GTCode

[Nome della società] |[Indirizzo della società]

Manuale della libreria

giorgio testa

1.0.0

Sommario

[**Generators** 2](#_Toc117505839)

[**ObservablePropertyWrapper** 3](#_Toc117505840)

[Introduzione 3](#_Toc117505841)

[Utilizzo 3](#_Toc117505842)

[Esempio 3](#_Toc117505843)

[**ObservableValidatedPropertyWrapper** 4](#_Toc117505844)

[Introduzione 4](#_Toc117505845)

[Utilizzo 4](#_Toc117505846)

[Esempio 4](#_Toc117505847)

[**ObservableClassWrapper [DEPRECATO DALLA 1.1.0]** 5](#_Toc117505848)

[Introduzione 5](#_Toc117505849)

[Utilizzo 5](#_Toc117505850)

[Esempio 5](#_Toc117505851)

[**Utils** 6](#_Toc117505852)

[**Optional** 7](#_Toc117505853)

[Introduzione 7](#_Toc117505854)

[Metodi 7](#_Toc117505855)

[Esempio 7](#_Toc117505856)

# **Extensions**

**GTCode.Extensions**

Estensioni per classi .net Core.

Versione Libreria 1.0.0

## **EnumExtension**

### Introduzione

Classe ideata per aggiungere nuove funzionalità a System.Enum e alle sue classi derivanti.

### Metodi

GetDescription() ottiene il contenuto dell’attributo System.ComponentModel.Descritpion, se assente restituisce una ***string*** vuota.

GetAttributeOfType<T> ottiene l’attributo fornito come T, se assente restituisce ***null***;

### Esempio

1. **public** **enum** MyEnum {
2. [Description(“MyEnum.P1”)]
3. Property1,
4. Property2
5. }
6. //class definition...
7. **public** **void** ExampleMethod {
8. MyEnum.Property1.GetDescription(); //return “MyEnum.P1”
9. MyEnum.Property2.GetDescription(); //return “”
10. MyEnum.Property1.GetAttributeOfType<Description>(); //return Description[“MyEnum.P1”]
11. MyEnum.Property2.GetAttributeOfType<Description>(); //return null
12. }

# **Generators**

**GTCode.Generators**

Attributi per la generazione di codice da iniettare nelle classi al momento della compilazione.

Versione Libreria 1.1.0

## **ObservablePropertyWrapper**

### Introduzione

Attributo ideato per semplificare l’esposizione delle Property di oggetti nativamente non-observable. L’attributo richiede una configurazione iniziale della classe dalla quale esporre le Property: la nuova classe deve essere annotata con il modificatore ***partial***, deve estendere la classe ***ObservableObject*** di CommunityToolkit e possedere un Field contenente la classe di origine (quella non-observable).

### Utilizzo

Dopo aver indicato lo ***using*** dell’attributo (GTCode.Generators.MVVM.CommunityToolkit.Wrappers) dichiarare e annotare i Field che si vogliono esporre e rendere observable. ObservablePropertyWrapper richiede obbligatoriamente di specificare il nome del Field contenente l’oggetto core: [ObservablePropertyWrapper(“\_coreItem”)]

In automatico viene generata una Property con il nome del Field annotato trasformato in PascalCase, rimuovendo eventuali underscore iniziali.

E’ possibile indicare un nome personalizzato da attribuire alla Property tramite il campo PropertyName: [ObservablePropertyWrapper(“\_coreItem”, PropertyName=”LaMiaProperty”)]

E’ possibile esporre una Property annidata utilizzando il campo CorePropertyChain: [ObservablePropertyWrapper(“\_coreItem”, CorePropertyChain=”InnerItem.Id”)]

### Esempio

1. //using ...
2. **using** GTCode.Generators.MVVM.CommunityToolkit.Wrappers;
3. //namespace ...
5. **public** **partial** **class** CarModel : ObservableObject {
7. **private** **readonly** Car \_car;
9. **public** CarModel(Car car) { **this**.\_car = car; }
11. [ObservablePropertyWrapper(“\_car”)]
12. **private** **int** id; //diventa Id
14. [ObservablePropertyWrapper(“\_car”, PropertyName=”Targa2”)]
15. **private** **string** \_targa; //diventa Targa2
17. [ObservablePropertyWrapper(“\_car”, CorePropertyChain=”Produttore.Nome”)]
18. **private** **string** \_produttore; //diventa Produttore, chiama \_car.Produttore.Nome
20. }

## **ObservableValidatedPropertyWrapper**

### Introduzione

Attributo ideato per semplificare l’esposizione delle Property di oggetti nativamente non-observable che necessitano di validazione. L’attributo richiede una configurazione iniziale della classe dalla quale esporre le Property: la nuova classe deve essere annotata con il modificatore ***partial***, deve estendere la classe ***ObservableValidator*** di CommunityToolkit e possedere un Field contenente la classe di origine (quella non-observable).

### Utilizzo

Dopo aver indicato lo ***using*** dell’attributo (GTCode.Generators.MVVM.CommunityToolkit.Wrappers) dichiarare e annotare i Field che si vogliono esporre e rendere observable. ObservablePropertyWrapper richiede obbligatoriamente di specificare il nome del Field contenente l’oggetto core: [ObservablePropertyWrapper(“\_coreItem”)]

In automatico viene generata una Property con il nome del Field annotato trasformato in PascalCase, rimuovendo eventuali underscore iniziali. Per indicare la validazione è necessario fornire, tramite il campo Validators, l’invocazione degli attributi di validazione separati da “;”: [ObservablePropertyWrapper(“\_coreItem”, Validators=”[System.ComponentModel.DataAnnotations.MinLength(2)]”)]

E’ possibile indicare un nome personalizzato da attribuire alla Property tramite il campo PropertyName: [ObservablePropertyWrapper(“\_coreItem”, PropertyName=”LaMiaProperty”)]

E’ possibile esporre una Property annidata utilizzando il campo CorePropertyChain: [ObservablePropertyWrapper(“\_coreItem”, CorePropertyChain=”InnerItem.Id”)]

### Esempio

1. //using ...
2. **using** GTCode.Generators.MVVM.CommunityToolkit.Wrappers;
3. //namespace ...
5. **public** **partial** **class** CarModel : ObservableValidator {
7. **private** **readonly** Car \_car;
9. **public** CarModel(Car car) { **this**.\_car = car; }
11. [ObservableValidatedPropertyWrapper(“\_car”, Validators=@“
12. [System.ComponentModel.DataAnnotations.Required];
13. [System.ComponentModel.DataAnnotations.MinLength(2)]
14. ”)]
15. **private** **string** \_targa;
16. }

## **ObservableClassWrapper [DEPRECATO DALLA 1.1.0]**

### Introduzione

Attributo ideato per semplificare l’esposizione delle Property di oggetti nativamente non-observable mantenendo tutte le funzionalità fornite dagli attributi CommunityToolkit. ObservableClassWrapper richiede una configurazione iniziale della classe dalla quale esporre le Property: la nuova classe deve essere annotata con il modificatore ***partial***, deve estendere la classe ***ObservableObject*** di CommunityToolkit e possedere un Field contenente la classe di origine (quella non-observable).

L’attributo indicherà al compilatore di generare un costruttore che inizializzerà tutti i Field annotati con ***ObservableProperty*** di CommunityToolkit, e un metodo denominato GetCore(), il quale restituisce l’oggetto core valorizzato opportunamente in base al contenuto dell’oggetto model.

### Utilizzo

Dopo aver indicato lo ***using*** dell’attributo (GTCode.Generators.MVVM.CommunityToolkit.Wrappers) annotare la classe.

ObservableClassWrapper richiede obbligatoriamente di specificare il nome del Field contenente l’oggetto core: [ObservableClassWrapper(“\_coreItem”)]

### Esempio

1. //using ...
2. **using** GTCode.Generators.MVVM.CommunityToolkit.Wrappers;
3. //namespace ...
5. [ObservableClassWrapper("\_car")]
6. **public** **partial** **class** CarModel : ObservableObject {
8. **private** **readonly** Car \_car;
10. //public CarModel(Car core) : costruttore generato
12. [ObservableProperty]
13. **private** **int** id;
15. [ObservableProperty]
16. **private** **string** \_targa;
18. **private** **string** \_produttore;
20. //public Car GetCore() : metodo generato
22. }

# **Services**

**GTCode.Services**

Servizi e oggetti ad essi relativi.

Versione Libreria 1.0.0

## **ApiClient\_HttpClient**

### Introduzione

Servizio ideato per semplificare la comunicazione verso REST API, implementa l’interfaccia IApiClient e utilizza System.Net.Http.HttpClient per eseguire la comunicazione. Espone vari metodi ***async*** e permette di indicare una basic-authentication. Ogni metodo solleva una InternalException se il body della risposta è null, e System.Net.Http.HttpRequestException se lo stato della risposta non è tra quelli di successo. E’ da notare che i metodi aspettano, nel body della risposta, un json modellato sulle classi derivanti da GenericResponse.

### Metodi

La classe mette a disposizione metodi che dipendono sulla presenza o assenza di un valore contenuto:

PostCallAPIAsync(...) esegue una POST-REQUEST asincrona;

PutCallAPIAsync(...) esegue una PUT-REQUEST asincrona;

GetCallAPIAsync(...) esegue una GET-REQUEST asincrona;

GetCallByteArrayAPIAsync(...) esegue una GET-REQUEST asincrona, scaricando un byteArray;

UploadAsync(...) esegue una POST-REQUEST inviando un multi-part-file e altri parametri definiti in un Dictionary;

### Esempio

1. **public** **void** ExampleMethod {
2. **var** emptyOpt = Optional<**string**>.Empty();
3. **var** nullableOpt = Optional<**string**>.OfNullable(**null**);
4. **var** optList = Optional<List<**string**>>.Of(**new** List<**string**>(){ "A", "B", “C” });
5. **var** opt = Optional<**string**>.Of("VALUE");
7. opt.IfPresent(**value** => **value**.Substring(1));
8. opt.IsPresent(); //true
9. emptyOpt.IsPresent(); //false
10. nullableOpt.IsPresent(); //false
11. optList.Filter(**value** => **value**.Contains("B”)); //return Optional[{“A”,”B”,”C”}]
12. optList.Filter(**value** => **value**.Contains("Z”)); //return Optional[]
13. opt.Get(); //return “VALUE”
14. emptyOpt.OrElse("X”); //return “X”
15. emptyOpt.OrElseGet(() => "Z”); //return “Z”
16. emptyOpt.OrElseThrow<Exception>(() => **throw new** Exception()); //throws Exception
17. }

# **Utils**

**GTCode.Utils**

Oggetti di utiliy originali o derivati dal porting di codice esistente in Java.

Versione Libreria 1.0.0

## **Optional**

### Introduzione

Classe ideata a partire da java.util.Optional già esistente su Java. Si tratta di un oggetto container il quale può o non può contenere un valore non-null. Se un valore è presente l’oggetto è considerato pieno, altrimenti viene considerato vuoto.

Optional è primariamente inteso per essere utilizzato come return type di un metodo dove è necessario rappresentare un “nessun risultato” ma al contempo evitare null perché potrebbe causare degli errori. Un’istanza di Optional non dovrebbe mai essere null; dovrebbe sempre puntare ad un’istanza vuota.

E’ da evitare l’utilizzo di operazioni identity-sensitive come hashCode, synchronization o equals (==), inquanto potrebbero avere risultati imprevedibili.

### Metodi

La classe mette a disposizione metodi che dipendono sulla presenza o assenza di un valore contenuto:

IsPresent() indica la presenza o assenza del valore, restituisce ***true*** se presente, ***false*** altrimenti;

IfPresent(Action) esegue l’azione fornita se IsPresent() è ***true***;

Filter(Predicate) esegue l’azione fornita se il valore è presente;

OrElse(T) restituisce il valore fornito in T se IsPresent() è ***false***;

OrElseGet(Func<T>) esegue l’azione fornita se IsPresent() è ***false***;

OrElseThrow(Func<X>) solleva l’eccezione fornita se IsPresent() è ***false***;

GetHashCode() ottiene l’HashCode del valore contenuto;

ToString() ottiene una string utile al debug.

### Esempio

1. **public** **void** ExampleMethod {
2. **var** emptyOpt = Optional<**string**>.Empty();
3. **var** nullableOpt = Optional<**string**>.OfNullable(**null**);
4. **var** optList = Optional<List<**string**>>.Of(**new** List<**string**>(){ "A", "B", “C” });
5. **var** opt = Optional<**string**>.Of("VALUE");
7. opt.IfPresent(**value** => **value**.Substring(1));
8. opt.IsPresent(); //true
9. emptyOpt.IsPresent(); //false
10. nullableOpt.IsPresent(); //false
11. optList.Filter(**value** => **value**.Contains("B”)); //return Optional[{“A”,”B”,”C”}]
12. optList.Filter(**value** => **value**.Contains("Z”)); //return Optional[]
13. opt.Get(); //return “VALUE”
14. emptyOpt.OrElse("X”); //return “X”
15. emptyOpt.OrElseGet(() => "Z”); //return “Z”
16. emptyOpt.OrElseThrow<Exception>(() => **throw new** Exception()); //throws Exception
17. }