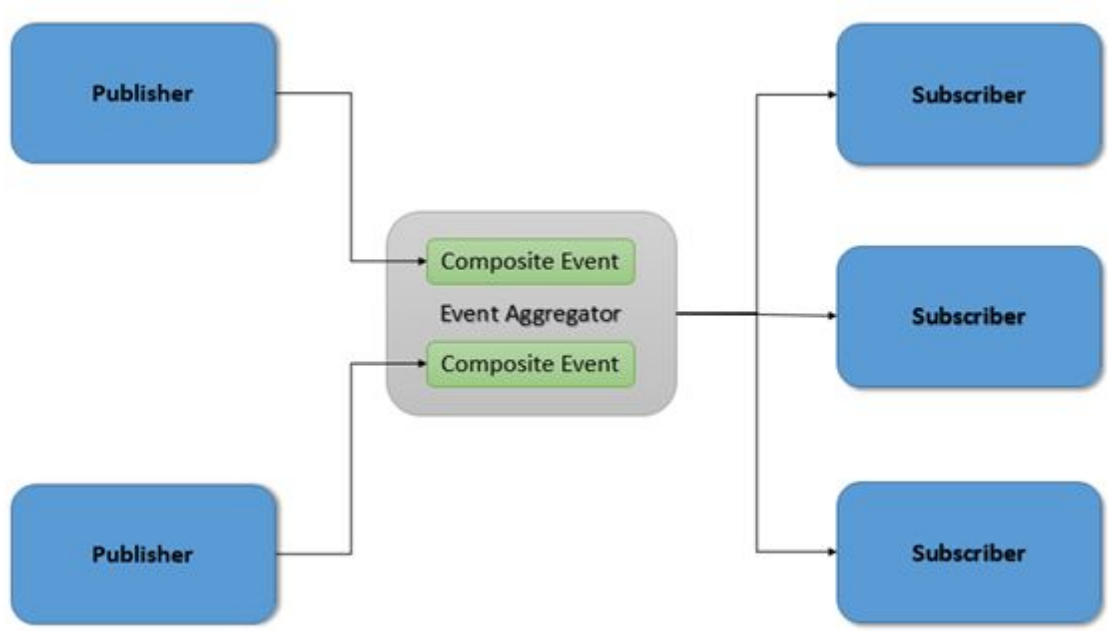
Event Aggregator Message System

Event Aggregator is a pattern that tries overcome the limitation of traditional event handling approach by providing a central place to publish and subscribe for events which is nothing but an Event Aggregator. The Event Aggregator takes care for registering, unregistering and invoking of events loosely coupling Publishers and Subscribers. All Publishers and Subscribers of event will know only about the Event Aggregator. The diagram below shows a typical Event aggregator in an Application.



Traditional Event handling

The steps below describes Traditional Event Handling.

1. Publisher advertises its event(s) and specifies when they will be invoked.
2. A Subscriber will then register itself to Publisher event(s).
3. As Publisher object undergoes changes in its lifetime it will invoke the event informing the Subscriber(s).

Disadvantages of Traditional Event handling

- If there are multiple Subscribers and Publishers then code become hard to read and debug.
- The Subscriber of the event needs to know the Publisher of an event and refer it directly by event names causing a tight coupling between them.
- The tight coupling will not allow Subscribers and Publishers to change independently of each other.

- Its Subscriber responsibility to register and unregister from an event, its seen many practical scenarios the subscriber typically forgets to unregister causing both subscriber and publisher to be in memory causing memory leaks.

Intent of Event Aggregator Pattern

- Simplify subscribing and unsubscribing to events
- Decouple publishers from subscribers allowing both to change without affecting the other.
- Makes it easy to add new events.
- Centralized handling of events.

How To's

How to create my own event?
For every event you want to create, you should create a new class deriving from BaseGameEvent.
Put in your new event every information you need.

```
public class MyEvent : BaseGameEvent { }
```

How to publish an event?
Instantiate (or reuse a previously instantiated) a new event of the type you want to publish. Fill all pertinent informations in event. Call EventAggregator.Publish using the signature that match your use.

```
EventAggregator.Publish("EventID", null);
EventAggregator.Publish<MyEvent>(new MyEvent());
EventAggregator.Publish<MyEvent>();
```

How to subscribe to an event?
All you need it to call EventAggregator.AddListener passing the type of the event you want to subscribe and the appropriate callback action. Do it using signature that match your use.

```
EventAggregator.AddListener("EventID", OnEventHandler);
EventAggregator.AddListener<MyEvent>(OnMyEventHandler);
```

May I publish/subscribe an event without create a specific class?
Yes, you can! Sometimes the event you need is so simple that you don't need (or don' want) to create a specific class just to make it type safe. In this case all you need is use the signature of AddListener and Publish that accept a string as event type.

```
EventAggregator.AddListener("MySpecialEvent", OnEventHandler);
EventAggregator.Publish("MySpecialEvent", null);
```

Any doubts: reigota@gmail.com