Scars_Healing

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 gtest_lite Namespace Reference

gtest_lite: a keretrendszer függvényinek és objektumainak névtere

Classes

- struct Test

Functions

- template<typename T1 , typename T2 >
  std::ostream & EXPECT_ (T1 exp, T2 act, bool(*pred)(T1, T2), const char *file, int line, const char *expr, const char *lhs="elvart", const char *rhs="aktual")

  általános sablon a várt értékhez.
- template<typename T1 , typename T2 >
  std::ostream & EXPECT_ (T1 *exp, T2 *act, bool(*pred)(T1 *, T2 *), const char *file, int line, const char *expr, const char *lhs="elvart", const char *rhs="aktual")

  pointerre specializált sablon a várt értékhez.
- std::ostream & EXPECTSTR (const char *exp, const char *act, bool(*pred)(const char *, const char *), const char *file, int line, const char *expr, const char *lhs="elvart", const char *rhs="aktual")
- template<typename T1 , typename T2 >
  bool eq (T1 a, T2 b)
- bool eqstr (const char *a, const char *b)
- template<typename T1 , typename T2 >
  bool ne (T1 a, T2 b)
- bool nestr (const char *a, const char *b)
- template<typename T1 , typename T2 >
  bool le (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool lt (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool ge (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool gt (T1 a, T2 b)
- template<typename T >
  bool almostEQ (T a, T b)

4.1.1 Detailed Description

gtest_lite: a keretrendszer függvényinek és objektumainak névtere

4.1.2 Function Documentation

4.1.2.1 almostEQ()

```
template<typename T >
bool gtest_lite::almostEQ (
            T a,
            T b )
```

Segédsablon valós számok összehasonlításához Nem bombabiztos, de nekünk most jó lesz Elméleti hátér: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm

4.1.2.2 eq()

```
template<typename T1 , typename T2 >
bool gtest_lite::eq (
            T1 a,
            T2 b )
```

segéd sablonok a relációkhoz. azért nem STL (algorithm), mert csak a függvény lehet, hogy menjen a deduckció

4.1.2.3 eqstr()

```
bool gtest_lite::eqstr (
            const char * a,
            const char * b )  [inline]
```

4.1.2.4 EXPECT_() [1/2]

```
template<typename T1 , typename T2 >
std::ostream& gtest_lite::EXPECT_ (
            T1 * exp,
            T2 * act,
            bool(*)(T1 *, T2 *) pred,
            const char * file,
            int line,
            const char * expr,
            const char * lhs = "elvart",
            const char * rhs = "aktual" )
```

pointerre specializált sablon a várt értékhez.

Here is the call graph for this function:

4.1.2.5 EXPECT_() [2/2]

```
template<typename T1 , typename T2 >
std::ostream& gtest_lite::EXPECT_ (
            T1 exp,
            T2 act,
            bool(*)(T1, T2) pred,
            const char * file,
            int line,
            const char * expr,
            const char * lhs = "elvart",
            const char * rhs = "aktual" )
```

általános sablon a várt értékhez.

Here is the call graph for this function:

4.1.2.6 EXPECTSTR()

```
std::ostream& gtest_lite::EXPECTSTR (
            const char * exp,
            const char * act,
            bool(*)(const char *, const char *) pred,
            const char * file,
            int line,
            const char * expr,
```

```
            const char * lhs = "elvart",
            const char * rhs = "aktual" )  [inline]
```

stringek összehasonlításához. azért nem spec. mert a sima EQ-ra másként kell működnie. Here is the call graph for this function:

4.1.2.7 ge()

```
template<typename T1 , typename T2 >
bool gtest_lite::ge (
            T1 a,
            T2 b )
```

4.1.2.8 gt()

```
template<typename T1 , typename T2 >
bool gtest_lite::gt (
            T1 a,
            T2 b )
```

4.1.2.9 le()

```
template<typename T1 , typename T2 >
bool gtest_lite::le (
            T1 a,
            T2 b )
```

4.1.2.10 lt()

```
template<typename T1 , typename T2 >
bool gtest_lite::lt (
            T1 a,
            T2 b )
```

4.1.2.11 ne()

```
template<typename T1 , typename T2 >
bool gtest_lite::ne (
            T1 a,
            T2 b )
```

4.1.2.12 nestr()

```
bool gtest_lite::nestr (
            const char * a,
            const char * b )  [inline]
```

Chapter 5

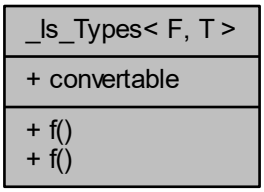
Class Documentation

5.1 _Is_Types< F, T > Struct Template Reference

Segédsablon típuskonverzió futás közbeni ellenőrzésere.

`#include <gtest_lite.h>`

Collaboration diagram for _Is_Types< F, T >:



Static Public Member Functions

- template<typename D >
  static char(& f (D))[1]
- template<typename D >
  static char(& f (...))[2]

Static Public Attributes

- static const bool convertable = sizeof(f<T>(F())) == 1

5.1.1 Detailed Description

template<typename F, typename T>
struct _Is_Types< F, T >

Segédsablon típuskonverzió futás közbeni ellenőrzésere.

5.1.2 Member Function Documentation

5.1.2.1 f() [1/2]

```
template<typename F , typename T >
template<typename D >
static char(& _Is_Types< F, T >::f (
              ... ))[2]  [static]
```

5.1.2.2 f() [2/2]

```
template<typename F , typename T >
template<typename D >
static char(& _Is_Types< F, T >::f (
             D ))[1]  [static]
```

5.1.3 Member Data Documentation

5.1.3.1 convertable

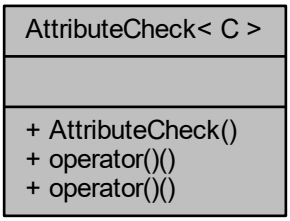
```
template<typename F , typename T >
const bool _Is_Types< F, T >::convertable = sizeof(f<T>(F())) == 1  [static]
```

5.2 AttributeCheck< C > Class Template Reference

Predicatum, which determines if an item in an array has a specific attribute.

`#include <Item.h>`

Collaboration diagram for AttributeCheck< C >:



Public Member Functions

- AttributeCheck (C Attribute)
  Default Constructor.
- bool operator() (const C &c)
  Decides if the attribute is equal.
- bool operator() (const Item &c)
  Checks if the type of an Item is something.

5.2.1 Detailed Description

template<class C>
class AttributeCheck< C >

Predicatum, which determines if an item in an array has a specific attribute.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AttributeCheck()

```
template<class C >
AttributeCheck< C >::AttributeCheck (
            C Attribute )  [inline]
```

Default Constructor.

5.2.3 Member Function Documentation

5.2.3.1 operator()() [1/2]

```
template<class C >
bool AttributeCheck< C >::operator() (
            const C & c )  [inline]
```

Decides if the attribute is equal.

5.2.3.2 operator()() [2/2]

```
template<class C >
bool AttributeCheck< C >::operator() (
            const Item & c )  [inline]
```

Checks if the type of an Item is something.

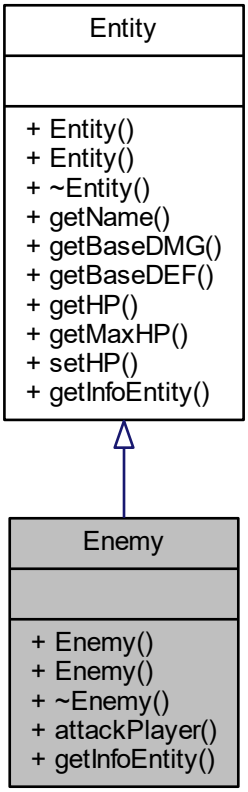
Here is the call graph for this function:

5.3 Enemy Class Reference

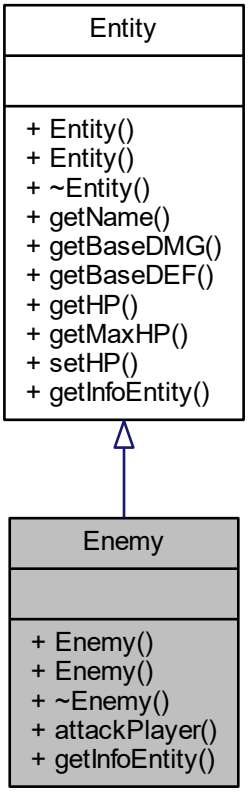
Enemy can attack a Player without the Player damaging it, every Enemy will have a Level, and will have a chance to drop an Item if it dies according to its Level.

```
#include <Enemy.h>
```

Inheritance diagram for Enemy:



Collaboration diagram for Enemy:



Public Member Functions

- Enemy (const char ∗Name, double HP=100, double MaxHP=100, double BaseDMG=5, double BaseDEF=5)

  Constructor.
- Enemy (double HP=100, double MaxHP=100, double BaseDMG=5, double BaseDEF=5)

  Default Constructor.
- virtual ∼Enemy ()

  Destructor.
- void attackPlayer (Player &PLYR, Item WS=Item())

  Attack Player without taking Damage from it.
- std::ostream & getInfoEntity (InfoPreset Preset=InfoPreset(), std::ostream &os=std::cout)

  Displays the info of an Enemy, dynamically controlable with an InfoPreset.

5.3.1 Detailed Description

Enemy can attack a Player without the Player damaging it, every Enemy will have a Level, and will have a chance to drop an Item if it dies according to its Level.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Enemy() [1/2]

```
Enemy::Enemy (
            const char * Name,
            double HP = 100,
            double MaxHP = 100,
            double BaseDMG = 5,
            double BaseDEF = 5 )
```

Constructor.

5.3.2.2 Enemy() [2/2]

```
Enemy::Enemy (
            double HP = 100,
            double MaxHP = 100,
            double BaseDMG = 5,
            double BaseDEF = 5 )
```

Default Constructor.

5.3.2.3 ∼Enemy()

```
Enemy::∼Enemy ( )  [virtual]
```

Destructor.

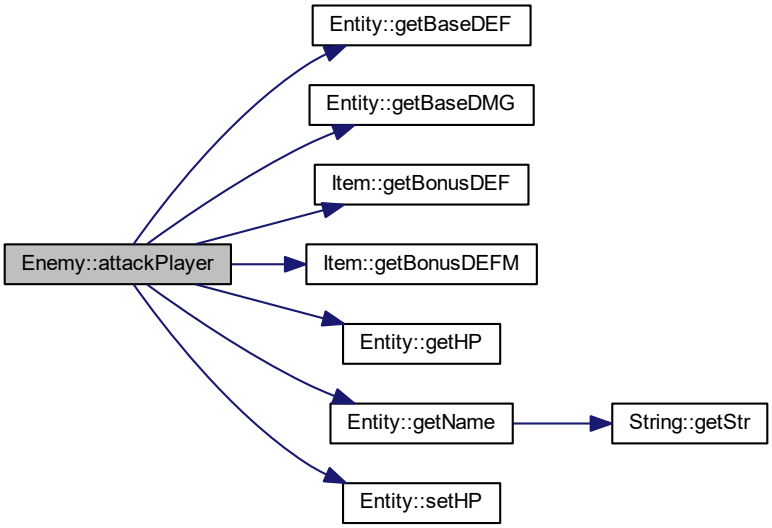
5.3.3 Member Function Documentation

5.3.3.1 attackPlayer()

```
void Enemy::attackPlayer (
            Player & PLYR,
            Item WS = Item() )
```

Attack Player without taking Damage from it.

Here is the call graph for this function:



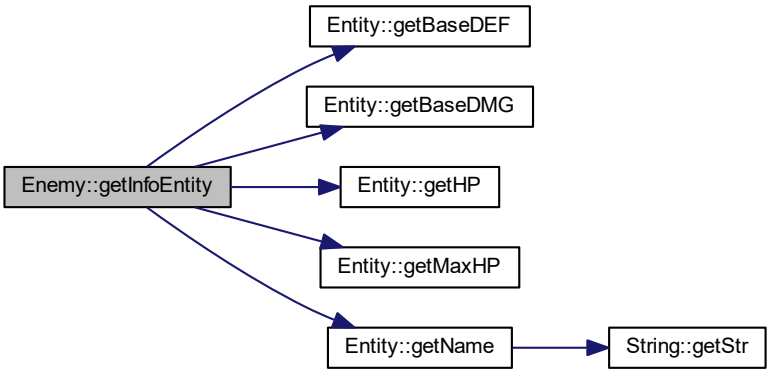
5.3.3.2 getInfoEntity()

```
std::ostream & Enemy::getInfoEntity (
            InfoPreset Preset = InfoPreset(),
            std::ostream & os = std::cout )  [virtual]
```

Displays the info of an Enemy, dynamically controlable with an InfoPreset.

Implements Entity.

Here is the call graph for this function:

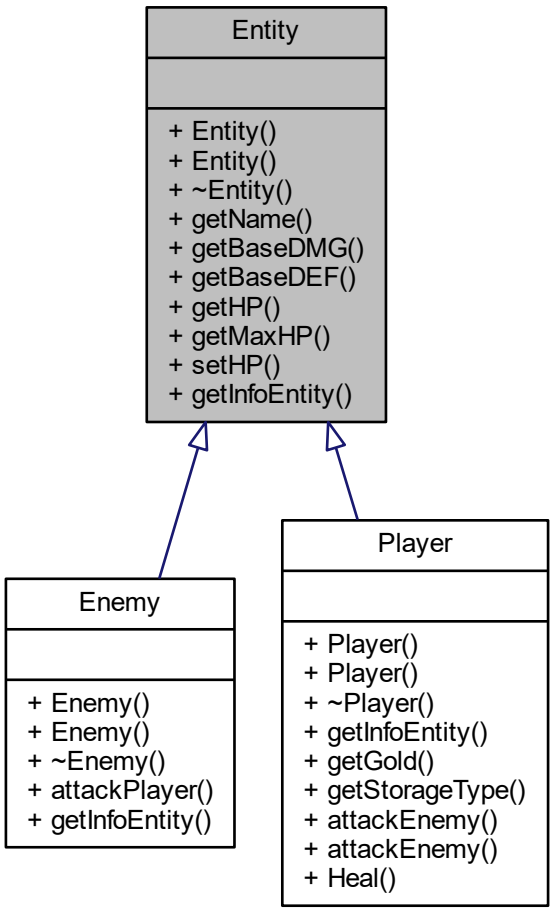


5.4 Entity Class Reference

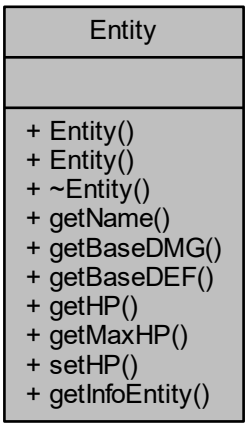
An alive Entity can be a Player or an Enemy (for now). It has HP, a Name and basic attributes, like BaseDMG and BaseDEF.

```
#include <Entity.h>
```

Inheritance diagram for Entity:



Collaboration diagram for Entity:



Public Member Functions

- Entity (const char ∗Name, double HP=100, double MaxHP=100, double BaseDMG=5, double BaseDEF=5)

  Constructor.
- Entity (double HP=100, double MaxHP=100, double BaseDMG=5, double BaseDEF=5)

  Default Constructor.
- virtual ∼Entity ()

  Destructor.
- const char ∗ getName ()

  Gets Entity's name.
- double getBaseDMG ()

  Gets Entity's Base Damage.
- double getBaseDEF ()

  Gets Entity's Base Defense.
- double getHP ()

  Gets Entity's HP.
- double getMaxHP ()

  Gets Entity's MaxHP.
- void setHP (const double hp)

  Sets Entity's HP.
- virtual std::ostream & getInfoEntity (InfoPreset Preset=InfoPreset(), std::ostream &os=std::cout)=0

  Displays the info of an Entity, dynamically controlable with an InfoPreset.

5.4.1 Detailed Description

An alive Entity can be a Player or an Enemy (for now). It has HP, a Name and basic attributes, like BaseDMG and BaseDEF.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Entity() [1/2]

```
Entity::Entity (
            const char * Name,
            double HP = 100,
            double MaxHP = 100,
            double BaseDMG = 5,
            double BaseDEF = 5 )
```

Constructor.

5.4.2.2 Entity() [2/2]

```
Entity::Entity (
            double HP = 100,
            double MaxHP = 100,
            double BaseDMG = 5,
            double BaseDEF = 5 )
```

Default Constructor.

5.4.2.3 ~Entity()

```
Entity::~Entity ( )  [virtual]
```

Destructor.

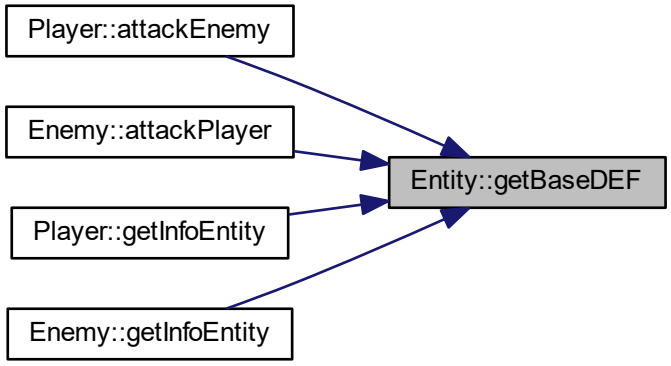
5.4.3 Member Function Documentation

5.4.3.1 getBaseDEF()

```
double Entity::getBaseDEF ( )
```

Gets Entity's Base Defense.

Here is the caller graph for this function:

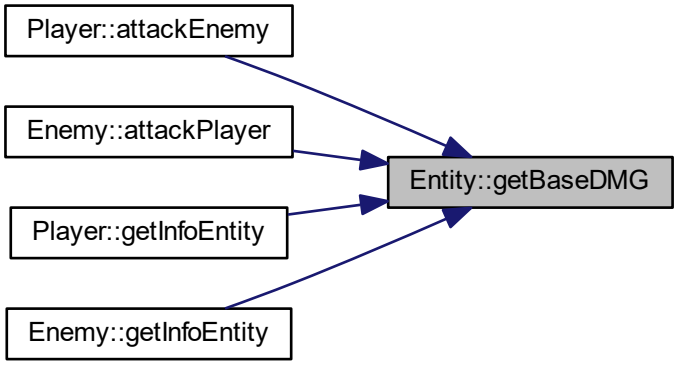


5.4.3.2 getBaseDMG()

```
double Entity::getBaseDMG ( )
```

Gets Entity's Base Damage.

Here is the caller graph for this function:

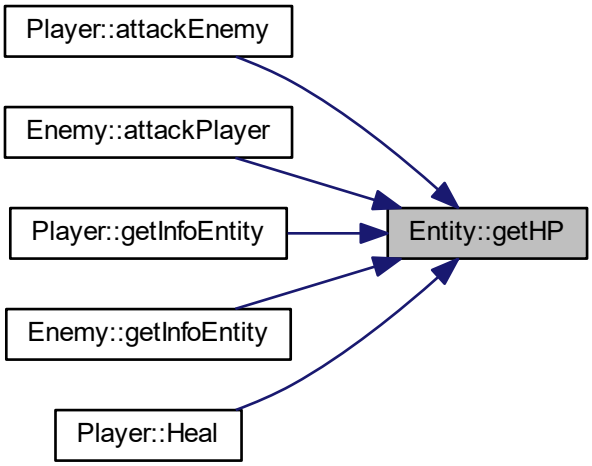


5.4.3.3 getHP()

```
double Entity::getHP ( )
```

Gets Entity's HP.

Here is the caller graph for this function:



5.4.3.4 getInfoEntity()

```
virtual std::ostream& Entity::getInfoEntity (
            InfoPreset Preset = InfoPreset(),
            std::ostream & os = std::cout )  [pure virtual]
```

Displays the info of an Entity, dynamically controlable with an InfoPreset.

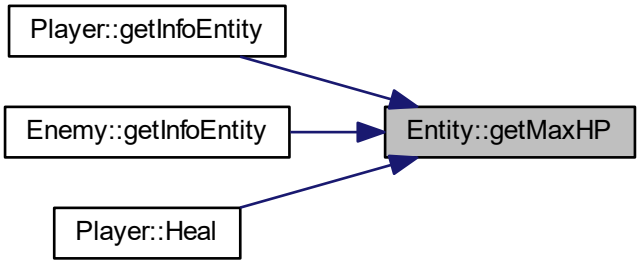
Implemented in Enemy, and Player.

5.4.3.5 getMaxHP()

```
double Entity::getMaxHP ( )
```

Gets Entity's MaxHP.

Here is the caller graph for this function:



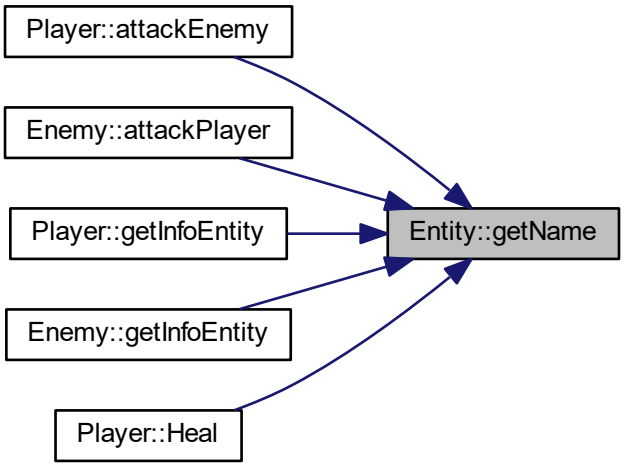
5.4.3.6 getName()

```
const char * Entity::getName ( )
```

Gets Entity's name.

Here is the call graph for this function:

Here is the caller graph for this function:

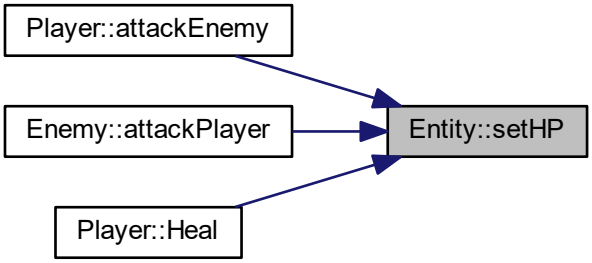


5.4.3.7 setHP()

```
void Entity::setHP (
            const double hp )
```

Sets Entity's HP.

Here is the caller graph for this function:

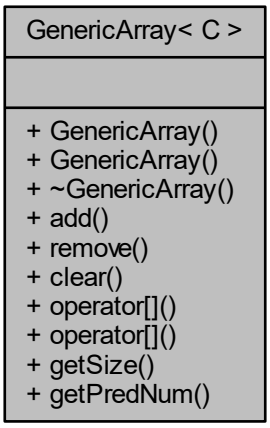


5.5 GenericArray< C > Class Template Reference

This GenericArray can store any type of things dynamically. Maybe I will do a static version as well. Or just implement it with a bool. Time will tell...

`#include <GenericArray.h>`

Collaboration diagram for GenericArray< C >:



Public Member Functions

- GenericArray ()
  Default Constructor.
- GenericArray (const GenericArray< C > &GA)
  Copy Constructor.
- virtual ~GenericArray ()
  Destructor.
- void add (const C &Element)
  Adding an Element to the Array.
- void remove (const size_t &Index)
  Removes an Element from the Array.
- void clear ()
  Clears (deletes and then reallocate) the Array.
- C & operator[ ] (size_t Index) const
  Accessing Data as constant.
- C & operator[ ] (size_t Index)
  Accessing Data.
- const size_t & getSize ()
  Get Size of the Array.
- template<class Pred >
  size_t getPredNum (Pred pred)
  Get Number of specific Items in the Array.

5.5.1 Detailed Description

template<class C>
class GenericArray< C >

This GenericArray can store any type of things dynamically. Maybe I will do a static version as well. Or just implement it with a bool. Time will tell...

5.5.2 Constructor & Destructor Documentation

5.5.2.1 GenericArray() [1/2]

```
template<class C >
GenericArray< C >::GenericArray ( )  [inline]
```

Default Constructor.

5.5.2.2 GenericArray() [2/2]

```
template<class C >
GenericArray< C >::GenericArray (
            const GenericArray< C > & GA )  [inline]
```

Copy Constructor.

5.5.2.3 ~GenericArray()

```
template<class C >
virtual GenericArray< C >::~GenericArray ( )  [inline], [virtual]
```

Destructor.

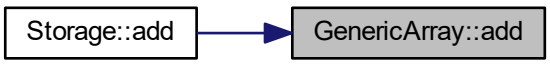
5.5.3 Member Function Documentation

5.5.3.1 add()

```
template<class C >
void GenericArray< C >::add (
            const C & Element )  [inline]
```

Adding an Element to the Array.

Here is the caller graph for this function:

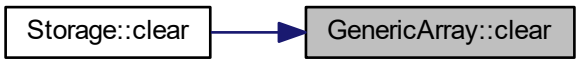


5.5.3.2 clear()

```
template<class C >
void GenericArray< C >::clear ( )  [inline]
```

Clears (deletes and then reallocate) the Array.

Here is the caller graph for this function:



5.5.3.3 getPredNum()

```
template<class C >
template<class Pred >
size_t GenericArray< C >::getPredNum (
            Pred pred )  [inline]
```

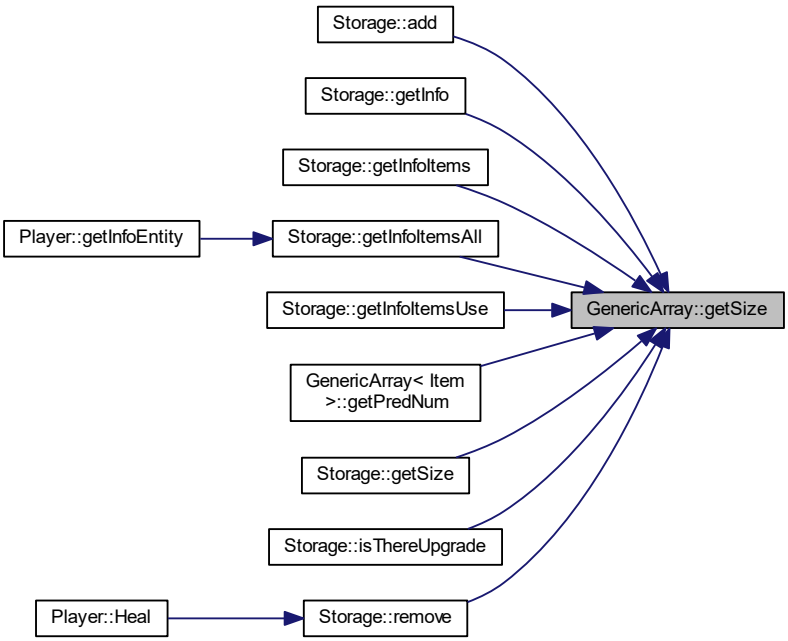
Get Number of specific Items in the Array.

5.5.3.4 getSize()

```
template<class C >
const size_t& GenericArray< C >::getSize ( )  [inline]
```

Get Size of the Array.

Here is the caller graph for this function:



5.5.3.5 operator[]() [1/2]

```
template<class C >
C& GenericArray< C >::operator[] (
            size_t Index )  [inline]
```

Accessing Data.

5.5.3.6 operator[]() [2/2]

```
template<class C >
C& GenericArray< C >::operator[] (
            size_t Index ) const  [inline]
```

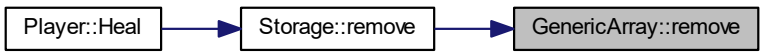
Accessing Data as constant.

5.5.3.7 remove()

```
template<class C >
void GenericArray< C >::remove (
            const size_t & Index )  [inline]
```

Removes an Element from the Array.

Here is the caller graph for this function:

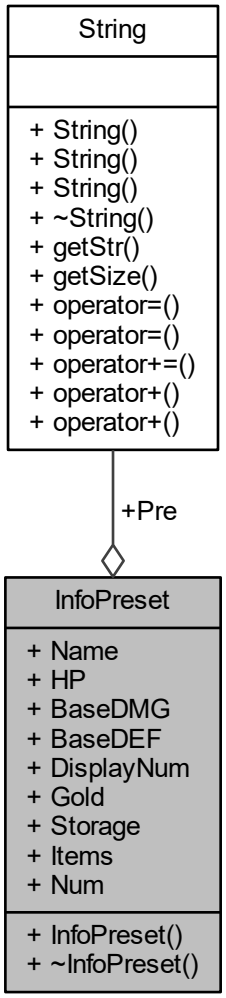


5.6 InfoPreset Class Reference

A preset tool for displaying dynamic information about anything.

```
#include <InfoPreset.h>
```

Collaboration diagram for InfoPreset:



Public Member Functions

- InfoPreset (const char *cPre="", bool Name=true, bool HP=true, bool BaseDMG=true, bool BaseDEF=true, bool DisplayNum=false, bool Gold=true, bool Storage=true, bool Items=true, const size_t &dNum=0)

  Constructor.
- virtual ~InfoPreset ()

  Destructor.

Public Attributes

- bool Name

  Displays Name.
- bool HP

Displays HP of an Entity or Healing for a Potion.

- bool BaseDMG

  Displays BonusDMG and BonusDMGM of a Weapon/Shield or BaseDMG for an Entity.
- bool BaseDEF

  Displays BonusDEF and BonusDEFM of a Weapon/Shield or BaseDEF for an Entity.
- bool DisplayNum

  Displays a given number, like the index of an Enemy before its Name.
- String Pre

  A String, which is displayed before everything, it can give some context.
- bool Gold

  Displays the Player's Gold.
- bool Storage

  Displays the Player's Storage.
- bool Items

  Displays the Player's Storage's Items.
- size_t Num

  If DisplayNum is TRUE, then it displays this Number.

5.6.1 Detailed Description

A preset tool for displaying dynamic information about anything.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 InfoPreset()

```
InfoPreset::InfoPreset (
            const char * cPre = "",
            bool Name = true,
            bool HP = true,
            bool BaseDMG = true,
            bool BaseDEF = true,
            bool DisplayNum = false,
            bool Gold = true,
            bool Storage = true,
            bool Items = true,
            const size_t & dNum = 0 )  [inline]
```

Constructor.

5.6.2.2 ~InfoPreset()

```
virtual InfoPreset::~InfoPreset ( )  [inline], [virtual]
```

Destructor.

5.6.3 Member Data Documentation

5.6.3.1 BaseDEF

```
bool InfoPreset::BaseDEF
```

Displays BonusDEF and BonusDEFM of a Weapon/Shield or BaseDEF for an Entity.

5.6.3.2 BaseDMG

```
bool InfoPreset::BaseDMG
```

Displays BonusDMG and BonusDMGM of a Weapon/Shield or BaseDMG for an Entity.

5.6.3.3 DisplayNum

```
bool InfoPreset::DisplayNum
```

Displays a given number, like the index of an Enemy before its Name.

5.6.3.4 Gold

```
bool InfoPreset::Gold
```

Displays the Player's Gold.

5.6.3.5 HP

```
bool InfoPreset::HP
```

Displays HP of an Entity or Healing for a Potion.

5.6.3.6 Items

```
bool InfoPreset::Items
```

Displays the Player's Storage's Items.

5.6.3.7 Name

```
bool InfoPreset::Name
```

Displays Name.

5.6.3.8 Num

```
size_t InfoPreset::Num
```

If DisplayNum is TRUE, then it displays this Number.

5.6.3.9 Pre

```
String InfoPreset::Pre
```

A String, which is displayed before everything, it can give some context.

5.6.3.10 Storage

```
bool InfoPreset::Storage
```

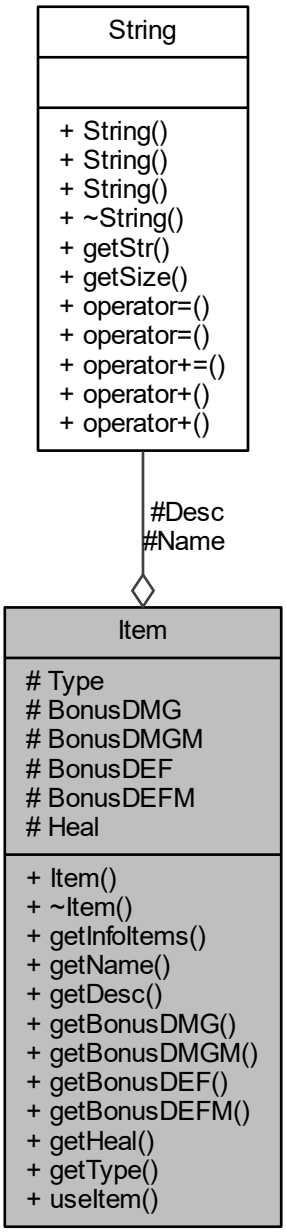
Displays the Player's Storage.

5.7 Item Class Reference

An Item can be stored in a storage and also has a type. Depending on the type, an Item can heal the player or damage an enemy, etc.

`#include <Item.h>`

Collaboration diagram for Item:



Public Member Functions

- Item (const ItemType Type=ItemType::WeaponShield, const char *Name="DisItem", const char *Desc="This is an Item.", const double &BonusDMG=0, const double &BonusDMGM=1, const double &BonusDEF=0, const double &BonusDEFM=1, const double &Heal=0)

  Default Constructor.
- virtual ~Item ()

  Destructor.
- virtual std::ostream & getInfoItems (InfoPreset Preset=InfoPreset(), std::ostream &os=std::cout)

  Writes out the Item's info to an output.
- const char * getName ()

  Gets Name.
- const char * getDesc ()

  Gets Description.
- double getBonusDMG ()

  Gets Additive Bonus Damage of Item.
- double getBonusDMGM ()

  Gets Multiplied Bonus Damage of Item.
- double getBonusDEF ()

  Gets Additive Bonus Defense of Item.
- double getBonusDEFM ()

  Gets Multiplied Bonus Defense of Item.
- double getHeal ()

  Gets Healing of Item.
- ItemType getType () const

  Gets Type of Item.
- UseCases useItem ()

  Use an Item.

Protected Attributes

- ItemType Type

  Type of the Item.
- String Name

  Name of the Item.
- String Desc

  Description of the Item.
- double BonusDMG

  Additive Bonus Damage of Item.
- double BonusDMGM

  Multiplied Bonus Damage of Item.
- double BonusDEF

  Additive Bonus Defense of Item.
- double BonusDEFM

  Multiplied Bonus Defense of Item.
- double Heal

  Healing of Item.

5.7.1 Detailed Description

An Item can be stored in a storage and also has a type. Depending on the type, an Item can heal the player or damage an enemy, etc.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Item()

```
Item::Item (
            const ItemType Type = ItemType::WeaponShield,
            const char * Name = "DisItem",
            const char * Desc = "This is an Item.",
            const double & BonusDMG = 0,
            const double & BonusDMGM = 1,
            const double & BonusDEF = 0,
            const double & BonusDEFM = 1,
            const double & Heal = 0 )
```

Default Constructor.

5.7.2.2 ~Item()

```
Item::~Item ( )  [virtual]
```

Destructor.

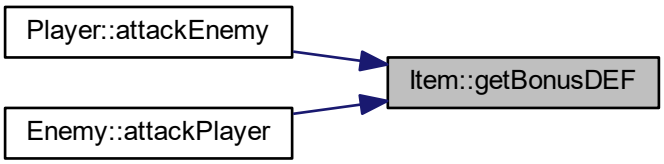
5.7.3 Member Function Documentation

5.7.3.1 getBonusDEF()

```
double Item::getBonusDEF ( )
```

Gets Additive Bonus Defense of Item.

Here is the caller graph for this function:

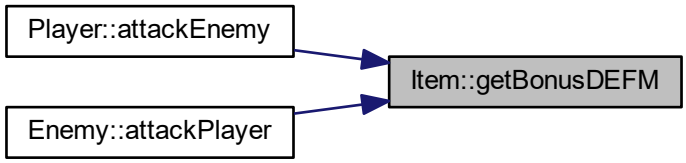


5.7.3.2 getBonusDEFM()

```
double Item::getBonusDEFM ( )
```

Gets Multiplied Bonus Defense of Item.

Here is the caller graph for this function:



5.7.3.3 getBonusDMG()

```
double Item::getBonusDMG ( )
```

Gets Additive Bonus Damage of Item.

Here is the caller graph for this function:

5.7.3.4 getBonusDMGM()

```
double Item::getBonusDMGM ( )
```

Gets Multiplied Bonus Damage of Item.

Here is the caller graph for this function:

5.7.3.5 getDesc()

```
const char * Item::getDesc ( )
```

Gets Description.

Here is the call graph for this function:

5.7.3.6 getHeal()

```
double Item::getHeal ( )
```

Gets Healing of Item.

5.7.3.7 getInfoItems()

```
std::ostream & Item::getInfoItems (
            InfoPreset Preset = InfoPreset(),
            std::ostream & os = std::cout )  [virtual]
```

Writes out the Item's info to an output.

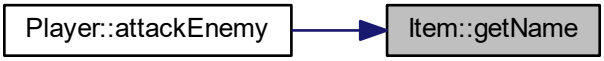
5.7.3.8 getName()

```
const char * Item::getName ( )
```

Gets Name.

Here is the call graph for this function:

Here is the caller graph for this function:

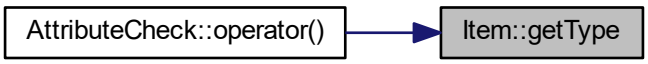


5.7.3.9 getType()

```
ItemType Item::getType ( ) const
```

Gets Type of Item.

Here is the caller graph for this function:



5.7.3.10 useItem()

```
UseCases Item::useItem ( )
```

Use an Item.

Use a Weapon/Shield

Use a Healing Potion

5.7.4 Member Data Documentation

5.7.4.1 BonusDEF

```
double Item::BonusDEF  [protected]
```

Additive Bonus Defense of Item.

5.7.4.2 BonusDEFM

```
double Item::BonusDEFM  [protected]
```

Multiplied Bonus Defense of Item.

5.7.4.3 BonusDMG

```
double Item::BonusDMG  [protected]
```

Additive Bonus Damage of Item.

5.7.4.4 BonusDMGM

```
double Item::BonusDMGM  [protected]
```

Multiplied Bonus Damage of Item.

5.7.4.5 Desc

```
String Item::Desc  [protected]
```

Description of the Item.

5.7.4.6 Heal

```
double Item::Heal  [protected]
```

Healing of Item.

5.7.4.7 Name

```
String Item::Name  [protected]
```

Name of the Item.

5.7.4.8 Type

```
ItemType Item::Type  [protected]
```

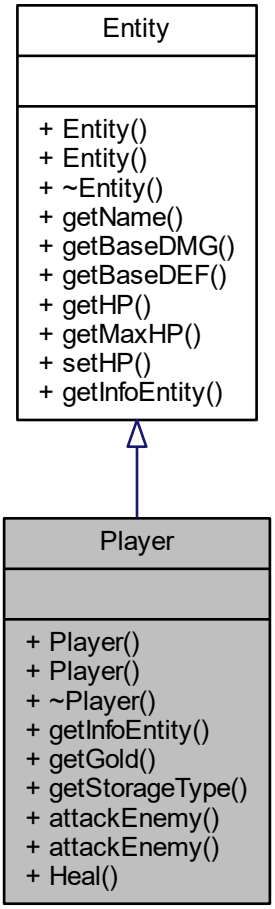
Type of the Item.

5.8 Player Class Reference

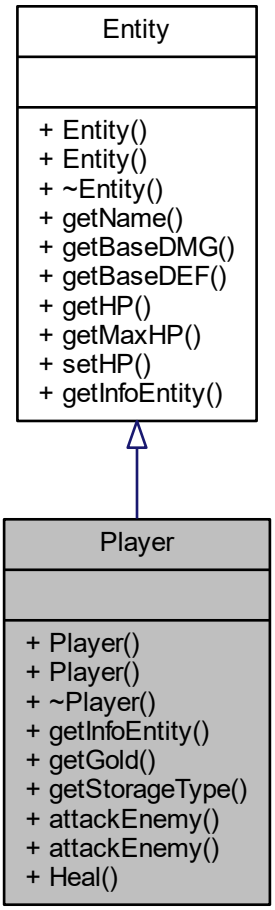
Player has Gold and a Storage in which it can hold some Item. The Player will be able to spend Gold in the Shop.

```
#include <Player.h>
```

Inheritance diagram for Player:



Collaboration diagram for Player:



Public Member Functions

- Player (const char *Name, double HP=100, double MaxHP=100, double BaseDMG=5, double BaseDEF=5, size_t Gold=0, Storage StorageType=Storage())

  Constructor.
- Player (double HP=100, double MaxHP=100, double BaseDMG=5, double BaseDEF=5, size_t Gold=0)

  Default Constructor.
- virtual ~Player ()

  Destructor.
- std::ostream & getInfoEntity (InfoPreset Preset=InfoPreset(), std::ostream &os=std::cout)

  Writes out the Player's info to an output.
- size_t getGold ()

  Gets Player's amount of Gold.
- Storage & getStorageType ()

  Gets Player's type of Storage.
- void attackEnemy (Enemy &e)

Attack an Enemy.

- void attackEnemy (Enemy &e, Item &WS)

  Attack an Enemy with a Weapon/Shield.

- void Heal (const size_t Index)

  Healing with an Item.

5.8.1 Detailed Description

Player has Gold and a Storage in which it can hold some Item. The Player will be able to spend Gold in the Shop.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Player() [1/2]

```
Player::Player (
            const char * Name,
            double HP = 100,
            double MaxHP = 100,
            double BaseDMG = 5,
            double BaseDEF = 5,
            size_t Gold = 0,
            Storage StorageType = Storage() )
```

Constructor.

5.8.2.2 Player() [2/2]

```
Player::Player (
            double HP = 100,
            double MaxHP = 100,
            double BaseDMG = 5,
            double BaseDEF = 5,
            size_t Gold = 0 )
```

Default Constructor.

5.8.2.3 ~Player()

```
Player::~Player ( )  [virtual]
```

Destructor.

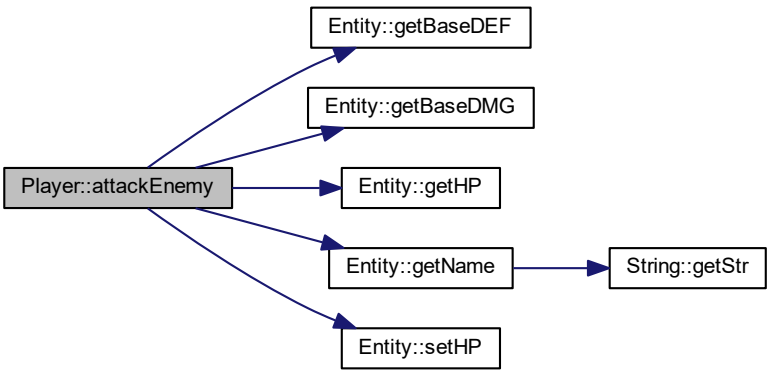
5.8.3 Member Function Documentation

5.8.3.1 attackEnemy() [1/2]

```
void Player::attackEnemy (
            Enemy & e )
```

Attack an Enemy.

Here is the call graph for this function:

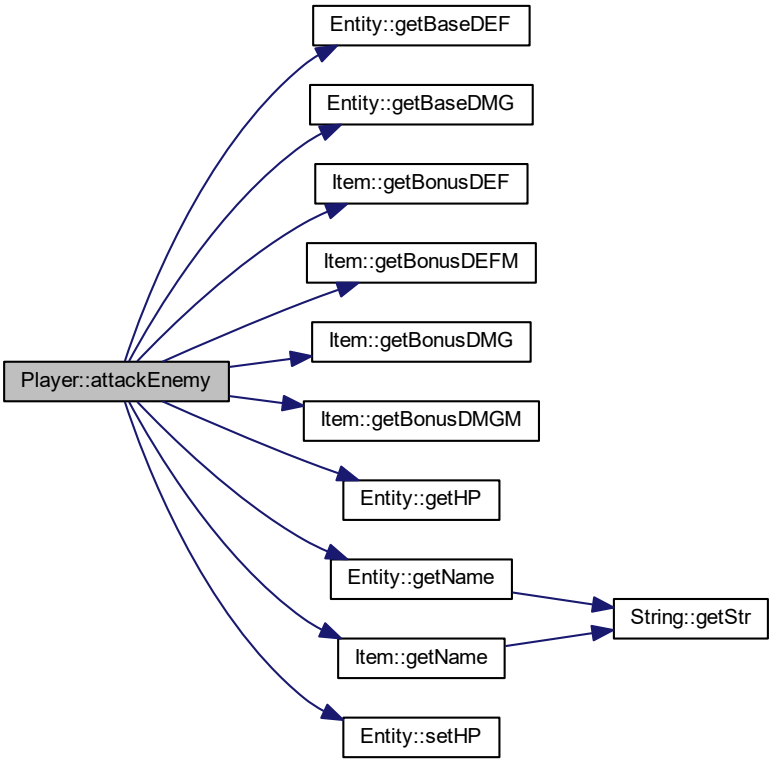


5.8.3.2 attackEnemy() [2/2]

```
void Player::attackEnemy (
            Enemy & e,
            Item & WS )
```

Attack an Enemy with a Weapon/Shield.

Here is the call graph for this function:

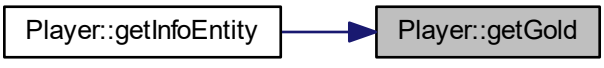


5.8.3.3 getGold()

```
size_t Player::getGold ( )
```

Gets Player's amount of Gold.

Here is the caller graph for this function:



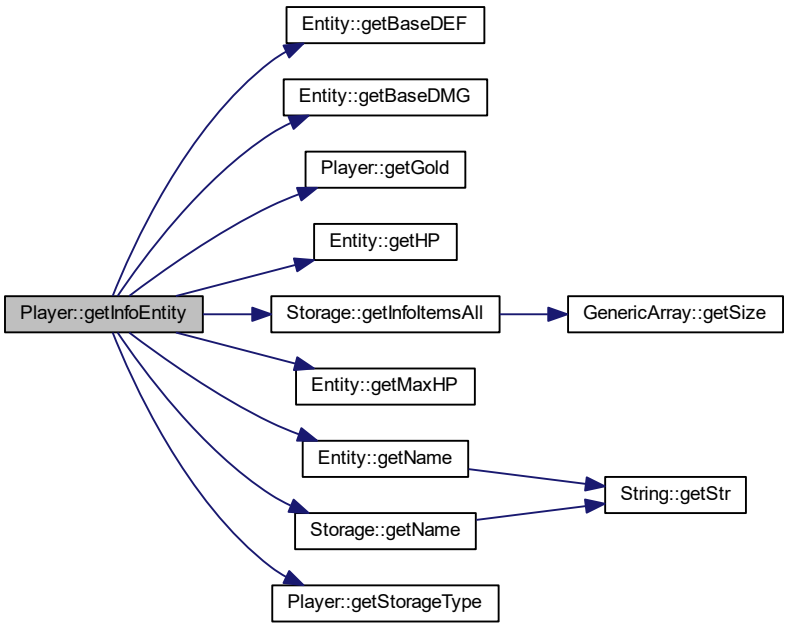
5.8.3.4 getInfoEntity()

```
std::ostream & Player::getInfoEntity (
            InfoPreset Preset = InfoPreset(),
            std::ostream & os = std::cout )  [virtual]
```

Writes out the Player's info to an output.

Implements Entity.

Here is the call graph for this function:

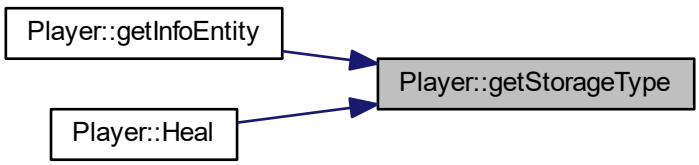


5.8.3.5 getStorageType()

```
Storage & Player::getStorageType ( )
```

Gets Player's type of Storage.

Here is the caller graph for this function:

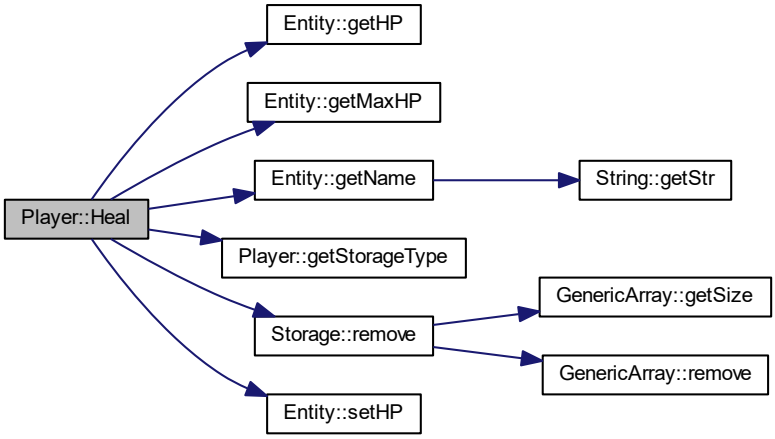


5.8.3.6 Heal()

```
void Player::Heal (
            const size_t Index )
```

Healing with an Item.

Here is the call graph for this function:

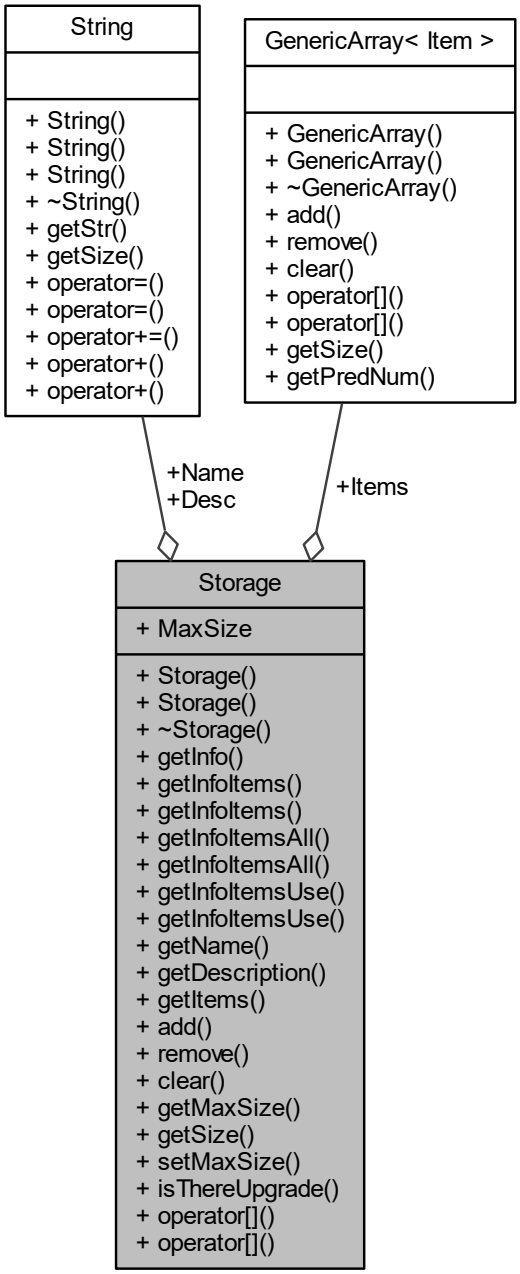


5.9 Storage Class Reference

A Storage is able to store Items, but to a limited extent (MaxSize). Also has a Name and a Description.

```
#include <Storage.h>
```

Collaboration diagram for Storage:



Public Member Functions

- Storage (const char *Name="BakPak", const char *Desc="This is a BakPak.", size_t MaxSize=5)

  Default Constructor.
- Storage (const Storage &S)

  Copy Constructor.
- virtual ~Storage ()

Destructor.

- std::ostream & getInfo (std::ostream &os=std::cout)

  Writes out the Storage's info to the console.
- std::ostream & getInfoItems (InfoPreset Preset=InfoPreset(), std::ostream &os=std::cout)

  Writes out the Item's infos in the Storage to the console.
- std::ostream & getInfoItems (const ItemType &Type, InfoPreset Preset=InfoPreset(), std::ostream &os=std↩ ::cout)

  Writes out a specific Item's infos in the Storage to the console.
- std::ostream & getInfoItemsAll (std::ostream &os=std::cout)

  Writes out the Item's infos in the Storage to the console.
- std::ostream & getInfoItemsAll (const ItemType &Type, std::ostream &os=std::cout)

  Writes out a specific Item's infos in the Storage to the console.
- virtual std::ostream & getInfoItemsUse (std::ostream &os=std::cout)

  Writes out the Item's infos in the Storage to the console.
- virtual std::ostream & getInfoItemsUse (const ItemType &Type, std::ostream &os=std::cout)

  Writes out a specific Item's infos in the Storage to the console.
- const char * getName ()

  Gets Name.
- const char * getDescription ()

  Gets Description.
- GenericArray< Item > getItems () const

  Gets the Item's array, though cannot modify it.
- void add (const Item &Element)

  Adds an Item to the Storage.
- void remove (const size_t &Index)

  Removes an Item from the Storage according to its index.
- void clear ()

  Clears the Storage.
- size_t getMaxSize ()

  Gets size of Storage.
- size_t getSize ()

  Gets number of Items in the of Storage.
- void setMaxSize (const size_t &SizeC)

  Sets size of Storage.
- bool isThereUpgrade ()

  Returns if there are upgrades in the inventory.
- Item & operator[ ] (size_t Index) const

  Accessing an Item as constant.
- Item & operator[ ] (size_t Index)

  Accessing an Item.

Public Attributes

- String Name

  Name of the Storage.
- String Desc

  Description of the Storage.
- size_t MaxSize

  Size of Storage, how many Item it can hold.
- GenericArray< Item > Items

  Items in the Storage.

5.9.1 Detailed Description

A Storage is able to store Items, but to a limited extent (MaxSize). Also has a Name and a Description.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Storage() [1/2]

```
Storage::Storage (
            const char * Name = "BakPak",
            const char * Desc = "This is a BakPak.",
            size_t MaxSize = 5 )
```

Default Constructor.

5.9.2.2 Storage() [2/2]

```
Storage::Storage (
            const Storage & S )
```

Copy Constructor.

5.9.2.3 ~Storage()

```
Storage::~Storage ( )  [virtual]
```

Destructor.

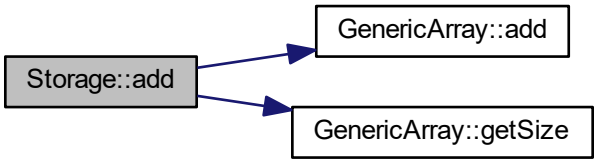
5.9.3 Member Function Documentation

5.9.3.1 add()

```
void Storage::add (
            const Item & Element )
```

Adds an Item to the Storage.

Here is the call graph for this function:

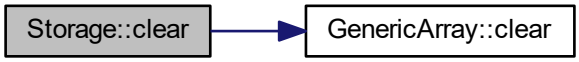


5.9.3.2 clear()

```
void Storage::clear ( )
```

Clears the Storage.

Here is the call graph for this function:

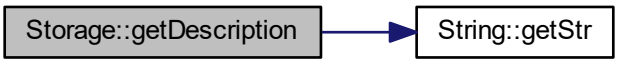


5.9.3.3 getDescription()

```
const char * Storage::getDescription ( )
```

Gets Description.

Here is the call graph for this function:

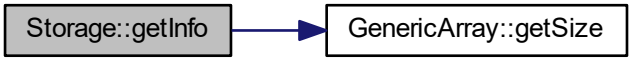


5.9.3.4 getInfo()

```
std::ostream & Storage::getInfo (
            std::ostream & os = std::cout )
```

Writes out the Storage's info to the console.

Here is the call graph for this function:



5.9.3.5 getInfoItems() [1/2]

```
std::ostream & Storage::getInfoItems (
            const ItemType & Type,
            InfoPreset Preset = InfoPreset(),
            std::ostream & os = std::cout )
```

Writes out a specific Item's infos in the Storage to the console.

Here is the call graph for this function:

5.9.3.6 getInfoItems() [2/2]

```
std::ostream & Storage::getInfoItems (
            InfoPreset Preset = InfoPreset(),
            std::ostream & os = std::cout )
```

Writes out the Item's infos in the Storage to the console.

Here is the call graph for this function:

5.9.3.7 getInfoItemsAll() [1/2]

```
std::ostream & Storage::getInfoItemsAll (
            const ItemType & Type,
            std::ostream & os = std::cout )
```

Writes out a specific Item's infos in the Storage to the console.

Here is the call graph for this function:

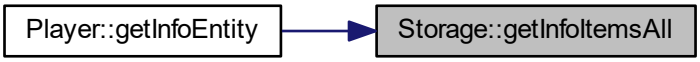
5.9.3.8 getInfoItemsAll() [2/2]

```
std::ostream & Storage::getInfoItemsAll (
            std::ostream & os = std::cout )
```

Writes out the Item's infos in the Storage to the console.

Here is the call graph for this function:

Here is the caller graph for this function:

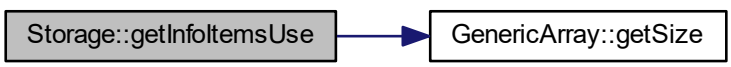


5.9.3.9 getInfoItemsUse() [1/2]

```
std::ostream & Storage::getInfoItemsUse (
            const ItemType & Type,
            std::ostream & os = std::cout )  [virtual]
```

Writes out a specific Item's infos in the Storage to the console.

Here is the call graph for this function:



5.9.3.10 getInfoItemsUse() [2/2]

```
std::ostream & Storage::getInfoItemsUse (
            std::ostream & os = std::cout )  [virtual]
```

Writes out the Item's infos in the Storage to the console.

Here is the call graph for this function:

5.9.3.11 getItems()

```
GenericArray< Item > Storage::getItems ( ) const
```

Gets the Item's array, though cannot modify it.

5.9.3.12 getMaxSize()

```
size_t Storage::getMaxSize ( )
```

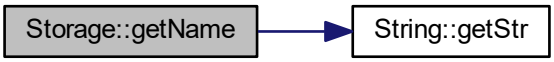
Gets size of Storage.

5.9.3.13 getName()

```
const char * Storage::getName ( )
```

Gets Name.

Here is the call graph for this function:



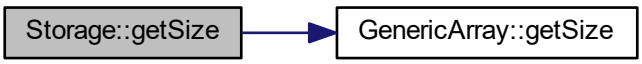
Here is the caller graph for this function:

5.9.3.14 getSize()

```
size_t Storage::getSize ( )
```

Gets number of Items in the of Storage.

Here is the call graph for this function:



5.9.3.15 isThereUpgrade()

```
bool Storage::isThereUpgrade ( )
```

Returns if there are upgrades in the inventory.

Here is the call graph for this function:

5.9.3.16 operator[]() [1/2]

```
Item& Storage::operator[] (
            size_t Index )  [inline]
```

Accessing an Item.

5.9.3.17 operator[]() [2/2]

```
Item& Storage::operator[] (
            size_t Index ) const  [inline]
```

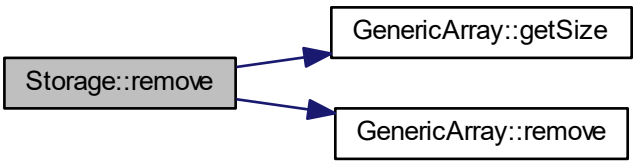
Accessing an Item as constant.

5.9.3.18 remove()

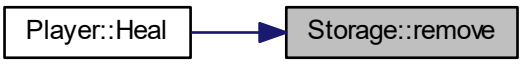
```
void Storage::remove (
            const size_t & Index )
```

Removes an Item from the Storage according to its index.

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.19 setMaxSize()

```
void Storage::setMaxSize (
            const size_t & SizeC )
```

Sets size of Storage.

5.9.4 Member Data Documentation

5.9.4.1 Desc

```
String Storage::Desc
```

Description of the Storage.

5.9.4.2 Items

```
GenericArray<Item> Storage::Items
```

Items in the Storage.

5.9.4.3 MaxSize

```
size_t Storage::MaxSize
```

Size of Storage, how many Item it can hold.

5.9.4.4 Name

```
String Storage::Name
```

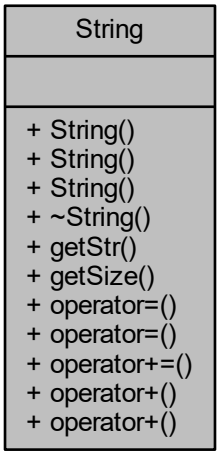
Name of the Storage.

5.10 String Class Reference

Seperate String class, because STL are not allowed...

`#include <String.h>`

Collaboration diagram for String:



Public Member Functions

- String (String const &str)
  Copy Constructor.
- String (const char *str="")
  Default Constructor from const char.*
- String (const char c)
  Constructor from char.
- virtual ~String ()
  Destructor.
- const char * getStr ()
  Return the content of the String.
- const size_t getSize ()
  Return the length of the String.
- String & operator= (const String &rhs_s)
  Assign the String's data to the String.
- String & operator= (const char *rhs_s)
  Assign const char's data to the String.*
- String & operator+= (const String &rhs_s)
  Adding String to the original.
- String operator+ (const String &rhs_s) const
  Adding 2 Strings together (returns constant)
- String operator+ (char rhs_c) const
  Adding a char to the String (returns constant)

5.10.1 Detailed Description

Seperate String class, because STL are not allowed...

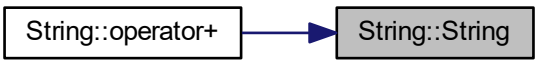
5.10.2 Constructor & Destructor Documentation

5.10.2.1 String() [1/3]

```
String::String (
            String const & str )
```

Copy Constructor.

Here is the caller graph for this function:



5.10.2.2 String() [2/3]

```
String::String (
            const char * str = "" )
```

Default Constructor from const char∗.

5.10.2.3 String() [3/3]

```
String::String (
            const char c )
```

Constructor from char.

5.10.2.4 ~String()

```
virtual String::~String ( )  [inline], [virtual]
```

Destructor.

5.10.3 Member Function Documentation

5.10.3.1 getSize()

```
const size_t String::getSize ( )  [inline]
```

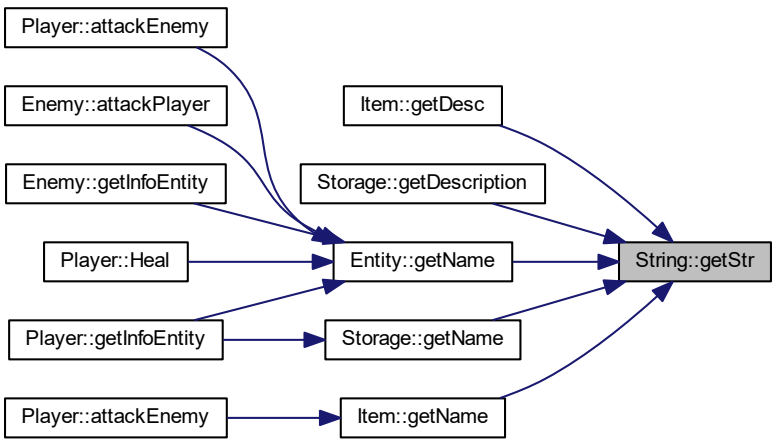
Return the length of the String.

5.10.3.2 getStr()

```
const char* String::getStr ( )  [inline]
```

Return the content of the String.

Here is the caller graph for this function:



5.10.3.3 operator+() [1/2]

```
String String::operator+ (
            char rhs_c ) const  [inline]
```

Adding a char to the String (returns constant)

Here is the call graph for this function:

5.10.3.4 operator+() [2/2]

```
String String::operator+ (
            const String & rhs_s ) const
```

Adding 2 Strings together (returns constant)

5.10.3.5 operator+=()

```
String& String::operator+= (
            const String & rhs_s )  [inline]
```

Adding String to the original.

5.10.3.6 operator=() [1/2]

```
String & String::operator= (
            const char * rhs_s )
```

Assign const char*'s data to the String.

5.10.3.7 operator=() [2/2]

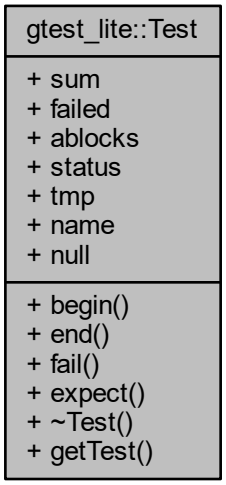
```
String & String::operator= (
            const String & rhs_s )
```

Assign the String's data to the String.

5.11 gtest_lite::Test Struct Reference

```
#include <gtest_lite.h>
```

Collaboration diagram for gtest_lite::Test:



Public Member Functions

- void begin (const char *n)

  Teszt kezdete.
- std::ostream & end (bool memchk=false)

  Teszt vége.
- bool fail ()
- std::ostream & expect (bool st, const char *file, int line, const char *expr, bool pr=false)

  Eredményt adminisztráló tagfüggvény True a jó eset.
- ~Test ()

  Destruktor.

Static Public Member Functions

- static Test & getTest ()

Public Attributes

- int sum

  tesztek számlálója
- int failed

  hibás tesztek
- int ablocks

  allokált blokkok száma
- bool status

  éppen futó teszt státusza.
- bool tmp

  temp a kivételkezeléshez;
- std::string name

  éppen futó teszt neve.
- std::fstream null

  nyelő, ha nem kell kiírni semmit

5.11.1 Detailed Description

Tesztek állapotát tároló osztály. Egyetlen egy statikus példány keletkezik, aminek a destruktora a futás végén hívódik meg.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ~Test()

```
gtest_lite::Test::~Test ( )  [inline]
```

Destruktor.

5.11.3 Member Function Documentation

5.11.3.1 begin()

```
void gtest_lite::Test::begin (
            const char * n )  [inline]
```

Teszt kezdete.

5.11.3.2 end()

```
std::ostream& gtest_lite::Test::end (
            bool memchk = false )  [inline]
```

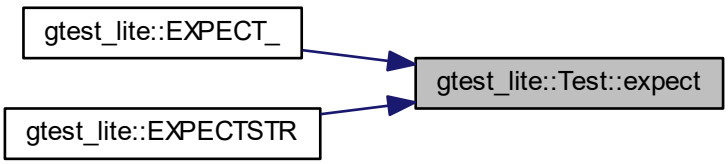
Teszt vége.

5.11.3.3 expect()

```
std::ostream& gtest_lite::Test::expect (
            bool st,
            const char * file,
            int line,
            const char * expr,
            bool pr = false )  [inline]
```

Eredményt adminisztráló tagfüggvény True a jó eset.

Here is the caller graph for this function:



5.11.3.4 fail()

```
bool gtest_lite::Test::fail ( )  [inline]
```

5.11.3.5 getTest()

```
static Test& gtest_lite::Test::getTest ( )  [inline], [static]
```

< egyedüli (singleton) példány

5.11.4 Member Data Documentation

5.11.4.1 ablocks

```
int gtest_lite::Test::ablocks
```

allokált blokkok száma

5.11.4.2 failed

```
int gtest_lite::Test::failed
```

hibás tesztek

5.11.4.3 name

```
std::string gtest_lite::Test::name
```

éppen futó teszt neve.

5.11.4.4 null

```
std::fstream gtest_lite::Test::null
```

nyelő, ha nem kell kiírni semmit

5.11.4.5 status

```
bool gtest_lite::Test::status
```

éppen futó teszt státusza.

5.11.4.6 sum

```
int gtest_lite::Test::sum
```

tesztek számlálója

5.11.4.7 tmp

```
bool gtest_lite::Test::tmp
```

temp a kivételkezeléshez;

Index