# Better Event System

*Release 0.0.1*

**addikted**

**Apr 10, 2022**

# CONTENTS


**1 Starter Guide** **3**
- 1.1 Installation . . . 3
- 1.2 Usage . . . 3

**2 API Reference** **5**
- 2.1 Event . . . 5
- 2.2 Event System . . . 6
- 2.3 Event Args . . . 7

**3 Help** **9**
- 3.1 preprocessor / postprocessor . . . 9

*Create an extensive Event System with ease*

**Modern**

BetterEventSystem is a modern library, with support for asynchronous events.

**Extensive**

BetterEventSystem is extensive, with a lot of features that you will never need. but if you do, you got it!

**Easy to use**

BetterEventSystem is a simple, easy to use library, with a simple API. simple tasks, simple code.

**Open source**

Nothing spooky, nothing scary, BetterEventSystem is open source, and licensed under a very permissive license.

CHAPTER

ONE

# STARTER GUIDE

*How to get started with BetterEventSystem*

**See also:**

This is a short guide to help you get a project up and running. you should then go to the *API reference* to extend the functionality of your project.

## 1.1 Installation

Installation is simple. just install `Addikted.BetterEventSystem` From your nuget package manager.

## 1.2 Usage

**Note:** before you start using BetterEventSystem, you need to add the `Addikted.BetterEventSystem` namespace to your project.

### 1.2.1 Creating a new event

```
new Event("event_name")
```

An event will be created with the name `event_name`.

### 1.2.2 Getting an event

```
EventSystem.GetEvent("event_name")
```

This will find and return an event with the name `event_name`, if it does not exist it will create it.

### 1.2.3 Adding a listener

```
EventSystem.GetEvent("event_name").AddListener((e) => {
    //do something
});
```

A listener is a function that is called when the event is fired. All listeners are called in the order they were added.

---

### 1.2.4 Broadcasting / emitting an event

```
EventSystem.GetEvent("event_name").Broadcast(data);
```

you can pass any object you want to the event, and it will be passed to all listeners/preprocessors through the EventArgs.data property.

All done, check out the *API reference* to see how to extend the functionality of your project.

CHAPTER

TWO

# API REFERENCE

**See also:**

This is the API reference for the BetterEventSystem. If you are looking for a guide, please refer to the *starter guide*.

## 2.1 Event

all the following properties and methods are available in the `Event.` class.

### 2.1.1 Constructor

| Name | Type | Default | Description |
|---|---|---|---|
| name | `String` | N/A | The name of the Event |
| allowAsync | `Boolean` | `true` | Whether or not to allow async calls to the listeners |
| register | `Boolean` | `true` | Whether or not to register the event in the Event System. |

### 2.1.2 Methods

| Name | Return Type | Parameters | Description |
|---|---|---|---|
| AddListener | `void` | `Action<EventArgs>` listener | Add a listener to the Event |
| RemoveListener | `void` | `Action<EventArgs>` listener | Remove a listener from the event |
| AddPreprocessor | `void` | `Action<EventArgs, Action<EventArgs>>` preprocessor | Add a preprocessor to the event |
| RemovePreprocessor | `void` | `Action<EventArgs, Action<EventArgs>>` preprocessor | Remove a preprocessor from the event |
| RemoveAllListeners | `void` | N/A | Remove all listeners from the event |
| RemoveAllPreprocessors | `void` | N/A | Remove all preprocessor from the event |
| RemoveAll | `void` | N/A | Remove all listeners and preprocessor |
| Broadcast | `EventArgs` | `object` data = null | Broadcast the event to all listeners, passing the data to all inside the `EventArgs.data` property, returns the final EventArgs object |

### 2.1.3 Properties

| Name | Type | Description |
|---|---|---|
| Name | `String` | The name of the event |
| AllowAsync | `Boolean` | Whether or not to allow async calls to the listeners |

## 2.2 Event System

The event system is the heart of the BetterEventSystem. It is a static class that contains all the events and their listeners. The following properties and methods are available in the `EventSystem.` class.

### 2.2.1 Methods

| Name | Return Type | Parameters | Description |
|---|---|---|---|
| GetEvent | `Event` | `String` name, `bool` safe = true | Get an event by name, creating the event if `safe` is true |
| Register | `Event` | `Event` event | Register an event |

---

### 2.2.2 Properties

| Name | Type | Description |
|---|---|---|
| Events | `List<Event>` | A list of all the events |

## 2.3 Event Args

The `EventArgs` class allows you to pass data to listeners.

### 2.3.1 Methods

| Name | Return Type | Parameters | Description |
|---|---|---|---|
| cancel | `void` | `bool` cancel = true | cancel the execution of the layers. *learn more* |

---

### 2.3.2 Properties

| Name | Type | Description |
|---|---|---|
| data | `object` | contains the data sent to the broadcast method, can be modified during middleware |

CHAPTER

THREE

# HELP

Here are some things about BetterEventSystem that i think might be confusing, so i wrote this page to help you.

Don't be afraid to use the contents section at the right of this page to find what you need.

If you still have questions, check the *API reference* or feel free to contact me

## 3.1 preprocessor / postprocessor

### postprocessor

preprocessors are functions that are called before all listeners. They can be used to modify the data before your listeners are called.

### How do I create a preprocessor?

to add a preprocessor to an event, use the `AddPreprocessor` method. your preprocessor function must take two parameters, the first is an EventData, the second is an Action.

Once you have finished modifying the data, you must call the second parameter to continue the event. If you don't, the preprocessor will be called in an infinite loop until you do.

If you don't want to modify the data, just call the second parameter anyway.

Example:

```
// add preprocessor to the event
EventSystem.GetEvent("my_event").AddPreprocessor((e, next) => {
    Console.WriteLine("my_event is about to be triggered, this is a preprocessor");
    // cast the data to a dictionary, as we send it as a Dictionary. If your data is not␣
↪a dictionary, you must cast it to whatever you want to send.
    // But keep in mind if it is not a dictionary, the following code will not work.
    Dictionary<string, string> data = e.data as Dictionary<string, string>;
    foreach (var item in data.Values.ToList()) {
        Console.WriteLine("data: " + item); // this will print all the values in the␣
↪dictionary
    }
    data.Add("preprocessor", "true"); // add a new key to the dictionary
    e.data = data;  // set our changed data
    next(e); // pass our data to the next preprocessor or the event listener
});
```

### What is the difference between a preprocessor and a listener?

A preprocessor is a function that can be used to modify the data before your listeners are called.

A listener cannot modify data, but a preprocessor can.

### postprocessor

postprocessors are functions that are caled after all listeners.

### How do i create a postprocessor

to add a postprocessor to an event, use the `AddPostprocessor` method. your postprocessor function must take an `EventData` object.

### What is the difference between a postprocessor and a listener?

It is the exact same as a listener, except it is called after and can be cancelled by the listener dynamically.

Example:

```
// add postprocessor to the event
EventSystem.GetEvent("my_event").AddPostprocessor(e => {
    Console.WriteLine("my_event has finished being run.");
    // cast the data to a dictionary, as we send it as a Dictionary. If your data is not␣
↪a dictionary, you must cast it to whatever you want to send.
    // But keep in mind if it is not a dictionary, the following code will not work.
    Dictionary<string, string> data = e.data as Dictionary<string, string>;
    foreach (var item in data.Values.ToList()) {
        Console.WriteLine("data: " + item); // this will print all the values in the␣
↪dictionary
    }
});
```