GTCode

[Nome della società] |[Indirizzo della società]

Manuale della libreria

giorgio testa

1.0.0

Sommario

[**• Extensions** 3](#_Toc118365651)

[**EnumExtension** 4](#_Toc118365652)

[Introduzione 4](#_Toc118365653)

[Metodi 4](#_Toc118365654)

[Esempio 4](#_Toc118365655)

[**• Generators** 5](#_Toc118365656)

[**ObservablePropertyWrapper** 6](#_Toc118365657)

[Introduzione 6](#_Toc118365658)

[Utilizzo 6](#_Toc118365659)

[Esempio 6](#_Toc118365660)

[**ObservableValidatedPropertyWrapper** 7](#_Toc118365661)

[Introduzione 7](#_Toc118365662)

[Utilizzo 7](#_Toc118365663)

[Esempio 7](#_Toc118365664)

[**ObservableClassWrapper [DEPRECATO DALLA 1.1.0]** 8](#_Toc118365665)

[Introduzione 8](#_Toc118365666)

[Utilizzo 8](#_Toc118365667)

[Esempio 8](#_Toc118365668)

[**• Services** 9](#_Toc118365669)

[**IApiClient - ApiClient\_HttpClient** 10](#_Toc118365670)

[Introduzione 10](#_Toc118365671)

[Metodi 10](#_Toc118365672)

[Esempio 10](#_Toc118365673)

[**GenericResponse** 11](#_Toc118365674)

[Introduzione 11](#_Toc118365675)

[Metodi&Property 11](#_Toc118365676)

[SingleResponse 11](#_Toc118365677)

[ListResponse 11](#_Toc118365678)

[Esempio 11](#_Toc118365679)

[**• Utils** 12](#_Toc118365680)

[**Optional** 13](#_Toc118365681)

[Introduzione 13](#_Toc118365682)

[Metodi 13](#_Toc118365683)

[Esempio 13](#_Toc118365684)

[**IRecordNavigator - RecordNavigator** 14](#_Toc118365685)

[Introduzione 14](#_Toc118365686)

[Utilizzo 14](#_Toc118365687)

[Esempio 14](#_Toc118365688)

[**IRecordNavigator - RecordNavigatorPaginatedDecorator** 15](#_Toc118365689)

[Introduzione 15](#_Toc118365690)

[Utilizzo 15](#_Toc118365691)

[Esempio 15](#_Toc118365692)

[**IPaginationHandler - PaginationHandler** 16](#_Toc118365693)

[Introduzione 16](#_Toc118365694)

[Utilizzo 16](#_Toc118365695)

[Esempio 16](#_Toc118365696)

# **• 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 ObservablePropertydi 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

## **IApiClient - 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. **private readonly** IApiClient \_apiClient = **new** ApiClient\_HttpClient(**new** HttpClient());
2. **private readonly string** API\_BASE\_URL = "http://localhost:3002/gt-code/basic-auth";
3. **private readonly string** AUTH\_TOKEN = "gtcode:gtcode";
4. **public** **async void** ExampleMethod {
5. **var** url = $"{API\_BASE\_URL}/..."
6. **var** resList = **await** \_apiClient.GetCallAPIAsync<ListResponse<**string**>>(url, AUTH\_TOKEN);
7. **var** resByte = **await** \_apiClient.GetCallByteArrayAPIAsync(url, AUTH\_TOKEN);
8. **var** toSend = **new** MyObject{Prop1 = **1**, Prop2 = "PostCallAPIAsync"};
9. **var** resSingle = **await** \_apiClient.PostCallAPIAsync<SingleResponse<**string**>>(
10. url, toSend, AUTH\_TOKEN
11. );
12. resSingle = **await** \_apiClient.PutCallAPIAsync<SingleResponse<**string**>>(
13. url, toSend, AUTH\_TOKEN
14. );
15. **var** dict = **new** Dictionary<**string**,**string**>(){{"1","uno"},{"2","due"},{"3","tre"}};
16. resSingle = **await** \_apiClient.PutCallAPIAsync<SingleResponse<**string**>>(
17. url, toSend, AUTH\_TOKEN
18. );
19. **var** \_ = **await** \_apiClient.UploadAsync<GenericResponse>(
20. url, File.ReadAllBytes("/path..."), "myFile.txt", "fileToUpload", **null**, AUTH\_TOKEN
21. );
22. }

## **GenericResponse**

### Introduzione

Classe ideata per rappresentare il corpo Json di risposta di una richiesta HTTP verso un applicativo REST. Espone Property e metodi volti al controllo dell’esito della chiamata effettuata.

### Metodi&Property

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

CheckStatus() se presente propaga un eccezione restituita dal server sollevando ServerException;

Success il successo o fallimento dell’operazione richiesta al server;

Message il messaggio restituito dal server;

### SingleResponse

Sottoclasse di GenericResponse specializzata per contenere un singolo oggetto definito in ***T***.

### ListResponse

Sottoclasse di GenericResponse specializzata per contenere una lista di oggetti definiti in ***T***. Espone una Property aggiuntiva denominata TotalCount, ideata per contenere il numero totale di elementi presenti sul server al fine di gestire una visualizzazione paginata.

### Esempio

1. **public** **async void** ExampleMethod {
2. **var** generic = **new** GenericResponse(){
3. Success = false, Message = "Message",
4. };
5. **var** single = **new** SingleResponse<**string**>(){
6. Success = true, Message = "Message",
7. Data = "1"
8. };
9. **var** generic = **new** ListResponse<**string**>(){
10. Success = true, Message = "Message",
11. Data = **new** List<**string**>(){"1", "2", "3"},
12. TotalCount = **30**
13. };
15. }

# **• 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. }

## **IRecordNavigator - RecordNavigator**

### Introduzione

Classe di utility ideata per semplificare navigazione di collezioni, implementa l’interfaccia IRecordNavigator e si focalizza sul navigare liste.

### Utilizzo

Per utilizzare RecordNavigator è necessario istanziarlo fornendo la collezione che si vuole navigare e l’azione da effettuare al passaggio da un elemento e l’altro della collezione.

La prima volta, precedentemente ad eseguire Next o Previous, o dopo la sostituzione della collezione da navigare è necessario eseguire il metodo Start.

### Esempio

Esempio utilizzando RecordNavigator in congiunzione con CommunityToolkit.

1. **private readonly** IRecordNavigator<**string**> \_navigator;
2. **private** List<**string**> \_collection;
3. [ObservableProperty]
4. **public string** \_selectedItem;
5. **public** **void** Costruttore {
6. \_collection = **new** List<**string**>();
7. \_navigator = **new** RecordNavigator<**string**>(\_collection, (**index**) => {
8. SelectedItem = \_collection[**index**];
9. });
10. }
11. **private void** LoadCollection() {
12. \_collection = //valorizzo la collezione con dati prelevati da un DAO o Messenger
13. \_navigator.Start(\_collection, **0**);
14. }
15. [RelayCommand]
16. **public void** Next() {
17. \_navigator.Next();
18. }
19. [RelayCommand]
20. **public void** Previous() {
21. \_navigator.Previous();
22. }

## **IRecordNavigator - RecordNavigatorPaginatedDecorator**

### Introduzione

Classe di utility ideata per semplificare la navigazione di collezioni gestendone la paginazione. Chiamando Next o Previous viene invocato il relativo metodo del RecordNavigator fornito, al raggiungimento dell’ultimo item della collezione viene chiamato il metodo di caricamento di una nuova pagina. Se si raggiunge uno dei due estremi massimi della paginazione (prima pagina o ultima pagina) non viene caricata nessuna nuova pagina.

### Utilizzo

Per utilizzare questo decorator è necessario istanziarlo fornendo un RecordNavigator e le azioni da intraprendere al passaggio da una pagina e l’altra.

La prima volta, precedentemente ad eseguire Next o Previous, o dopo la sostituzione della collezione da navigare è necessario eseguire il metodo Start.

### Esempio

Esempio utilizzando RecordNavigatorPaginatedDecorator in congiunzione con CommunityToolkit.

1. **private readonly** RecordNavigatorPaginatedDecorator<**string**> \_navigator;
2. **private** List<**string**> \_collection;
3. [ObservableProperty]
4. **public string** \_selectedItem;
5. **public** **void** Costruttore {
6. \_collection = **new** List<**string**>();
7. **var** navigator = **new** RecordNavigator<**string**>(\_collection, (**index**) => {
8. SelectedItem = \_collection[**index**];
9. });
10. //rispettivamente: recordNavigator, nextAction, previousAction, limit, totalPages
11. \_navigator = **new** RecordNavigatorPaginatedDecorator<**string**>(navigator, () => {
12. **this**.LoadCollection();
13. }, () => {
14. **this**.LoadCollection();
15. }, **10**, **5**);
16. }
17. **private void** LoadCollection() {
18. \_collection = //valorizzo la collezione con dati prelevati da un DAO o Messenger
19. \_navigator.Start(\_collection, **0**);
20. }
21. [RelayCommand]
22. **public void** Next() {
23. \_navigator.Next();
24. }
25. [RelayCommand]
26. **public void** Previous() {
27. \_navigator.Previous();
28. }

## **IPaginationHandler - PaginationHandler**

### Introduzione

Classe di utility ideata per semplificare la paginazione di risultati e la loro navigazione.

### Utilizzo

Per utilizzare questa classe è necessario istanziarla fornendo le azioni da intraprendere al passaggio da una pagina e l’altra, il numero di elementi per pagina e il numero totale di elementi tra tutte le pagine.

### Esempio

Esempio utilizzando PaginationHandler in congiunzione con CommunityToolkit.

1. **private readonly** PaginationHandler \_paginationHandler;
2. **private** List<**string**> \_collection;
3. [ObservableProperty]
4. **public string** \_selectedPage;
5. **public** **void** Costruttore {
6. \_collection = **new** List<**string**>();
7. //rispettivamente: nextAction, previousAction, limit, totalCount
8. \_paginationHandler = **new** PaginationHandler((**offset**, **limit**) => {
9. **this**.LoadCollection(offset, limit);
10. }, (**offSet**, **limit**) => {
11. **this**.LoadCollection(offset, limit);
12. }, **10**, **50**);
13. }
14. **private void** LoadCollection(**int** offset, **int** limit) {
15. \_collection = //valorizzo la collezione con dati prelevati da un DAO o Messenger
16. }
17. [RelayCommand]
18. **public void** NextPage() {
19. SelectedPage = \_paginationHandler.NextPage().ToString();
20. }
21. [RelayCommand]
22. **public void** PreviousPage() {
23. SelectedPage = \_paginationHandler.PreviousPage().ToString();
24. }