Events Plus 1.1 Documentation

What is it?

Events Plus is a powerful delegate serializer for the Unity Engine. It has been designed to wholly replace the stock UnityEvent system, while offering significantly more features and improved runtime performance. Like UnityEvents, Events Plus enables developers to avoid otherwise hard-coded event hookups and instead manage it all solely within the user-friendly, inspector window.

Core Features

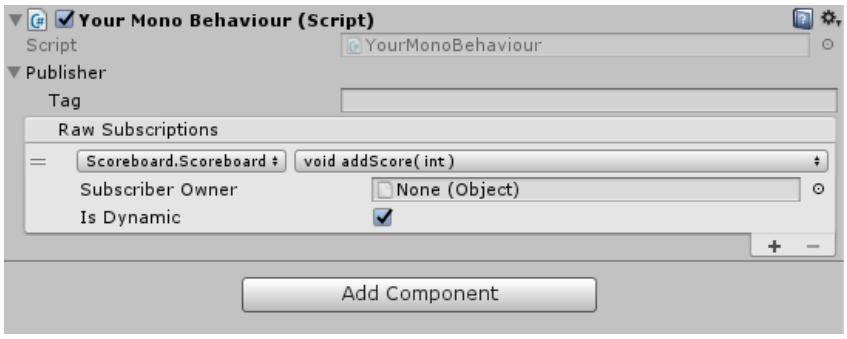
- Supports both dynamic and preconfigured calls to all public variables, properties and methods
- Supports methods with return types
- Supports function overloading
- Allows up to 6 event parameters
- Support for every serializable Unity type
- Type-safe, using C# generics
- Automatic event subscriptions
- Expandable and re-orderable inspector drawers
- Optimized for cross-platform with up to 1000% faster invocations speeds over UnityEvents
- Custom, specialized event pattern that offers memory-leak protection over UnityEvents
- Source code included and fully-documented

Quick Start

Events Plus setup is similar to UnityEvents, but requires a couple extra lines of code for initializing and un-initializing. This is necessary for registering everything to the event system and avoids any possible memory leaks.

```
[Serializable]
public class PublisherInt : Publisher<int>
{
}
```

```
public class YourMonoBehaviour : MonoBehaviour
{
	public PublisherInt publisher;

	public void Awake()
	{
		publisher.initialize();
	}

	public void OnDestroy()
	{
		publisher.clear();
	}

	public void Start()
	{
		publisher.publish( 5 );
	}
}
```

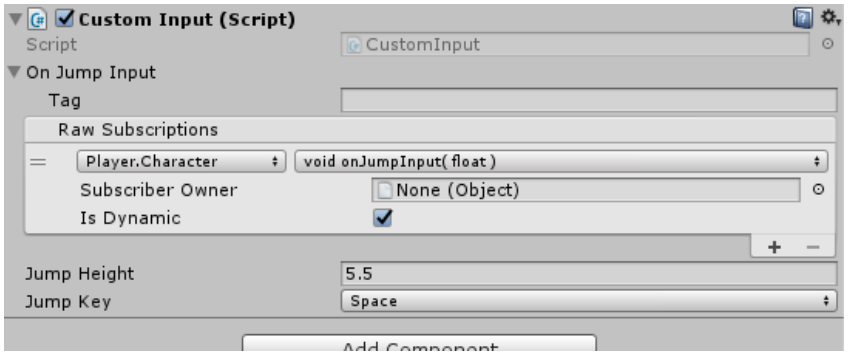


Publisher

A Publisher works just like a standard UnityEvent. It contains a list of desired calls that can be setup to be either dynamic or preconfigured with arguments. When a Publisher's "publish" method is fired, it will execute every call in the list.

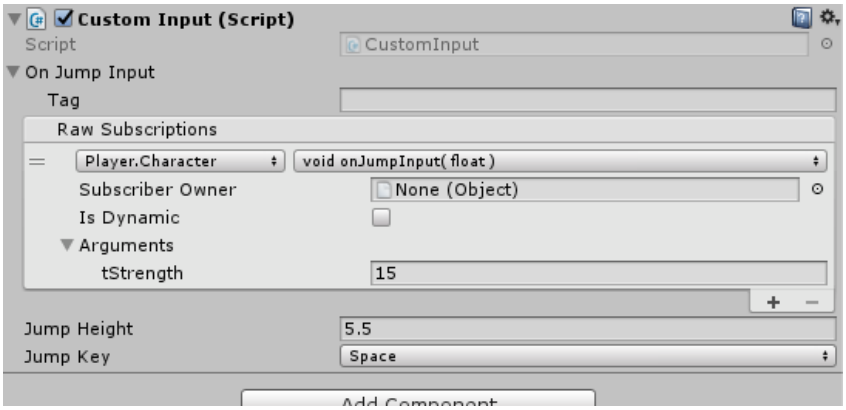
Dynamic Calls

If a call is dynamic, the variable(s) passed through the Publisher's "publish" method gets forwarded to that call.



Preconfigured Calls

If a call is preconfigured, the arguments within the inspector are used whenever the Publisher's "publish" method is fired.



Subscriber

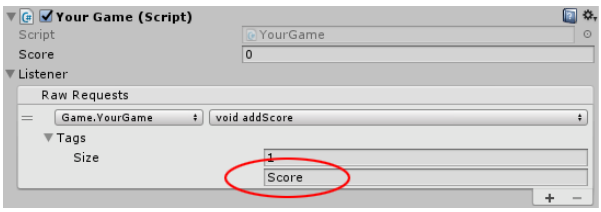
A Subscriber is an optional object that can be put onto custom classes. It manages automatic event registration and also serves as a way to prevent memory leaks.

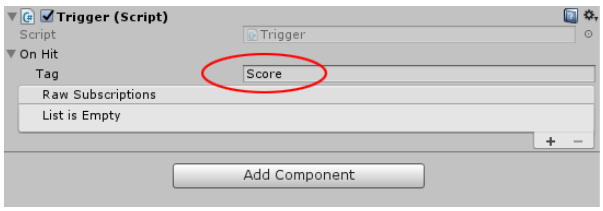
Setup

```
public class YourMonoBehaviour : MonoBehaviour
{
	public Subscriber subscriber;

	public void Awake()
	{
		subscriber.initialize();
	}

	public void OnDestroy()
	{
		subscriber.clear();
	}
}
```

Requests

Every Subscriber contains a list of “requests” that operate similar to the Publisher’s list of calls. Each request can be setup to automatically register to a Publisher if that Publisher contains a matching “tag” name. This is especially useful for hooking events up to spawned objects.

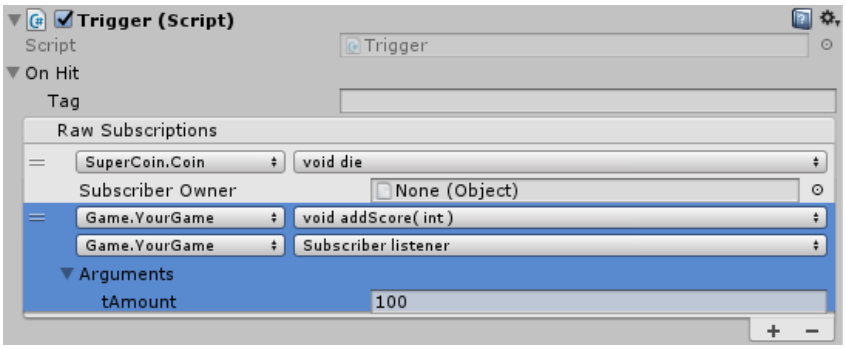




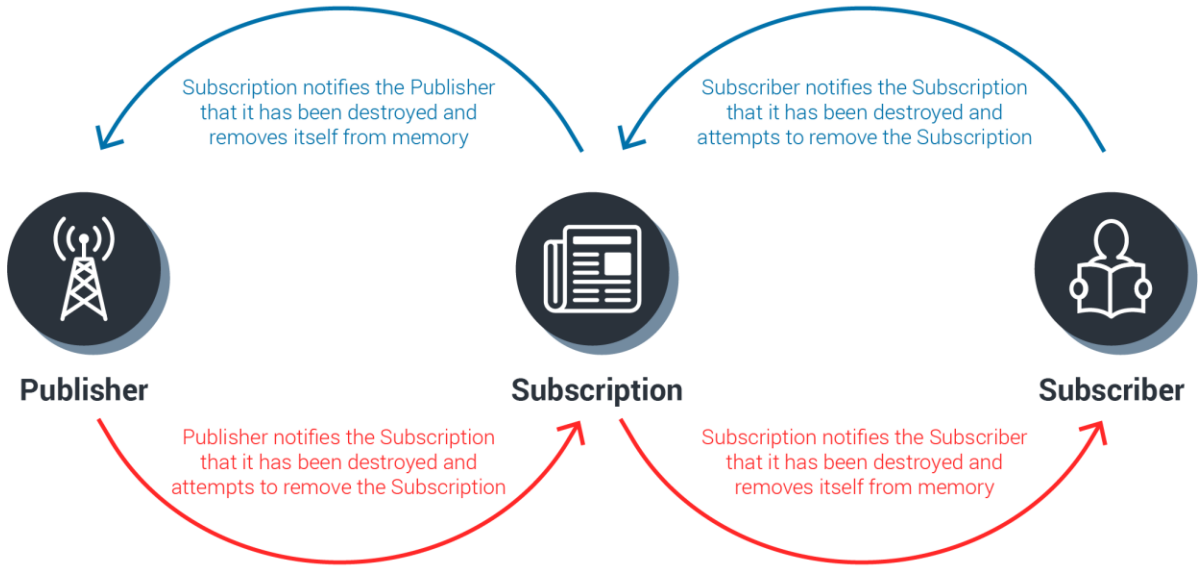
Memory Management

Subscriber objects are also used to remove memory leaks that can happen when an object is destroyed, but a Publisher (or even UnityEvent) tries to still make calls to it.

To use this feature, a Subscriber must be exposed as a variable inside of a custom class. Then a Publisher's call must be setup in the inspector and connected to that Subscriber variable.



This will establish a "subscription" object that couples the Subscriber and Publisher together. When a Subscriber is destroyed, it is then able to remove the subscription from itself and the Publisher. This works the same for when a Publisher is destroyed.



Release Notes

1.1

- Updated to the latest 2017 Unity build
- Fixed specific crash issue with IOS
- Replaced the “channel” integer with a string “tag” name for request tracking
- Added an early version of a “Settings” panel for method filtering
- Included an example that utilizes a spawned Singleton instance