aZero ECS

Generated by Doxygen 1.9.3

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 COMPONENTENUM Namespace Reference

Component enumeration for usage in conjunction with the Entity std::bitset. You should create an enumeration for your new custom components. The enum should be used within ComponentManager and its methods.

Enumerations

- enum **COMPONENTBITID** { **COMP** }

4.1.1 Detailed Description

Component enumeration for usage in conjunction with the Entity std::bitset. You should create an enumeration for your new custom components. The enum should be used within ComponentManager and its methods.

Chapter 5

Class Documentation

5.1 ComponentManager Class Reference

Contains and handles all components. This class can be used in conjunction with the Entity struct to register components for an Entity object.

`#include <ECSBase.h>`

Public Member Functions

- template<typename T >
  T * RegisterComponent (Entity &_entity, const T &_initValue)
- template<typename T >
  void RemoveComponent (Entity &_entity)
- template<typename T >
  T * GetComponent (const Entity &_entity)
- template<typename T >
  T * GetComponentFast (const Entity &_entity)

5.1.1 Detailed Description

Contains and handles all components. This class can be used in conjunction with the Entity struct to register components for an Entity object.

NOTE!!!! Several things has to be added to this class before using. One BiDirectionalMap should be created as a member variable for each component. You should also write the logic within the ComponentManager::RegisterComponent(), ComponentManager::RemoveComponent(), ComponentManager::GetComponent(), and ComponentManager::GetComponentFast() for each component. Simply replace Comp1 and COMPONENTENUM↩
::Comp1 within the source code with the component type and COMPONENTENUM enumeration of the custom made component. Read the source code for examples.

5.1.2 Member Function Documentation

5.1.2.1 GetComponent()

```
template<typename T >
T * ComponentManager::GetComponent (
             const Entity & _entity )  [inline]
```

Returns a pointer to the newly registered component within a internal std::vector. The template argument specifies what type of component that will be returned.

Parameters

_entity	The Entity to get the component for
templateArg	Specifies which component type to return

Returns

Pointer to the internal component object

Nullptr if the input Entity doesn't have a component of the specified type registered for it.

5.1.2.2 GetComponentFast()

```
template<typename T >
T * ComponentManager::GetComponentFast (
            const Entity & _entity )  [inline]
```

Returns a pointer to the newly registered component within a internal std::vector. The template argument specifies what type of component that will be returned.

NOTE!!!! Be careful when using this method. The application will crash if the input Entity doesn't have a component of the specified type registered. Because of this, only use this function when efficiency is the prioriy and you're sure the Entity has the specified component registered.

Parameters

_entity	The Entity to get the component for
templateArg	Specifies which component type to return

Returns

Pointer to the internal component object

5.1.2.3 RegisterComponent()

```
template<typename T >
T * ComponentManager::RegisterComponent (
            Entity & _entity,
            const T & _initValue )  [inline]
```

Registers a new component for the input Entity object and initializes the new component using the input component object. Uses the copy constructor to copy the input component into the newly created component.

Parameters

_entity	The Entity to register the component for
_initValue	Initial value of the component

Returns

Pointer to the newly registered component within a internal std::vector

5.1.2.4 RemoveComponent()

```
template<typename T >
void ComponentManager::RemoveComponent (
            Entity & _entity )  [inline]
```

Removes a component registered to the input Entity object. Component type is specified by using template arguments. Ex. RemoveComponent<ComponentX>(_entityX) will remove a registered component of type ComponentX.

Nothing will happen if the input Entity doesn't have a component of the specified type registered for it.

Parameters

_entity	The Entity to remove the component for
templateArg	Specifies which component type to remove

Returns

void

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

- Test/ECSBase.h

5.2 ECS Class Reference

Contains everything used to handle a singular Entity Component System. Enables creation and management of Entity objects and components through the internal EntityManager and ComponentManager member variables used via the ECS::GetEntityManager() and ECS::GetComponentManager() methods.

```
#include <ECSBase.h>
```

Public Member Functions

- ECS (unsigned int _maxEntities)
- EntityManager & GetEntityManager ()
- ComponentManager & GetComponentManager ()
- void ObliterateEntity (Entity &_entity)

5.2.1 Detailed Description

Contains everything used to handle a singular Entity Component System. Enables creation and management of Entity objects and components through the internal EntityManager and ComponentManager member variables used via the ECS::GetEntityManager() and ECS::GetComponentManager() methods.

NOTE!!!! Custom made ECSystem subclasses has to be added manually to this class. ECSystem::UnBind() also has to be called within ECS::ObliterateEntity(), otherwise it won't be unbound from that ECSystem. Same goes for components. See the commented examples within this classes source code.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ECS()

```
ECS::ECS (
            unsigned int _maxEntities )  [inline]
```

Initiates the internal EntityManager variable with the input value.

Parameters

_maxEntities	Maximum number of Entity objects that the ECS instance can contain

5.2.3 Member Function Documentation

5.2.3.1 GetComponentManager()

```
ComponentManager & ECS::GetComponentManager ( )  [inline]
```

Returns a reference to a internal ComponentManager object.

Returns

Reference to a internal ComponentManager object

5.2.3.2 GetEntityManager()

```
EntityManager & ECS::GetEntityManager ( )  [inline]
```

Returns a reference to a internal EntityManager object.

Returns

Reference to a internal EntityManager object

5.2.3.3 ObliterateEntity()

```
void ECS::ObliterateEntity (
            Entity & _entity )  [inline]
```

Completely removes the Entity object, and everything linked to it, from the ECS instance by calling ComponentManager::RemoveComponent(), ECSystem::UnBind(), and EntityManager::RemoveEntity().

Parameters

_entity	The Entity to obliterate

Returns

void

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

- Test/ECSBase.h

5.3 ECSystem Class Reference

An abstract base class for systems used within the ECS framework. New systems should inherit from this and implement appropriate functionality for the ECSystem::Update() pure virtual method.

```
#include <ECSBase.h>
```

Public Member Functions

- void Bind (const Entity &_entity)
- void BindFast (const Entity &_entity)
- void UnBind (const Entity &_entity)
- void RemoveEntities ()
- virtual void **Update** ()=0

  A pure virtual function that should be implemented for an inheriting subclass. It should operate on the bound Entity objects, but that isn't mandatory.

Public Attributes

- std::bitset< MAXCOMPONENTS > componentMask

5.3.1 Detailed Description

An abstract base class for systems used within the ECS framework. New systems should inherit from this and implement appropriate functionality for the ECSystem::Update() pure virtual method.

5.3.2 Member Function Documentation

5.3.2.1 Bind()

```
void ECSystem::Bind (
            const Entity & _entity )  [inline]
```

Used to bind an Entity object to be used within the subclasses' implementation of the ECSystem::Update() pure virtual method.

Parameters

_entity	The Entity to bind to the ECSystem

Returns

void

5.3.2.2 BindFast()

```
void ECSystem::BindFast (
            const Entity & _entity )  [inline]
```

Used to bind an Entity object to be used within the subclasses' implementation of the ECSystem::Update() pure virtual method.

NOTE!!!! Be careful when using this since there is no check if the input Entity object has the required components registered.

Parameters

_entity	The Entity to bind to the ECSystem

Returns

void

5.3.2.3 RemoveEntities()

```
void ECSystem::RemoveEntities ( )  [inline]
```

Clears the list of Entity objects bound to the system.

Returns

void

5.3.2.4 UnBind()

```
void ECSystem::UnBind (
            const Entity & _entity )  [inline]
```

Used to unbind an Entity object from the ECSystem.

Parameters

_entity	The Entity to unbind from the ECSystem

Returns

void

5.3.3 Member Data Documentation

5.3.3.1 componentMask

```
std::bitset<MAXCOMPONENTS> ECSystem::componentMask
```

Describes what type of components a bound Entity should have registered. This should be overwritten in the constructor of an inheriting class.

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

- Test/ECSBase.h

5.4 Entity Struct Reference

Contains an ID and std::bitset which a user can register components for using the ComponentManager class.

```
#include <ECSBase.h>
```

Public Attributes

- int id
- std::bitset< MAXCOMPONENTS > componentMask

5.4.1 Detailed Description

Contains an ID and std::bitset which a user can register components for using the ComponentManager class.

5.4.2 Member Data Documentation

5.4.2.1 componentMask

```
std::bitset<MAXCOMPONENTS> Entity::componentMask
```

Describes what type of components that the instance of the Entity has registered

5.4.2.2 id

```
int Entity::id
```

Unique ID mapped to a component within the ComponentManager class that the component was registered for using ComponentManager::RegisterComponent(Entity& _entity, const T& _initValue)

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

- Test/ECSBase.h

5.5 EntityManager Class Reference

Used to generate new Entity objects.

```
#include <ECSBase.h>
```

Public Member Functions

- EntityManager (unsigned int _maxEntities)
- Entity CreateEntity ()
- void RemoveEntity (Entity &_entity)
- void Expand (unsigned int _amount)

5.5.1 Detailed Description

Used to generate new Entity objects.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 EntityManager()

```
EntityManager::EntityManager (
            unsigned int _maxEntities )  [inline]
```

Initiates free Entity object IDs.

Parameters

_maxEntities	Specifies how many Entity IDs that will be generated.

5.5.3 Member Function Documentation

5.5.3.1 CreateEntity()

```
Entity EntityManager::CreateEntity ( )  [inline]
```

Create and returns a new unique Entity object.

Returns

Copy of the newly created Entity object

5.5.3.2 Expand()

```
void EntityManager::Expand (
            unsigned int _amount )  [inline]
```

Generates new IDs to give new Entity objects returned by EntityManager::CreateEntity().

Parameters

_amount	How many IDs to generate

Returns

void

5.5.3.3 RemoveEntity()

```
void EntityManager::RemoveEntity (
            Entity & _entity )  [inline]
```

Recycles the Entity by resetting the Entity::componentMask and enabling the Entity::id to be reused using the EntityManager::CreateEntity() method.

Parameters

_entity	The Entity to recycle

Returns

void

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

- Test/ECSBase.h

Chapter 6

File Documentation

6.1 ECSBase.h

```
1 #pragma once
2 #include <unordered_map>
3 #include <map>
4 #include <bitset>
5 #include <type_traits>
6 #include <queue>
7
8 // COMPONENTS
12 static const int MAXCOMPONENTS = 10;
13
17 namespace COMPONENTENUM
18 {
19     enum COMPONENTBITID { /*COMP1, COMP2, COMP3...*/ COMP};
20 }
21
22 // ---------------  SOURCE CODE EXAMPLES    ----------------------
23 //struct Comp1
24 //{
25 //  int age;
26 //  std::string name;
27 //  bool male;
28 //};
29 //
31 //struct Comp2
32 //{
33 //  std::string x;
34 //};
35 //
37 //struct Comp3
38 //{
39 //  int x;
40 //};
41 // ----------------------------------------------------------------
42
45 struct Entity
46 {
47     int id;
48     std::bitset<MAXCOMPONENTS> componentMask;
49 };
50
52 template<typename T>
53 struct BiDirectionalMap
54 {
55     std::map<int, int>idToIndex;
56     std::map<int, int>indexToId;
57     std::vector<T>objects;
58
59     BiDirectionalMap() = default;
60
61     void Insert(const Entity& _entity, const T& _value);
62
63     void Remove(const Entity& _entity);
64
65     void Clear();
66
67     T* GetObjectByID(int _id);
68 };
69
```

```
72 class ECSystem
73 {
74 private:
75     BiDirectionalMap<Entity>entityIDMap;
76 public:
77
78     std::bitset<MAXCOMPONENTS> componentMask;
80     ECSystem() = default;
81
86     void Bind(const Entity& _entity)
87     {
88         // Note - CHANGE THIS TO BITWISE OPERATOR...  HOW TO DO THAT WHEN U WANNA CHECK FOR PATTERN?
89         for (int i = 0; i < MAXCOMPONENTS; ++i)
90         {
91             if (componentMask.test(i))
92             {
93                 if (!_entity.componentMask.test(i)) // No binding since Entity doesn't have that
    component
94                 {
95                     printf("No such component registered!\n");
96                     return;
97                 }
98             }
99         }
100
101         entityIDMap.Insert(_entity, _entity);
102
103         return;
104     }
105
113     void BindFast(const Entity& _entity)
114     {
115         entityIDMap.Insert(_entity, _entity);
116     }
117
122     void UnBind(const Entity& _entity)
123     {
124         entityIDMap.Remove(_entity);
125     }
126
130     void RemoveEntities()
131     {
132         entityIDMap.Clear();
133     }
134
137     virtual void Update() = 0;
138 };
139
140 // ---------------  SOURCE CODE EXAMPLES    ----------------------
141 //class TestSystem :  public ECSystem
142 //{
143 //public:
144 //  TestSystem()
145 //      :ECSystem()
146 //  {
147 //      // Note - CHANGE THIS TO BITWISE OPERATOR
148 //      componentMask.set(0, false);
149 //      componentMask.set(1, true);
150 //      componentMask.set(2, true);
151 //  }
152 //
153 //  // Inherited via ECSystem
154 //  virtual void Update() override;
155 //
156 //};
157     // ------------------------------------------------------------
158
168 class ComponentManager
169 {
170 private:
171     // ---------------  SOURCE CODE EXAMPLES    ----------------------
172     // Add custom component bidirectional maps here
173     //BiDirectionalMap<Comp1>comp1Map;
174     //BiDirectionalMap<Comp2>comp2Map;
175     //BiDirectionalMap<Comp3>comp3Map;
176     // ------------------------------------------------------------------
177
178
179 public:
180     ComponentManager() = default;
186     template <typename T>
187     T* RegisterComponent(Entity& _entity, const T& _initValue);
188
197     template <typename T>
198     void RemoveComponent(Entity& _entity);
199
207     template <typename T>
```

```
208     T* GetComponent(const Entity& _entity);
209
210
221     template <typename T>
222     T* GetComponentFast(const Entity& _entity);
223 };
224
227 class EntityManager
228 {
229 private:
230     unsigned int maxEntities;
231     std::queue<int>freeIDs;
232 public:
236     EntityManager(unsigned int _maxEntities)
237     {
238         maxEntities = _maxEntities;
239
240         // Note - Expensive as fuck...
241         for (int i = 0; i < _maxEntities; ++i)
242         {
243             freeIDs.emplace(i);
244         }
245     }
246
250     Entity CreateEntity()
251     {
252         Entity entity;
253         if (freeIDs.empty())
254         {
255             entity.id = -1;
256             return entity;
257         }
258         entity.id = freeIDs.front();
259         freeIDs.pop();
260         return entity;
261     }
262
267     void RemoveEntity(Entity& _entity)
268     {
269         freeIDs.push(_entity.id);
270         _entity.id = -1;
271         _entity.componentMask.reset();
272     }
273
278     void Expand(unsigned int _amount)
279     {
280         int lastID = maxEntities;
281         maxEntities += _amount;
282         for (int i = 0; i < _amount; ++i)
283         {
284             freeIDs.emplace(lastID + i);
285         }
286     }
287 };
288
297 class ECS
298 {
299 private:
300     ComponentManager componentManager;
301     EntityManager entityManager;
302 public:
306     ECS(unsigned int _maxEntities)
307         :entityManager(_maxEntities)
308     {
309
310     }
311
312     ~ECS() = default;
313
317     EntityManager& GetEntityManager() { return entityManager; }
318
322     ComponentManager& GetComponentManager() { return componentManager; }
323
328     void ObliterateEntity(Entity& _entity)
329     {
330
331         // Call for each Component
332         //componentManager.RemoveComponent<Comp1>(_entity); // Works regardless if it's bound or not
333         //componentManager.RemoveComponent<Comp2>(_entity); // Works regardless if it's bound or not
334         //componentManager.RemoveComponent<Comp3>(_entity); // Works regardless if it's bound or not
335
336         // Call for each ECSystem
337         //tSystem.UnBind(_entity); // Works regardless if it's bound or not
338
339         entityManager.RemoveEntity(_entity);
340     }
341
```

```
342     // Systems
343     //TestSystem tSystem; // Example of adding a custom ECSystem
344 };
345 
346 template<typename T>
347 inline void BiDirectionalMap<T>::Insert(const Entity& _entity, const T& _value)
348 {
349     int insertIndex = objects.size();
350     objects.emplace_back(_value);
351     idToIndex.emplace(_entity.id, insertIndex);
352     indexToId.emplace(insertIndex, _entity.id);
353 }
354 
355 template<typename T>
356 inline void BiDirectionalMap<T>::Remove(const Entity& _entity)
357 {
358     if (idToIndex.count(_entity.id) == 0)
359         return;
360     int indexToRemove = idToIndex.at(_entity.id);     // Get index of the entity to remove
361     int lastIndex = objects.size() - 1;              // Get the last index in the array (to avoid
    multiple .size() calls)
362 
363     objects[indexToRemove] = objects[lastIndex];     // Replace element to remove with the last element
364     objects.resize(lastIndex);                       // Resize array to fit new number of elements
365     int tempId = indexToId.at(lastIndex);            // Get id for the element at the last index
366     idToIndex.at(tempId) = indexToRemove;            // ID of last element remapped to the same elements
    new index
367     indexToId.at(indexToRemove) = tempId;            // Index of the element that was moved remapped to
    match the id of the element moved
368     idToIndex.erase(_entity.id);                       // Remove the id->index pair since that id wont be
    mapped to anything anymore
369     indexToId.erase(lastIndex);
370 }
371 
372 template<typename T>
373 inline void BiDirectionalMap<T>::Clear()
374 {
375     idToIndex.clear();
376     indexToId.clear();
377     objects.clear();
378     objects.resize(0);
379 }
380 
381 template<typename T>
382 inline T* BiDirectionalMap<T>::GetObjectByID(int _id)
383 {
384     return &objects[idToIndex.at(_id)];
385 }
386 
387 template<typename T>
388 inline T* ComponentManager::RegisterComponent(Entity& _entity, const T& _initValue)
389 {
390     // ---------------  SOURCE CODE EXAMPLES    ----------------------
391     //if constexpr (std::is_same_v<T, Comp1>)
392     //{
393     //  if (!_entity.componentMask.test(COMPONENTENUM::COMP1))
394     //  {
395     //      _entity.componentMask.set(COMPONENTENUM::COMP1);
396     //      comp1Map.Insert(_entity, _initValue);
397     //      return comp1Map.GetObjectByID(_entity.id);
398     //  }
399     //}
400     //else if constexpr (std::is_same_v<T, Comp2>)
401     //{
402     //  if (!_entity.componentMask.test(COMPONENTENUM::COMP2))
403     //  {
404     //      _entity.componentMask.set(COMPONENTENUM::COMP2);
405     //      comp2Map.Insert(_entity, _initValue);
406     //      return comp2Map.GetObjectByID(_entity.id);
407     //  }
408     //}
409     //else if constexpr (std::is_same_v<T, Comp3>)
410     //{
411     //  if (!_entity.componentMask.test(COMPONENTENUM::COMP3))
412     //  {
413     //      _entity.componentMask.set(COMPONENTENUM::COMP3);
414     //      comp3Map.Insert(_entity, _initValue);
415     //      return comp3Map.GetObjectByID(_entity.id);
416     //  }
417     //}
418     // ---------------------------------------------------------------
419 
420 
421     return nullptr;
422 }
423 
424 template<typename T>
```

```
425 inline void ComponentManager::RemoveComponent(Entity& _entity)
426 {
427     // ---------------  SOURCE CODE EXAMPLES    ----------------------
428     //if constexpr (std::is_same_v<T, Comp1>)
429     //{
430     //  if (_entity.componentMask.test(COMPONENTENUM::COMP1))
431     //  {
432     //      _entity.componentMask.set(COMPONENTENUM::COMP1, false);
433     //      comp1Map.Remove(_entity);
434     //  }
435     //}
436     //else if constexpr (std::is_same_v<T, Comp2>)
437     //{
438     //  if (_entity.componentMask.test(COMPONENTENUM::COMP2))
439     //  {
440     //      _entity.componentMask.set(COMPONENTENUM::COMP2, false);
441     //      comp2Map.Remove(_entity);
442     //  }
443     //}
444     //else if constexpr (std::is_same_v<T, Comp3>)
445     //{
446     //  if (_entity.componentMask.test(COMPONENTENUM::COMP3))
447     //  {
448     //      _entity.componentMask.set(COMPONENTENUM::COMP3, false);
449     //      comp3Map.Remove(_entity);
450     //  }
451     //}
452     // ------------------------------------------------------------
453 }
454
455 template<typename T>
456 inline T* ComponentManager::GetComponent(const Entity& _entity)
457 {
458     // ---------------  SOURCE CODE EXAMPLES    ----------------------
459     //if constexpr (std::is_same_v<T, Comp1>)
460     //{
461     //  if (_entity.componentMask.test(COMPONENTENUM::COMP1))
462     //      return &comp1Map.objects[comp1Map.idToIndex.at(_entity.id)];
463     //  else
464     //      return nullptr;
465     //}
466     //else if constexpr (std::is_same_v<T, Comp2>)
467     //{
468     //  if (_entity.componentMask.test(COMPONENTENUM::COMP2))
469     //      return &comp2Map.objects[comp2Map.idToIndex.at(_entity.id)];
470     //  else
471     //      return nullptr;
472     //}
473     //else if constexpr (std::is_same_v<T, Comp3>)
474     //{
475     //  if (_entity.componentMask.test(COMPONENTENUM::COMP3))
476     //      return &comp3Map.objects[comp3Map.idToIndex.at(_entity.id)];
477     //  else
478     //      return nullptr;
479     //}
480     // ------------------------------------------------------------
481
482
483     return nullptr;
484 }
485
486 template<typename T>
487 inline T* ComponentManager::GetComponentFast(const Entity& _entity)
488 {
489     // ---------------  SOURCE CODE EXAMPLES    ----------------------
490     //if constexpr (std::is_same_v<T, Comp1>)
491     //{
492     //  return &comp1Map.objects[comp1Map.idToIndex.at(_entity.id)];
493     //}
494     //else if constexpr (std::is_same_v<T, Comp2>)
495     //{
496     //  return &comp2Map.objects[comp2Map.idToIndex.at(_entity.id)];
497     //}
498     //else if constexpr (std::is_same_v<T, Comp3>)
499     //{
500     //  return &comp3Map.objects[comp3Map.idToIndex.at(_entity.id)];
501     //}
502     // ------------------------------------------------------------
503
504
505     return nullptr;
506 }
```

Index