6.8 ui.h File Reference . . . . .	29
6.8.1 Function Documentation . . . . .	31
6.9 util.h File Reference . . . . .	31
6.9.1 Macro Definition Documentation . . . . .	33
6.10 vec.h File Reference . . . . .	34
6.10.1 Detailed Description . . . . .	35
6.10.2 Macro Definition Documentation . . . . .	35
6.10.3 Function Documentation . . . . .	36
<b>Index</b>	<b>37</b>

## 1 2Pll 2

### 1.1 Doxygen

The use of `make doc` requires doxygen, LaTeX and graphviz.

The use of the command `make htmldoc` require firefox

## 2 resources

Here is a list of documents used during the making of this project

Websites and pages :

- <https://web.archive.org/web/19990903133921/http://www.concentric.net/~Ttwang/tech/primehash.htm>
- <https://courses.csail.mit.edu/6.006/spring11/rec/rec07.pdf>
- <https://wiki.libsdl.org/SDL2>
- <https://en.cppreference.com>
- <https://ianjk.com/ecs-in-rust/>
- [https://austinmorlan.com/posts/entity\\_component\\_system/](https://austinmorlan.com/posts/entity_component_system/)
- <https://www.david-colson.com/2020/02/09/making-a-simple-ecs.html>
- [https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo\\_hash\\_function](https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function)
- <https://www.libsdl.org/release/SDL-1.2.15/docs/html/>
- <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf>
- <https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#work-sharing-directives>
- <http://www.gameaipro.com/>
- <https://valgrind.org/docs/manual/index.html>

Books and articles :

- Game AI Pro 360: Guide to Movement and Pathfinding - Steve Rabin - 2019
- Steering Behaviors For Autonomous Characters - Craig W. Reynolds - 1999
- Game Engine Architecture, 3rd edition - Jason Gregory - 2018

## 3 Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<b>Background</b>	
Entities with this component are the background of the user interface	4
<b>Button</b>	4
<b>Camera</b>	5

<a href="#">Clickable</a>		
Entities with this component start an action when clicked on		6
<a href="#">ComponentWrapper</a>		
Used to store the component, its type and its id		6
<a href="#">Entity</a>		
The entity structure for the ECS		7
<a href="#">HashMap</a>		
A hash map		7
<a href="#">HashMapEntry</a>		
An entry in a <a href="#">HashMap</a> , i.e. a key-value pair		8
<a href="#">Hoverable</a>		
Entities with this component show text when hovered		8
<a href="#">Inputs</a>		
Stores keys and mouse buttons		9
<a href="#">LinkedList</a>		
A singly linked list		9
<a href="#">LinkedListLink</a>		
A link of <a href="#">LinkedList</a>		10
<a href="#">Minimap</a>		
Component that corresponds to the minimap		11
<a href="#">Position</a>		
A component that contains the world space coordinates of an entity		11
<a href="#">Rc</a>		12
<a href="#">Sprite</a>		12
<a href="#">World</a>		
The world structure used to store the different parts of the ECS		12

## 4 File Index

### 4.1 File List

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

<a href="#">asset_manager.h</a>	14
<a href="#">bitflag.h</a>	15
<a href="#">camera.h</a>	16
<a href="#">ecs.h</a>	18
<a href="#">hash_map.h</a>	22
<a href="#">input.h</a>	25

<a href="#">linked_list.h</a>	28
<a href="#">ui.h</a>	29
<a href="#">util.h</a>	31
<a href="#">vec.h</a>	34

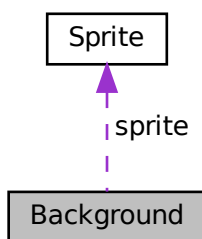
## 5 Data Structure Documentation

### 5.1 Background Struct Reference

Entities with this component are the background of the user interface.

```
#include <ui.h>
```

Collaboration diagram for Background:



#### Data Fields

- [Sprite](#) \* **sprite**
- `SDL_Rect` \* **rect**

#### 5.1.1 Detailed Description

Entities with this component are the background of the user interface.

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

- [ui.h](#)

### 5.2 Button Struct Reference

```
#include <ui.h>
```

### Data Fields

- `SDL_Rect * rect`
- `char * text`
- `Uint8 is_clicked`

#### 5.2.1 Detailed Description

These entities are similar to the clickables but they also a text instead of a sprite

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

- [ui.h](#)

## 5.3 Camera Struct Reference

```
#include <camera.h>
```

### Data Fields

- `float x`
- `float y`
- `float zoom`

#### 5.3.1 Detailed Description

The [Camera](#) struct is not a component, it is meant to have exactly one instance and serves as the base for screenspace<->worldspace calculations

#### 5.3.2 Field Documentation

##### 5.3.2.1 zoom `float Camera::zoom`

`zoom` is such that if `zoom==1`, one pixel in screenspace is one pixel in worldspace, while if `zoom==2`, one pixel in screenspace is two pixels in worldspace

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

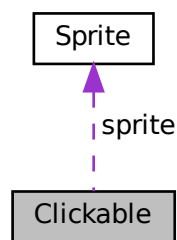
- [camera.h](#)

## 5.4 Clickable Struct Reference

Entities with this component start an action when clicked on.

```
#include <ui.h>
```

Collaboration diagram for Clickable:



### Data Fields

- [Sprite](#) \* **sprite**
- `SDL_Rect` \* **rect**
- `Uint8` **is\_clicked**

#### 5.4.1 Detailed Description

Entities with this component start an action when clicked on.

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

- [ui.h](#)

## 5.5 ComponentWrapper Struct Reference

Used to store the component, its type and its id.

```
#include <ecs.h>
```

### Data Fields

- `uint64_t` **id**  
*The component id.*
- `int` **type\_id**  
*The id referring to the component type.*
- `void *` **component**  
*A pointer to the component itself.*



### 5.5.1 Detailed Description

Used to store the component, its type and its id.

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

- [ecs.h](#)

## 5.6 Entity Struct Reference

The entity structure for the ECS.

```
#include <ecs.h>
```

### Public Member Functions

- [VEC](#) (uint64\_t) components  
*A vector of [ComponentWrapper](#) containing the entity's components.*

### Data Fields

- uint64\_t [id](#)  
*The entity's id.*

### 5.6.1 Detailed Description

The entity structure for the ECS.

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

- [ecs.h](#)

## 5.7 HashMap Struct Reference

A hash map.

```
#include <hash_map.h>
```

### Public Member Functions

- [VEC](#) ([LinkedList](#)) bucket  
*The vector that stores the entries.*

## Data Fields

- `uint64_t(* hash_function )(void *)`  
*The function used for hashing the values stored in the [HashMap](#)*
- `char(* comp_function )(void *, void *)`  
*The function used to compare values in the [HashMap](#)*
- `uint length`  
*Length of the bucket.*
- `uint size`  
*Numberb of elements in the hashmap.*

### 5.7.1 Detailed Description

A hash map.

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

- [hash\\_map.h](#)

## 5.8 HashMapEntry Struct Reference

An entry in a [HashMap](#), i.e. a key-value pair.

```
#include <hash_map.h>
```

## Data Fields

- `void * key`
- `void * value`
- `uint64_t hash`  
*The hash of value*

### 5.8.1 Detailed Description

An entry in a [HashMap](#), i.e. a key-value pair.

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

- [hash\\_map.h](#)

## 5.9 Hoverable Struct Reference

Entities with this component show text when hovered.

```
#include <ui.h>
```

### Data Fields

- `SDL_Rect * rect`
- `char * text`

#### 5.9.1 Detailed Description

Entities with this component show text when hovered.

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

- [ui.h](#)

## 5.10 Inputs Struct Reference

stores keys and mouse buttons

```
#include <input.h>
```

### Data Fields

- `int * keys`  
*uses SDL Scancodes as indices*
- `Uint64 key_nb`  
*number of keys currently in*
- `char mouse`  
*1st bit = mb\_left; 2nd bit = mb\_middle; 3rd bit = mb\_right*

#### 5.10.1 Detailed Description

stores keys and mouse buttons

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

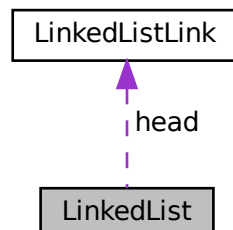
- [input.h](#)

## 5.11 LinkedList Struct Reference

A singly linked list.

```
#include <linked_list.h>
```

Collaboration diagram for LinkedList:



## Data Fields

- [LinkedListLink](#) \* [head](#)

*Pointer to the the first link of the list. `NULL` if empty.*

### 5.11.1 Detailed Description

A singly linked list.

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

- [linked\\_list.h](#)

## 5.12 LinkedListLink Struct Reference

A link of [LinkedList](#)

```
#include <linked_list.h>
```

## Data Fields

- void \* [data](#)
- struct \_Lk \* [next](#)

*Pointer to the next link in the list. `NULL` if last.*

### 5.12.1 Detailed Description

A link of [LinkedList](#)

### 5.12.2 Field Documentation

#### 5.12.2.1 **data** void\* `LinkedListLink::data`

Pointer to this link's data. Figuring out which type it is up to the user.

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

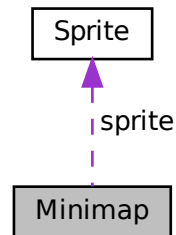
- [linked\\_list.h](#)

## 5.13 Minimap Struct Reference

Component that corresponds to the minimap.

```
#include <ui.h>
```

Collaboration diagram for Minimap:



### Data Fields

- [Sprite](#) \* **sprite**
- `SDL_Rect` \* **rect**

#### 5.13.1 Detailed Description

Component that corresponds to the minimap.

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

- [ui.h](#)

## 5.14 Position Struct Reference

A component that contains the world space coordinates of an entity.

```
#include <camera.h>
```

### Data Fields

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

### 5.14.1 Detailed Description

A component that contains the world space coordinates of an entity.

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

- [camera.h](#)

## 5.15 Rc Struct Reference

### Data Fields

- `uintptr_t` **counter**
- `void *` **ref**

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

- `asset_manager.c`

## 5.16 Sprite Struct Reference

### Data Fields

- `SDL_Texture *` **texture**
- `SDL_Rect *` **rect**

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

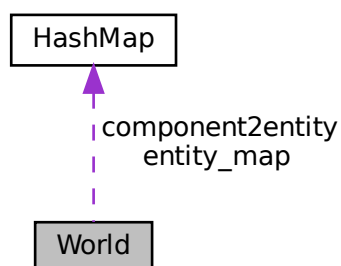
- `sprite.h`

## 5.17 World Struct Reference

The world structure used to store the different parts of the ECS.

```
#include <ecs.h>
```

Collaboration diagram for World:



## Public Member Functions

- [VEC](#) (uint) component\_sizes
- [void](#) ([VEC](#)() \*component\_free)(void \*)
- [VEC](#) ([ComponentWrapper](#)) components  
A vector of [ComponentWrapper](#) containing all the components.
- [VEC](#) ([Entity](#)) entities  
A vector of [Entity](#) containing all the entities.
- [VEC](#) (uint) component\_sparsity  
Stores the available spaces in *components* that entity deletion created.
- [VEC](#) (uint) entity\_sparsity  
Stores the available spaces in *entities* that entity deletion created.

## Data Fields

- [HashMap](#) entity\_map
- [HashMap](#) component2entity
- uint last\_component  
Indicates the id the next component to be added should take.

### 5.17.1 Detailed Description

The world structure used to store the different parts of the ECS.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 [VEC](#)() `World::VEC (uint )`

A vector containing all the sizes corresponding to each of the components' types

#### 5.17.2.2 [void](#)() `World::void (VEC() * component_free )`

A vector of functions used to free each of the components (one function per type)

### 5.17.3 Field Documentation

#### 5.17.3.1 [component2entity](#) [HashMap](#) `World::component2entity`

A [HashMap](#) with `uint64_t` as keys and `uint64_t` as values, the keys are components'ids and the values are entities'ids. It establishes for each component the list of the entities currently linked to it

### 5.17.3.2 entity\_map `HashMap` `World::entity_map`

A `HashMap` with `Bitflag` as keys and `VEC (uint64_t)` as values, the map is used to easily access the list of entities corresponding to the system represented by the `Bitflag` key

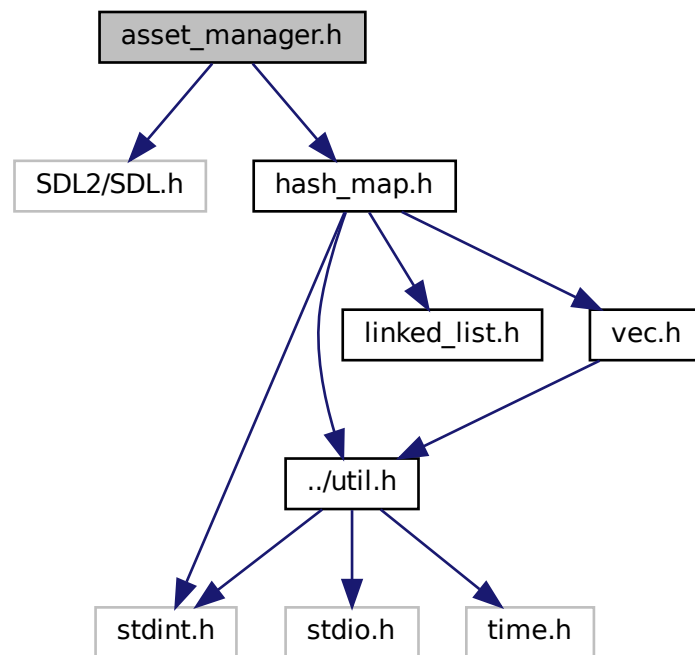
The documentation for this struct was generated from the following file:

- [ecs.h](#)

## 6 File Documentation

### 6.1 asset\_manager.h File Reference

```
#include <SDL2/SDL.h>
#include "hash_map.h"
Include dependency graph for asset_manager.h:
```



### Functions

- void `init_asset_manager` ()
- void \* `get_texture` (char \*t, `SDL_Renderer` \*renderer, `SDL_Window` \*window)
- void \* `load_texture` (char \*t, `SDL_Renderer` \*renderer, `SDL_Window` \*window)



## Variables

- [HashMap ASSET\\_STORE](#)

*Stores and manages the textures used in the game.*

### 6.1.1 Function Documentation

**6.1.1.1 get\_texture()** `void* get_texture (`  
    `char * t,`  
    `SDL_Renderer * renderer,`  
    `SDL_Window * window )`

Returns a pointer to the texture from file `t`. Will add it to the `ASSET_STORE` if it is not in it yet

**6.1.1.2 init\_asset\_manager()** `void init_asset_manager ( )`

Initializes the `ASSET_STORE`; must be called before any call to `get_texture` or `load_texture`

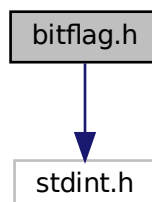
**6.1.1.3 load\_texture()** `void* load_texture (`  
    `char * t,`  
    `SDL_Renderer * renderer,`  
    `SDL_Window * window )`

Loads the texture from file `t` in the `ASSET_STORE`. While calling it multiple times with the same `t` shouldn't fail, it is unadvisable as slow. Crashes on invalid file path or texture creation.

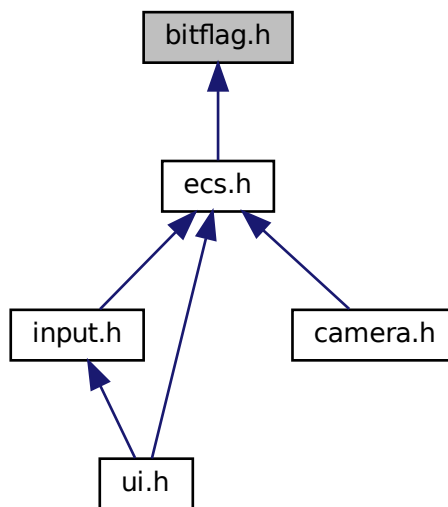
## 6.2 bitflag.h File Reference

```
#include <stdint.h>
```

Include dependency graph for `bitflag.h`:



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



## Macros

- `#define bitflag_get(b, r) (((b) >> (r)) & 1)`  
expands to the  $r$ th least significant bit of  $b$
- `#define bitflag_set(b, r, v) ((v) ? (1 << (r)) | (b) : (~(1 << (r))) & (b))`  
expands to the value of  $b$  with its  $r$ th least significant bit set to  $v$

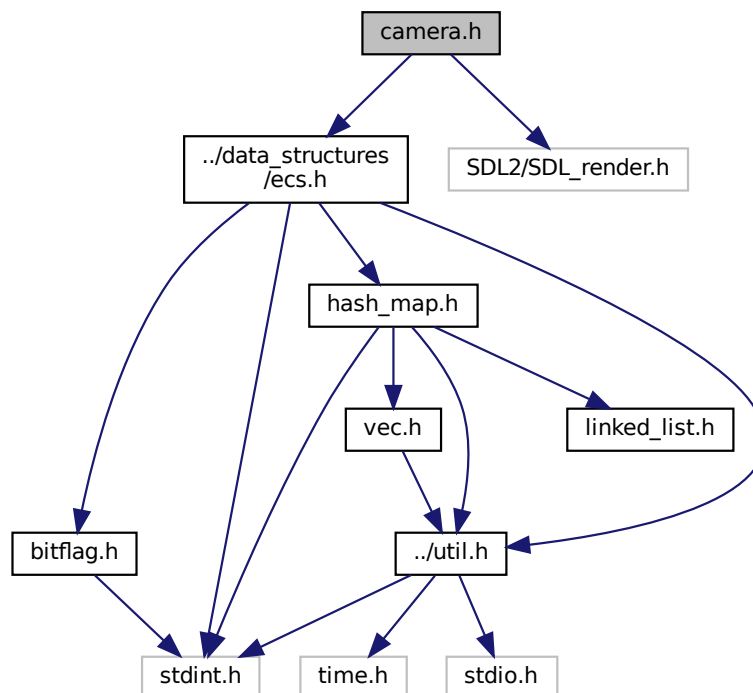
## Typedefs

- `typedef uint64_t Bitflag`

## 6.3 camera.h File Reference

```
#include "../data_structures/ecs.h"
#include <SDL2/SDL_render.h>
```

Include dependency graph for camera.h:



## Data Structures

- struct [Camera](#)
- struct [Position](#)

*A component that contains the world space coordinates of an entity.*

## Macros

- `#define WIN_H 360`  
*The main window's height.*
- `#define WIN_W 640`  
*The main window's width.*

## Functions

- [Position](#) `world2screenspace` ([Position](#) \*p, [Camera](#) \*cam)  
*Transfers p to screenspace, according to cam*
- [Position](#) `screen2worldspace` ([Position](#) \*p, [Camera](#) \*cam)  
*Transfers p to worldspace, according to cam*
- void `render` ([World](#) \*w, [SDL\\_Renderer](#) \*rdr, [Camera](#) \*cam)

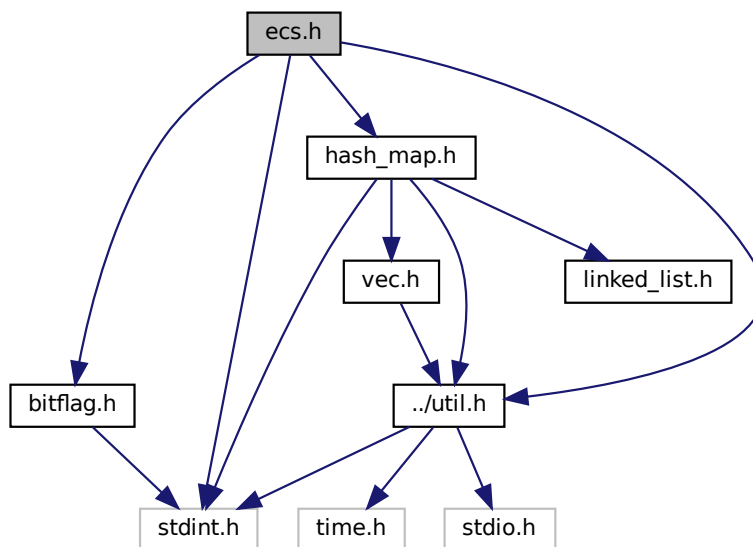
### 6.3.1 Function Documentation

**6.3.1.1 render()** `void render (`  
    `World * w,`  
    `SDL_Renderer * rdr,`  
    `Camera * cam )`

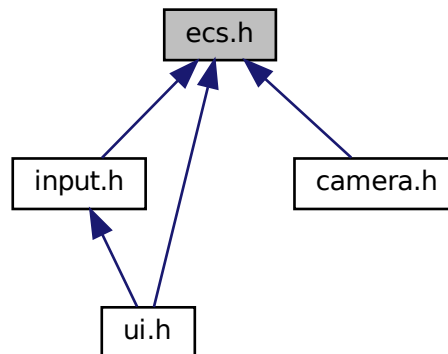
Renders any entity with a `Position` and a `Sprite`, according to `cam`. Said position must be in worldspace coordinates

## 6.4 ecs.h File Reference

```
#include "../util.h"
#include "bitflag.h"
#include "hash_map.h"
#include <stdint.h>
Include dependency graph for ecs.h:
```



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



## Data Structures

- struct [ComponentWrapper](#)  
*Used to store the component, its type and its id.*
- struct [Entity](#)  
*The entity structure for the ECS.*
- struct [World](#)  
*The world structure used to store the different parts of the ECS.*

## Macros

- #define [register\\_component](#)(w, tp) [register\\_component\\_inner\\_callback](#)((w), sizeof(tp), free)
- #define [register\\_component\\_callback](#)(w, tp, callback) [register\\_component\\_inner\\_callback](#)((w), sizeof(tp), (callback))
- #define [parallelize\\_query](#)(erefs, commands)

## Typedefs

- typedef uint64\_t [EntityRef](#)

## Functions

- char [eq\\_u64](#) (void \*a, void \*b)
- [World](#) [world\\_new](#) ()  
*Returns a new initialized [World](#) structure.*
- void [world\\_free](#) ([World](#) \*)  
*Frees a [World](#) structure created using [world\\_new](#)*
- int [register\\_component\\_inner\\_callback](#) ([World](#) \*w, int size, void(\*callback)(void \*))
- void [register\\_system\\_requirement](#) ([World](#) \*w, Bitflag b)
- [Entity](#) \* [spawn\\_entity](#) ([World](#) \*w)

Spawns an *Entity* into the world and returns a pointer to it.

- void `ecs_add_component` (World \*w, Entity \*e, int cid, void \*c)
- void `despawn_entity` (World \*w, Entity \*e)

Despawns an *Entity*

- Entity \* `get_entity` (World \*w, EntityRef ref)

Returns an *Entity* pointer corresponding to the passed reference.

- VEC (EntityRef) `world_query`(World \*w
- void \* `entity_get_component` (World \*w, Entity \*e, int type)

## Variables

- Bitflag \* b

### 6.4.1 Macro Definition Documentation

**6.4.1.1 `parallelize_query`** #define `parallelize_query`(  
     erefs,  
     commands )

Value:

```
{
    _Pragma("omp parallel") {
        _Pragma("omp for") {
            for (uint i = 0; i < vec_len(erefs); i++) {
                EntityRef ei = erefs[i];
                commands;
            }
        }
    }
}
```

```
//
//
//
//
//
//
//
```

Expands to a parallel query on the elements of `erefs`. `erefs` is expected to be the return value of `world_query`, and must be a glvalue. Commands are executed with the understanding that they can access the element they work on with `ei`. Note that spawning the threads is a significant overhead. For trivial cases, using the sequential method can be faster. If unsure, use `TIME` to benchmark both usecases. Note that Valgrind will detect some "possibly lost memory". This is intended behavior, see [https://gcc.gnu.org/bugzilla/show\\_bug.cgi?id=36298](https://gcc.gnu.org/bugzilla/show_bug.cgi?id=36298)

**6.4.1.2 `register_component`** #define `register_component`(  
     w,  
     tp ) `register_component_inner_callback`((w), sizeof(tp), free)

`register_component`(World\*, type) where type is the type of the component. Registers a new component that uses `free` as a way to free it

**6.4.1.3 `register_component_callback`** #define `register_component_callback`(  
     w,  
     tp,  
     callback ) `register_component_inner_callback`((w), sizeof(tp), (callback))

`register_component`(World\*, type, void (\*callback)(void \*)) where type is the type of the component. Registers a new component using a callback function to free it

## 6.4.2 Typedef Documentation

### 6.4.2.1 EntityRef `typedef uint64_t EntityRef`

Note that this reference is only valid until the number of entities decreases

## 6.4.3 Function Documentation

### 6.4.3.1 `ecs_add_component()` `void ecs_add_component (` `World * w,` `Entity * e,` `int cid,` `void * c )`

Links a component to an `Entity`. The component itself need to live as long as the world does (beware of scopes)

### 6.4.3.2 `entity_get_component()` `void* entity_get_component (` `World * w,` `Entity * e,` `int type )`

Returns a pointer to the component of type `type` linked to the `Entity`, if no component of this type is linked the the `Entity` the NULL pointer is returned

### 6.4.3.3 `eq_u64()` `char eq_u64 (` `void * a,` `void * b )`

Returns a normalized boolean (0 or 1) indicating if the two arguments are equal when both interpreted as `uint64_t`

### 6.4.3.4 `register_component_inner_callback()` `int register_component_inner_callback (` `World * w,` `int size,` `void(*) (void *) callback )`

Registers a new component using a callback function to free it, the size of the component's type needs to be passed instead of the type itself

### 6.4.3.5 `register_system_requirement()` `void register_system_requirement (` `World * w,` `Bitflag b )`

Updates the `entity_map` of the world to take into account the system represented by the `Bitflag` argument

#### 6.4.3.6 VEC() VEC ( EntityRef )

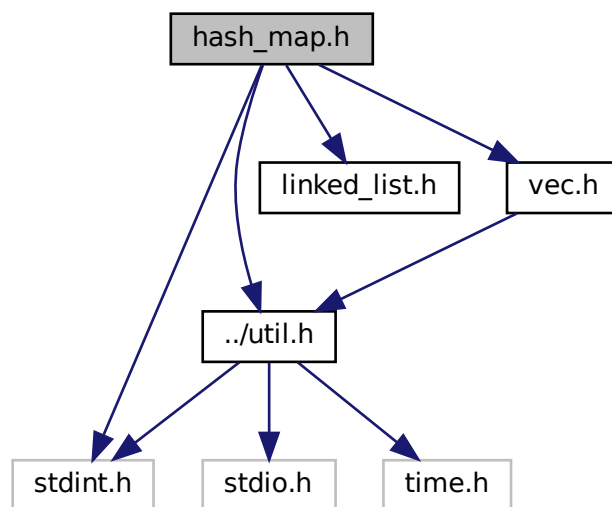
Returns a vector of `EntityRef` referencing entities corresponding to the system described by the `Bitflag` argument. If you want to modify the `World` based on the return value of this function, use `world_query_mut` instead. The system needs to be registered using `register_system_requirement` before using this function

Returns a pointer to a vector of `EntityRef` referencing entities corresponding to the system described by the `Bitflag` argument. The system needs to be registered using `register_system_requirement` before using this function

## 6.5 hash\_map.h File Reference

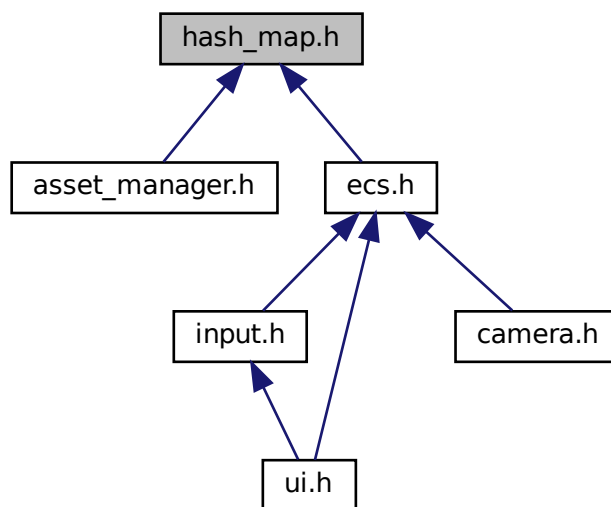
```
#include <stdint.h>
#include "../util.h"
#include "linked_list.h"
#include "vec.h"
```

Include dependency graph for `hash_map.h`:





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



## Data Structures

- struct [HashMapEntry](#)  
An entry in a [HashMap](#), i.e. a key-value pair.
- struct [HashMap](#)  
A hash map.

## Macros

- #define [HASHMAP\\_DEFAULT\\_LENGTH](#) 32  
The initial length of the internal array of a [HashMap](#)
- #define [HASHMAP\\_OCCUP\\_MAX](#) 0.7  
The occupation ratio of a [HashMap](#) over which it grows.
- #define [HASHMAP\\_OCCUP\\_MIN](#) 0.3  
The occupation ratio of a [HashMap](#) below which it shrinks.

## Functions

- uint64\_t [hash\\_str](#) (void \*)  
A polynomial rolling hash for strings.
- uint64\_t [hash\\_u64](#) (void \*)  
A FNV hash function for 64 bit integers.
- [HashMap](#) [hash\\_map\\_create](#) (uint64\_t(\*hash)(void \*), char(\*cmp)(void \*, void \*))
- void [hash\\_map\\_free\\_callback](#) ([HashMap](#) \*h, void(\*callback)(void \*))  
Frees h, calling *callback* on each entry to free it.
- void [hash\\_map\\_free](#) ([HashMap](#) \*h)

- *Same as `hash_map_free_callback` but uses `hash_map_entry_free` as callback.*  
• void `hash_map_free_void` (void \*h)  
*Same as `hash_map_free`, deprecated.*
- int `hash_map_insert_callback` (HashMap \*h, void \*k, void \*v, void(\*callback)(void \*))
- int `hash_map_insert` (HashMap \*h, void \*k, void \*v)
- int `hash_map_delete_callback` (HashMap \*h, void \*k, void(\*callback)(void \*))  
*deletes the entry with key k using callback*
- int `hash_map_delete` (HashMap \*h, void \*k)  
*Same as `hash_map_delete_callback` but uses `hash_map_entry_free` as callback.*
- void \* `hash_map_get` (HashMap \*h, void \*k)

## 6.5.1 Function Documentation

**6.5.1.1 `hash_map_create()`** `HashMap hash_map_create (`  
    `uint64_t(*) (void *) hash,`  
    `char(*) (void *, void *) cmp )`

Creates and returns a new `HashMap` that uses `hash` as the hash function and `cmp` as the comparison function

**6.5.1.2 `hash_map_get()`** `void* hash_map_get (`  
    `HashMap * h,`  
    `void * k )`

Returns the value associated with key `k`, or a null pointer if there is no such pair

**6.5.1.3 `hash_map_insert()`** `int hash_map_insert (`  
    `HashMap * h,`  
    `void * k,`  
    `void * v )`

Same as `hash_map_insert_callback` but uses `hash_map_entry_free` as callback

**6.5.1.4 `hash_map_insert_callback()`** `int hash_map_insert_callback (`  
    `HashMap * h,`  
    `void * k,`  
    `void * v,`  
    `void(*) (void *) callback )`

Inserts the key-value pair `k,v` in `h`, deleting any previous entry of key `k` with `callback`

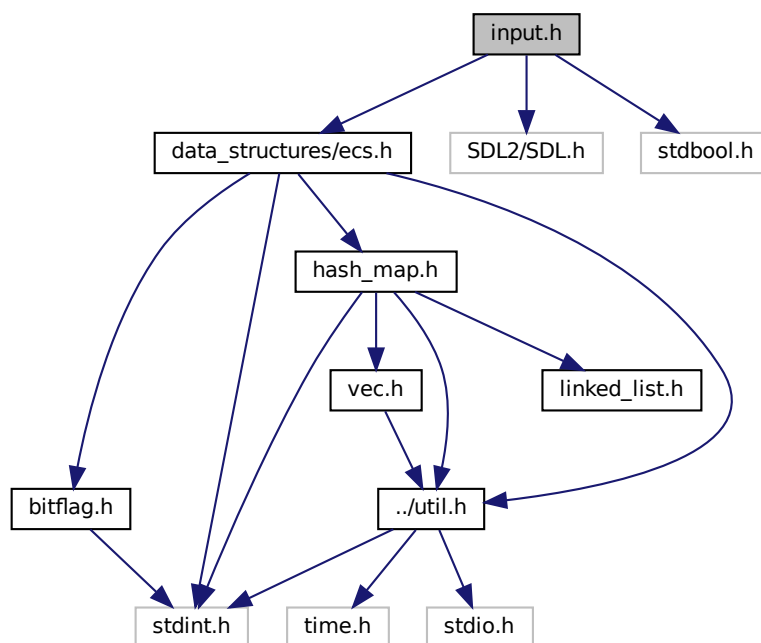
## 6.6 input.h File Reference

```
#include "data_structures/ecs.h"
```

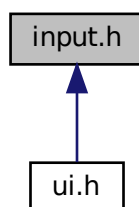
```
#include <SDL2/SDL.h>
```

```
#include <stdbool.h>
```

Include dependency graph for input.h:



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



### Data Structures

- struct [Inputs](#)  
*stores keys and mouse buttons*

## Macros

- `#define KEY_PRESSED 0`  
*the instant the key is pressed*
- `#define KEY_RELEASED 1`  
*the instant the key is released*
- `#define KEY_DOWN 2`  
*starts on press (included), ends on release (not included)*
- `#define inputs_is_key_in_from_scancode(inputs, scancode) ((inputs)->keys[(scancode)])`
- `#define inputs_is_key_in(inputs, key) ((inputs)->keys[SDL_GetScancodeFromKey(key)])`
- `#define inputs_is_mouse_button_in(inputs, button) (((inputs)->mouse >> ((button)-1)) & 1)`
- `#define inputs_update_key_in(inputs, key, new_val)`
- `#define inputs_update_mouse_button_in(inputs, button, new_val)`

## Typedefs

- `typedef Uint8 KeyState`
- `typedef Uint8 MouseButton`
- `typedef void(* KeyEvent)(World *, Entity *, Inputs *, KeyState)`  
*type of callback functions for the key events*

## Functions

- `Inputs * inputs_new ()`  
*creates a new Inputs instance*
- `void inputs_free (Inputs *)`  
*frees the Inputs instance*
- `void inputs_update_key_in_from_scancode (Inputs *inputs, SDL_Scancode scancode, bool new_val)`
- `void inputs_run_callbacks (World *, Inputs *, KeyState)`  
*calls all the callbacks for the keyevent*
- `Uint8 mouse_in_rect (SDL_Rect *rect)`  
*Checks if the mouse is in the rectangle.*

### 6.6.1 Macro Definition Documentation

**6.6.1.1 inputs\_is\_key\_in** `#define inputs_is_key_in(  
inputs,  
key ) ((inputs)->keys[SDL_GetScancodeFromKey(key)])`

the state of a key accessed using `SDL_KeyCode` bool `inputs_is_key_in(Inputs*, SDL_KeyCode)`

**6.6.1.2 inputs\_is\_key\_in\_from\_scancode** `#define inputs_is_key_in_from_scancode(  
inputs,  
scancode ) ((inputs)->keys[(scancode)])`

the state of a key accessed using `SDL_Scancode` !!!!!!!!! this does not take into account non QWERTY keyboards / remaps !!!!!!!!! bool `inputs_is_key_in_from_scancode(Input*,SDL_Scancode)`

**6.6.1.3 inputs\_is\_mouse\_button\_in** `#define inputs_is_mouse_button_in(  
     inputs,  
     button )   (((inputs)->mouse >> ((button)-1)) & 1)`

the state of a mouse button bool `inputs_is_mouse_button_in(Inputs*,MouseButton)`

**6.6.1.4 inputs\_update\_key\_in** `#define inputs_update_key_in(  
     inputs,  
     key,  
     new_val )`

**Value:**

`(inputs_update_key_in_from_scancode(inputs, SDL_GetScancodeFromKey(key), \`  
`new_val))`

updates the state of a key using SDL\_KeyCode void `inputs_update_key_in(Input*,SDL_KeyCode,bool)`

**6.6.1.5 inputs\_update\_mouse\_button\_in** `#define inputs_update_mouse_button_in(  
     inputs,  
     button,  
     new_val )`

**Value:**

`((inputs)->mouse = (((!(new_val)) << ((button)-1)) | \`  
`((inputs)->mouse & (~1 << ((button)-1))))))`

updates the state of a mouse button MouseButton `inputs_update_mouse_button_in(Input*,MouseButton,bool)`

## 6.6.2 Typedef Documentation

**6.6.2.1 MouseButton** `typedef Uint8 MouseButton`

describes any of the following: SDL\_BUTTON\_LEFT,SDL\_BUTTON\_MIDDLE, SDL\_BUTTON\_RIGHT

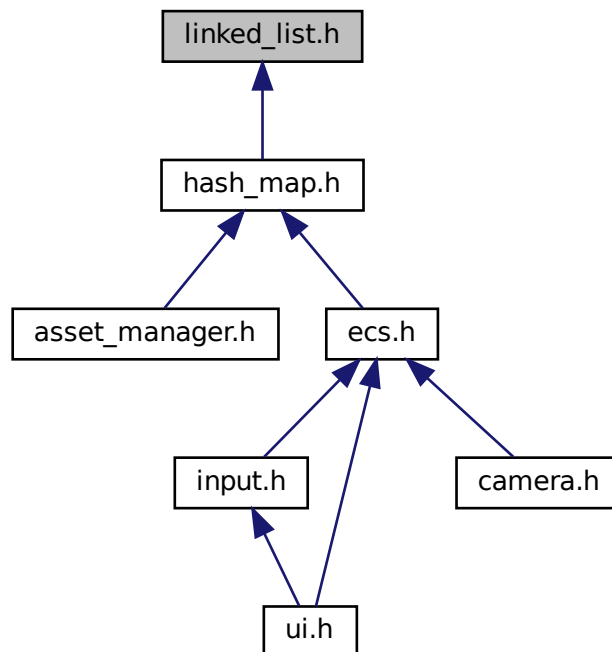
## 6.6.3 Function Documentation

**6.6.3.1 inputs\_update\_key\_in\_from\_scancode()** `void inputs_update_key_in_from_scancode (  
     Inputs * inputs,  
     SDL_Scancode scancode,  
     bool new_val )`

updates the state of a key using SDL\_Scancode !!!!!!!!! this does not take into account non QWERTY keyboards / remaps !!!!!!!!! void inputs\_update\_key\_in\_from\_scancode(Input\*,SDL\_Scancode,bool)

## 6.7 linked\_list.h File Reference

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



### Data Structures

- struct [LinkedListLink](#)  
A link of [LinkedList](#)
- struct [LinkedList](#)  
A singly linked list.

### Functions

- [LinkedList](#) [linked\\_list\\_create](#) ()  
Creates a [LinkedList](#)
- int [linked\\_list\\_insert](#) ([LinkedList](#) \*l, void \*e, int i)
- int [linked\\_list\\_remove](#) ([LinkedList](#) \*l, int i)  
Same as [linked\\_list\\_remove\\_callback](#), with `free` as the callback
- int [linked\\_list\\_remove\\_callback](#) ([LinkedList](#) \*l, int i, void(\*callback)(void \*))
- void [linked\\_list\\_free](#) ([LinkedList](#) \*)  
Same as [linked\\_list\\_free](#), with `free` as the callback
- void [linked\\_list\\_free\\_callback](#) ([LinkedList](#) \*l, void(\*callback)(void \*))
- void \* [linked\\_list\\_get](#) ([LinkedList](#) \*l, int i)  
Returns the `data` field of the `i`th element of `l`

### 6.7.1 Function Documentation

**6.7.1.1 linked\_list\_free\_callback()** `void linked_list_free_callback (`  
    `LinkedList * l,`  
    `void(*) (void *) callback )`

Frees `l`, calling `callback` on the data fields of each link as a way to free them

**6.7.1.2 linked\_list\_insert()** `int linked_list_insert (`  
    `LinkedList * l,`  
    `void * e,`  
    `int i )`

Add `e` as an element of `l` at index `i` Returns 0 on success, -1 on allocation error and -2 if `i` is out of range

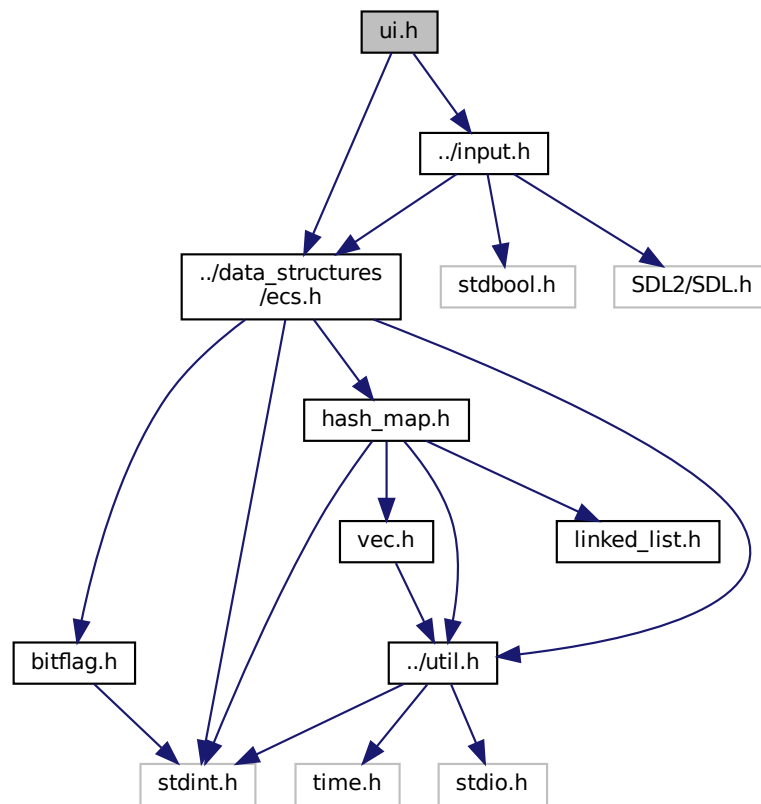
**6.7.1.3 linked\_list\_remove\_callback()** `int linked_list_remove_callback (`  
    `LinkedList * l,`  
    `int i,`  
    `void(*) (void *) callback )`

Removes element at index `i` in `l`, running `callback` on its data as a way to free it

## 6.8 ui.h File Reference

```
#include "../data_structures/ecs.h"
#include "../input.h"
#include "sprite.h"
```

Include dependency graph for ui.h:



## Data Structures

- struct [Background](#)  
*Entities with this component are the background of the user interface.*
- struct [Clickable](#)  
*Entities with this component start an action when clicked on.*
- struct [Button](#)
- struct [Minimap](#)  
*Component that corresponds to the minimap.*
- struct [Hoverable](#)  
*Entities with this component show text when hovered.*

## Functions

- void [render\\_ui](#) ([World](#) \*w, [SDL\\_Renderer](#) \*rdr)  
*Renders any entity that has user interface related component.*
- [Entity](#) \* [spawn\\_clickable](#) ([World](#) \*w, [Clickable](#) \*object, [KeyEvent](#) \*event)  
*Adds a clickable to the world.*
- void [clickable\\_event](#) ([World](#) \*w, [Entity](#) \*entity, [Inputs](#) \*in, [KeyState](#) keystate)
- void [render\\_hoverable](#) ([SDL\\_Rect](#) \*rect, char \*text)



### 6.8.1 Function Documentation

**6.8.1.1 clickable\_event()** `void clickable_event (`  
    `World * w,`  
    `Entity * entity,`  
    `Inputs * in,`  
    `KeyState keystate )`

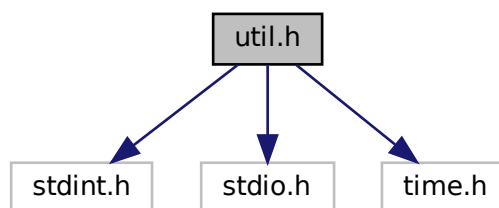
The `KeyEvent` of the entities associated with a clickable component, there are different cases, if the mouse is out of the sprite, it is set `is_clicked` to 0 as for doing nothing, if the left click is pressed on the sprite, it will be set to 1 and if it is set to 1 and the click is released then it will be set to 2. The idea is that if set to 1 there will be a visual change by darkening the sprite and if it set to 2 it will start the action linked to the sprite. It must be noted that if you click on the sprite, move your mouse out and then release the click it will do nothing as a way to correct missclicks.

**6.8.1.2 render\_hoverable()** `void render_hoverable (`  
    `SDL_Rect * rect,`  
    `char * text )`

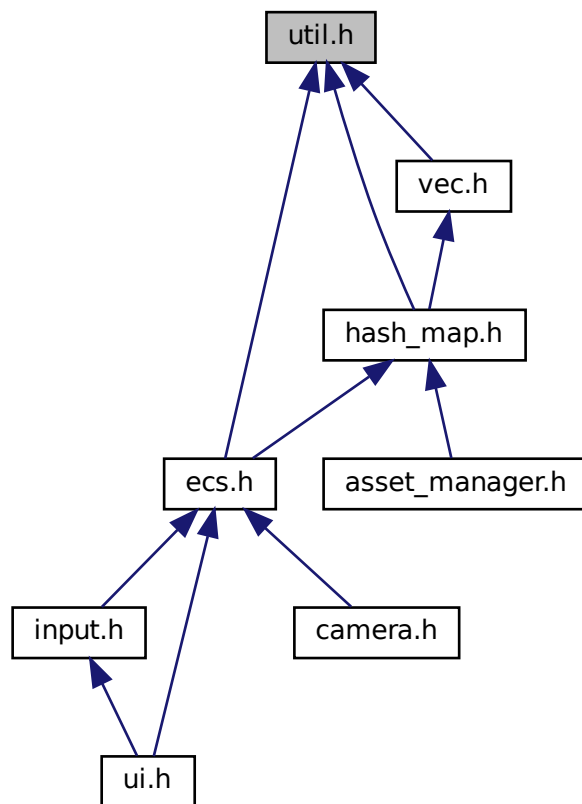
This function is used to render the entities associated with a hoverable component

## 6.9 util.h File Reference

```
#include "errors.h"
#include <stdint.h>
#include <stdio.h>
#include <time.h>
Include dependency graph for util.h:
```



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



## Macros

- `#define HANDLE_ERROR(err, message, callback)`  
*prints message when err != 0, and then runs callback*
- `#define TARGET_FRAMETIME (1000 / 60)`  
*The framerate that the game should try to maintain, in milliseconds.*
- `#define ASSERT(a)`
- `#define TIME(label, commands)`  
*Benchmarks commands*
- `#define max(a, b) ((a > b) ? (a) : (b))`
- `#define min(a, b) ((a < b) ? (a) : (b))`

## Typedefs

- `typedef unsigned int uint`

## Functions

- void `free_nothing` (void \*)  
*Does nothing. Used when a callback is necessary but nothing is to be done.*
- char `not_strerror` (void \*a, void \*b)  
*Strictly equivalent to `!strcmp(a, b)`. Used as a callback.*
- void `sleep_nano` (uint64\_t n)  
*Sleeps the calling thread for n nanoseconds. Uses GNU extensions.*

### 6.9.1 Macro Definition Documentation

**6.9.1.1 ASSERT** `#define ASSERT(  
a )`

#### Value:

```
{
    if (!(a)) {
        fprintf(stderr, "[%s:%d] assertion '%s' failed\n", __FILE__, __LINE__,
            #a);
        return ASSERTION_FAILED;
    }
}
```

Verify that `a != 0`. Otherwise, prints an error and exits the current function with error `-1`

**6.9.1.2 HANDLE\_ERROR** `#define HANDLE_ERROR(  
err,  
message,  
callback )`

#### Value:

```
{
    if (err) {
        fprintf(stderr, "[%s:%d] %s\n", __FILE__, __LINE__, message);
        callback;
    }
}
```

prints message when `err != 0`, and then runs `callback`

**6.9.1.3 TIME** `#define TIME(  
label,  
commands )`

#### Value:

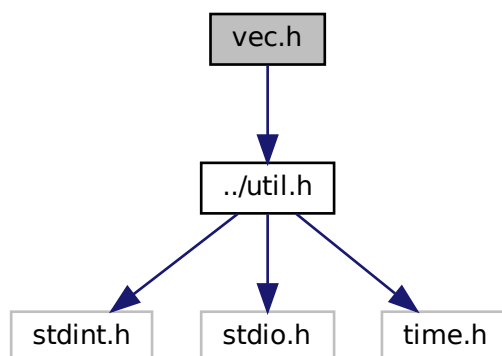
```
{
    struct timespec _beg;
    clock_gettime(CLOCK_MONOTONIC, &_beg);
    commands;
    struct timespec _end;
    clock_gettime(CLOCK_MONOTONIC, &_end);
    uint64_t elapsed = (uint64_t)(_end.tv_sec - _beg.tv_sec) * 1000000000 +
        (uint64_t)(_end.tv_nsec - _beg.tv_nsec);
    printf("\033[38;5;96mBENCHMARKING\033[0m] %s took %fs\n", label,
        (double)elapsed / 1000000000);
}
```

Benchmarks commands

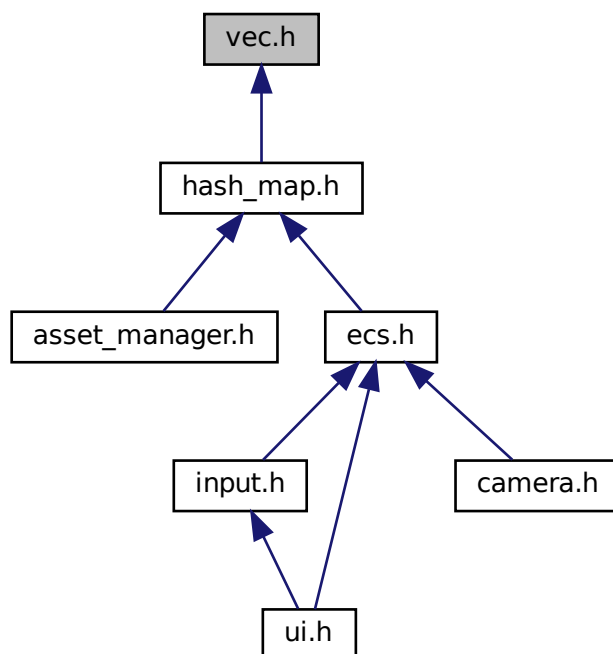
## 6.10 vec.h File Reference

```
#include "../util.h"
```

Include dependency graph for vec.h:



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



## Macros

- `#define VEC(x) x *`
- `#define VEC_INIT_CAPACITY 16`  
*The length of a `vec` at creation.*
- `#define vec_new(type) (vec_new_inner(sizeof(type)))`  
*Creates a new `vec` for type `type`*
- `#define vec_push(vec, obj) vec = (vec_push_inner(((void *) (vec)), (void *)&(obj)))`  
*adds a copy of `obj` at the end of `vec`*
- `#define vec_last(a) (a)[vec_len((a)) - 1]`  
*expands to the last element of the `vec`*

## Functions

- `VEC (void) vec_copy(VEC(void) vec)`  
*copies `vec` and returns the copy*
- `void vec_free (VEC(void) vec)`  
*frees a `vec`. This should always be used instead of `free (vec)`*
- `void vec_pop (VEC(void) vec)`
- `uint vec_len (VEC(void) vec)`  
*returns the length of `vec`. This is a  $O(1)$  operation.*
- `void vec_sort (VEC(void) vec, char(*gt)(void *a, void *b))`
- `void vec_swap (VEC(void) vec, int a, int b)`  
*swaps the elements at index `a` and `b` in `vec`*
- `char u64_gt (void *a, void *b)`
- `void vec_remove (void *vec, int a)`  
*removes element at index `a` in `vec`*

## Variables

- `void * obj`

### 6.10.1 Detailed Description

file This file defines a redimensionnable array, hereafter referred to as `vec`. Relevant informations about the content of the `vec` are stored just before the pointer that the user manipulates

### 6.10.2 Macro Definition Documentation

#### 6.10.2.1 VEC `#define VEC (` `x ) x *`

A macro that extends to a pointer to `x`, to differentiate vectors from arbitrary pointers

### 6.10.3 Function Documentation

**6.10.3.1 u64\_gt()** `char u64_gt (`  
    `void * a,`  
    `void * b )`

`a` and `b` are assumed to be `uint64_t`. returns true iff `&(uint64_t*) a >= &(uint64_t*) b`. Used for `vec←_sort`

**6.10.3.2 VEC()** `VEC (`  
    `void )`

copies `vec` and returns the copy

adds a copy of what `obj` points to at the end of `vec`. returns a potentially new pointer to the `vec`

**6.10.3.3 vec\_pop()** `void vec_pop (`  
    `VEC(void) vec )`

removes the last element of the `vec`. Doesn't return it for optimisation purposes

**6.10.3.4 vec\_sort()** `void vec_sort (`  
    `VEC(void) vec,`  
    `char(*) (void *a, void *b) gt )`

sorts `vec` in place, using `gt` as a way to compare elements. `gt`'s parameters are pointers to the actually compared data, and `gt` returns true iff `a >= b`. `vec_sort` uses merge sort and is consequentially in  $O(n \log n)$

## Index

### ASSERT

util.h, 33

### asset\_manager.h, 14

get\_texture, 15

init\_asset\_manager, 15

load\_texture, 15

### Background, 4

bitflag.h, 15

### Button, 4

### Camera, 5

zoom, 5

camera.h, 16

render, 18

### Clickable, 6

clickable\_event

ui.h, 31

component2entity

World, 13

ComponentWrapper, 6

### data

LinkedListLink, 10

### ecs.h, 18

ecs\_add\_component, 21

entity\_get\_component, 21

EntityRef, 21

eq\_u64, 21

parallelize\_query, 20

register\_component, 20

register\_component\_callback, 20

register\_component\_inner\_callback, 21

register\_system\_requirement, 21

VEC, 21

ecs\_add\_component

ecs.h, 21

### Entity, 7

entity\_get\_component

ecs.h, 21

entity\_map

World, 13

EntityRef

ecs.h, 21

eq\_u64

ecs.h, 21

get\_texture

asset\_manager.h, 15

### HANDLE\_ERROR

util.h, 33

### hash\_map.h, 22

hash\_map\_create, 24

hash\_map\_get, 24

hash\_map\_insert, 24

hash\_map\_insert\_callback, 24

hash\_map\_create

hash\_map.h, 24

hash\_map\_get

hash\_map.h, 24

hash\_map\_insert

hash\_map.h, 24

hash\_map\_insert\_callback

hash\_map.h, 24

HashMap, 7

HashMapEntry, 8

Hoverable, 8

init\_asset\_manager

asset\_manager.h, 15

input.h, 25

inputs\_is\_key\_in, 26

inputs\_is\_key\_in\_from\_scancode, 26

inputs\_is\_mouse\_button\_in, 26

inputs\_update\_key\_in, 27

inputs\_update\_key\_in\_from\_scancode, 27

inputs\_update\_mouse\_button\_in, 27

MouseButton, 27

Inputs, 9

inputs\_is\_key\_in

input.h, 26

inputs\_is\_key\_in\_from\_scancode

input.h, 26

inputs\_is\_mouse\_button\_in

input.h, 26

inputs\_update\_key\_in

input.h, 27

inputs\_update\_key\_in\_from\_scancode

input.h, 27

inputs\_update\_mouse\_button\_in

input.h, 27

linked\_list.h, 28

linked\_list\_free\_callback, 29

linked\_list\_insert, 29

linked\_list\_remove\_callback, 29

linked\_list\_free\_callback

linked\_list.h, 29

linked\_list\_insert

linked\_list.h, 29

linked\_list\_remove\_callback

linked\_list.h, 29

LinkedList, 9

LinkedListLink, 10

data, 10

load\_texture

asset\_manager.h, 15

Minimap, 11

MouseButton

input.h, 27

- parallelize\_query
  - ecs.h, [20](#)
- Position, [11](#)
- Rc, [12](#)
- register\_component
  - ecs.h, [20](#)
- register\_component\_callback
  - ecs.h, [20](#)
- register\_component\_inner\_callback
  - ecs.h, [21](#)
- register\_system\_requirement
  - ecs.h, [21](#)
- render
  - camera.h, [18](#)
- render\_hoverable
  - ui.h, [31](#)
- Sprite, [12](#)
- TIME
  - util.h, [33](#)
- u64\_gt
  - vec.h, [36](#)
- ui.h, [29](#)
  - clickable\_event, [31](#)
  - render\_hoverable, [31](#)
- util.h, [31](#)
  - ASSERT, [33](#)
  - HANDLE\_ERROR, [33](#)
  - TIME, [33](#)
- VEC
  - ecs.h, [21](#)
  - vec.h, [35](#), [36](#)
  - World, [13](#)
- vec.h, [34](#)
  - u64\_gt, [36](#)
  - VEC, [35](#), [36](#)
  - vec\_pop, [36](#)
  - vec\_sort, [36](#)
- vec\_pop
  - vec.h, [36](#)
- vec\_sort
  - vec.h, [36](#)
- void
  - World, [13](#)
- World, [12](#)
  - component2entity, [13](#)
  - entity\_map, [13](#)
  - VEC, [13](#)
  - void, [13](#)
- zoom
  - Camera, [5](#)