**How to use**

If you’re familiar with the UnityEvent, then this will be no problem, but AsyncEvent does have a few extra features.

**Adding methods**

Once you’ve added your AsyncEvent in code, it should look something like this:

Graphical user interface, text, application

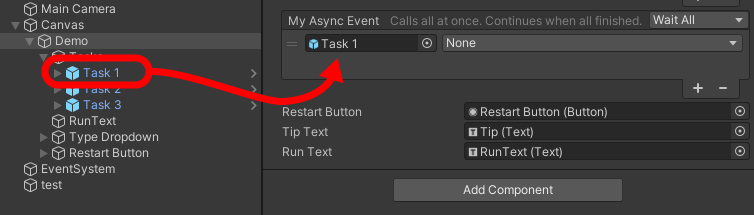
Description automatically generated

You can use the ‘+’ and ‘-’ to add or remove calls.

Graphical user interface

Description automatically generated

Then you just drag a GameObject from the hierarchy to the empty slot.



You will now be able to pick any public method defined on the object if it returns Task or void and has 1 or no parameters.

Once you’ve picked the method you want, you’re done. You can add as many events as you want.

A screenshot of a computer

Description automatically generated with medium confidence

**Types of Invoking**

The dropdown in the corner will specify, how the event will be invoked.

*WhenAll*Tasks will run simultaneously. The event finishes when all tasks are finished.

*Sequence* Tasks will run one after another. The event finishes when last task is finished.

*Synchronous*   
Tasks will run simultaneously. The event finished immediately!

**How to implement**

Implementing the AsyncEvent is simple.

Just add a using statement at the top of your file:

using AsyncEvent;

Then declare your event like any other variable:

public AsyncEvent myAsyncEvent;

That’s it! It should now show up in the inspector, ready for use!

To invoke the event you call the Invoke() method.

Calling Invoke() will invoke the event according to the type of Invoke, you specified in the Editor, but you can force a specific type by adding it as a parameter:

await myAsyncEvent?.Invoke();

await myAsyncEvent?.Invoke(AsyncEventType.Sequence);

As you can see, the event can be awaited, like any other async task.

*If you have problems or feedback regarding AsyncEvent, please email me:*[asyncevent@realfastgames.com](mailto:asyncevent@realfastgames.com)