Pub\Sub Messenger for Unity

The Problem:

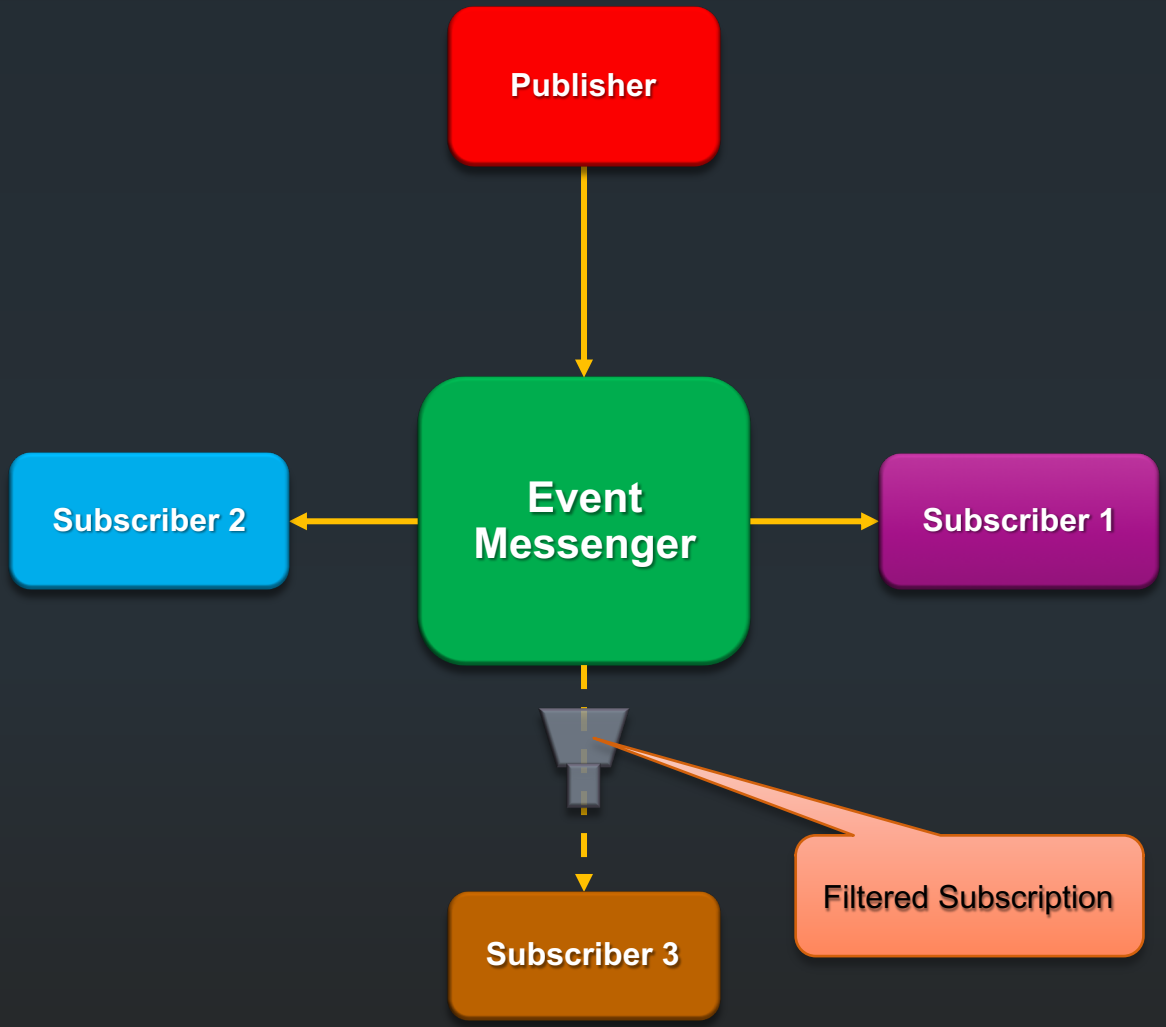
- **Wiring the Parts with Events** – Tightly Coupled and may cause Memory Leak problems. The Publisher and the Subscriber have to know of each other and a Subscriber can't be collected by the GC if it's connected with the Publisher with strong event reference.

- **Using Unity Event Routing** – Although Unity Event Routing is a very good feature, it is a Unity Specific Solution and we need a generic one. Also, we cannot use it everywhere even if the project is in Unity.

The Solution:

- ✓ **Custom Pub\Sub Messenger -** Container for Events that allows Decoupling of Publishers and Subscribers so they can evolve independently. This Decoupling is useful in Modularized Applications because new modules can be added that respond to events defined by the Shell or, more likely, other modules. All events have a Weak Reference and invocation can be done Async or Sync way.

Event Routing by Pub\Sub Messenger



Use Cases for Pub\Sub Messenger:

- ✓**Pass Payload** between disconnected/independent parts of code;
- ✓**Thread Safe** Invocation of Events between disconnected/independent parts of code;
- ✓**Asynchronous** Invocation of Events between disconnected/independent parts of code;
- ✓**Filtered** Invocation of Events between disconnected/independent parts of code;
- ✓**Obfuscated** Invocation of Events between disconnected/independent parts of code;

Messenger API:

- ✓ **Messenger**

  Implements this basic interface:

```
/// <summary>
/// Interface for Pub/Sub Messenger
/// </summary>
public interface IMessenger
{
    /// <summary>
    /// Publish given payload to relevant subscribers
    /// </summary>
    /// <param name="payload">Instance of payload to publish</param>
    /// <typeparam name="T">The type of payload to publish</typeparam>
    void Publish<T>(T payload);

    /// <summary>
    /// Subscribe given callback to receive payload
    /// </summary>
    /// <param name="callback">The callback that will receive the payload</param>
    /// <param name="predicate">The predicate to filter irrelevant payloads (optional)</param>
    /// <typeparam name="T">The type of payload to receive</typeparam>
    void Subscribe<T>(Action<T> callback, Predicate<T> predicate = null);

    /// <summary>
    /// Unsubscribe given callback from receiving payload
    /// </summary>
    /// <param name="callback">The callback that subscribed to receive payload</param>
    /// <typeparam name="T">Type of payload to unsubscribe from</typeparam>
    void Unsubscribe<T>(Action<T> callback);
}
```

Messenger API:

- ✓ **Messenger**

  Access to default Messenger instance via: `Messenger.Default`

- ✓ **Publish**

```
Messenger.Default.Publish<Payload>(new Payload{ /* payload params */ });

Payload – the payload that will be published to subscribers of this type
```

Messenger API:

✓ **Subscribe**

```
Messenger.Default.Subscribe<Payload>(Callback);

Payload – the type of Callback parameter
Callback – delegate (Method) that will receive payload

private static void Callback(Payload payload)
{
  // Callback logic
}
```

✓ **Subscribe with Predicate**

```
Messenger.Default.Subscribe<Payload>(Callback, Predicate);

Predicate – delegate (Function) that will receive payload to filter

private static bool Predicate(Payload payload)
{
  // Predicate filter logic
  // if function will return 'false' value, the Callback will not be invoked
  return accepted;
}
```

Messenger API:

- ✓ **Subscribe**

```
Messenger.Default.Unsubscribe<Payload>(Callback);
```

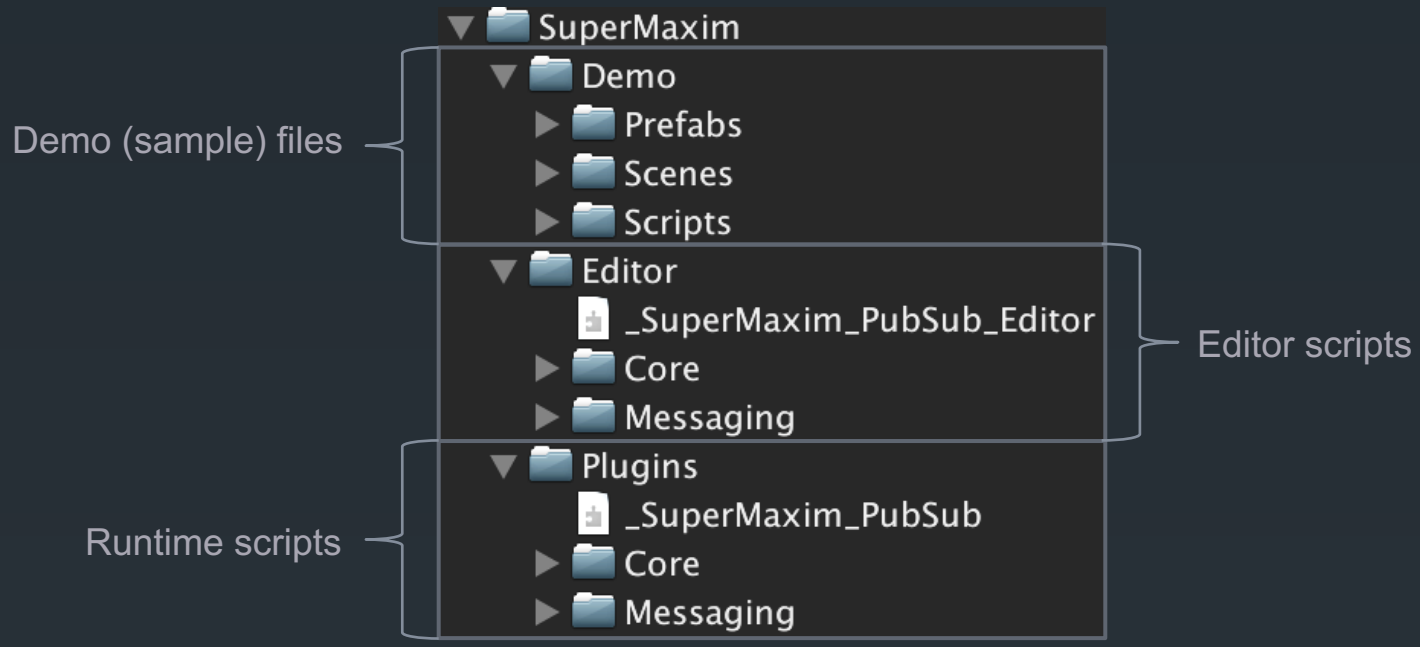
Payload – the type of Callback parameter
Callback – delegate (Method) that will be removed from subscribers

Tips for Correct Usage:

- ✓**DON'T** use Messenger if you have quick access to shared code parts to invoke events/methods in same module/class;
- ✓**ALWAYS** ensure that you unsubscribe when you're done with consuming of shared payloads;
- ✓**DON'T** publish events in endless loop;
- ✓**PREFERE** using Filtered subscriptions;

Package Structure:

✓**Folders**

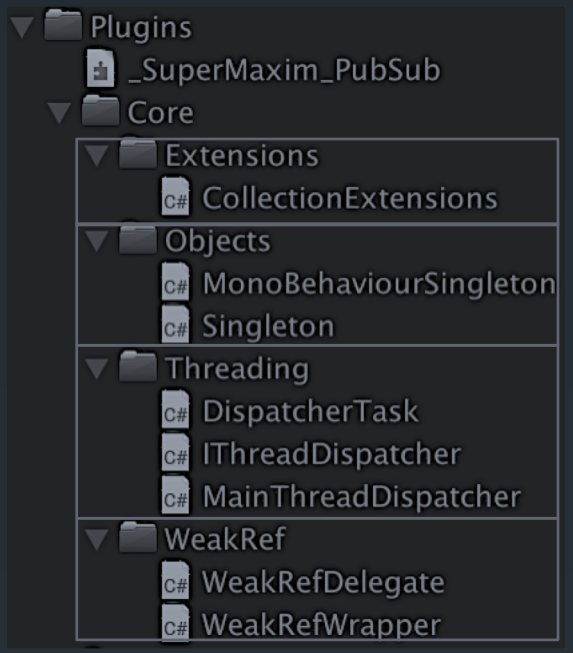


Notes:

Core – folder that contains base classes and scripts that are not specific for Messenger
Messaging – folder that contains classes and scripts that are specific for Messenger

Package Structure:

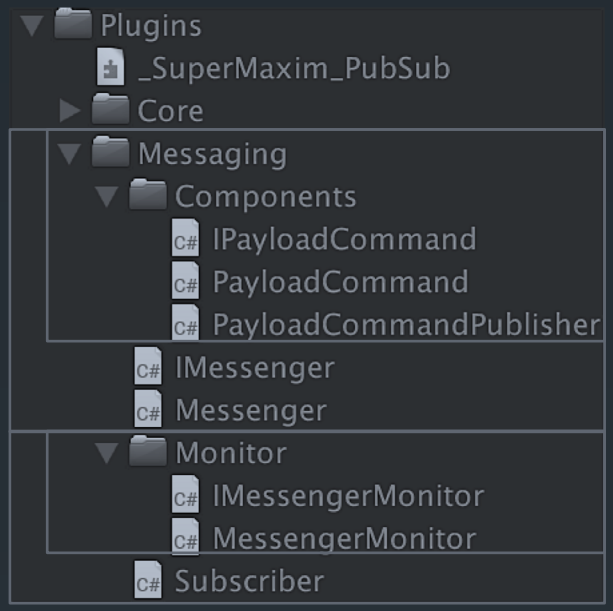
✓Plugins / Core



Folder	File/Class/Script	Description
Extensions	CollectionExtensions	Static class with collection extension functions
Objects	MonoBehaviourSingleton	Abstract base class for MB singletons
	Singleton	Abstract base class for classic singletons
Threading	DispatcherTask	Task unit with weak ref. (pointer) to delegate to execute on main thread
	IThreadDispatcher	Interface for Thread Dispatcher API
	MainThreadDispatcher	Singleton class that implements IThreadDispatcher and provides API to sync tasks between main thread and other threads
WeakRef	WeakRefDelegate	Weak reference for delegate (inherits from WeakRefWrapper)
	WeakRefWrapper	Weak reference wrapper (disposable)

Package Structure:

✓**Plugins / Messeging**



Folder	File/Class/Script	Description