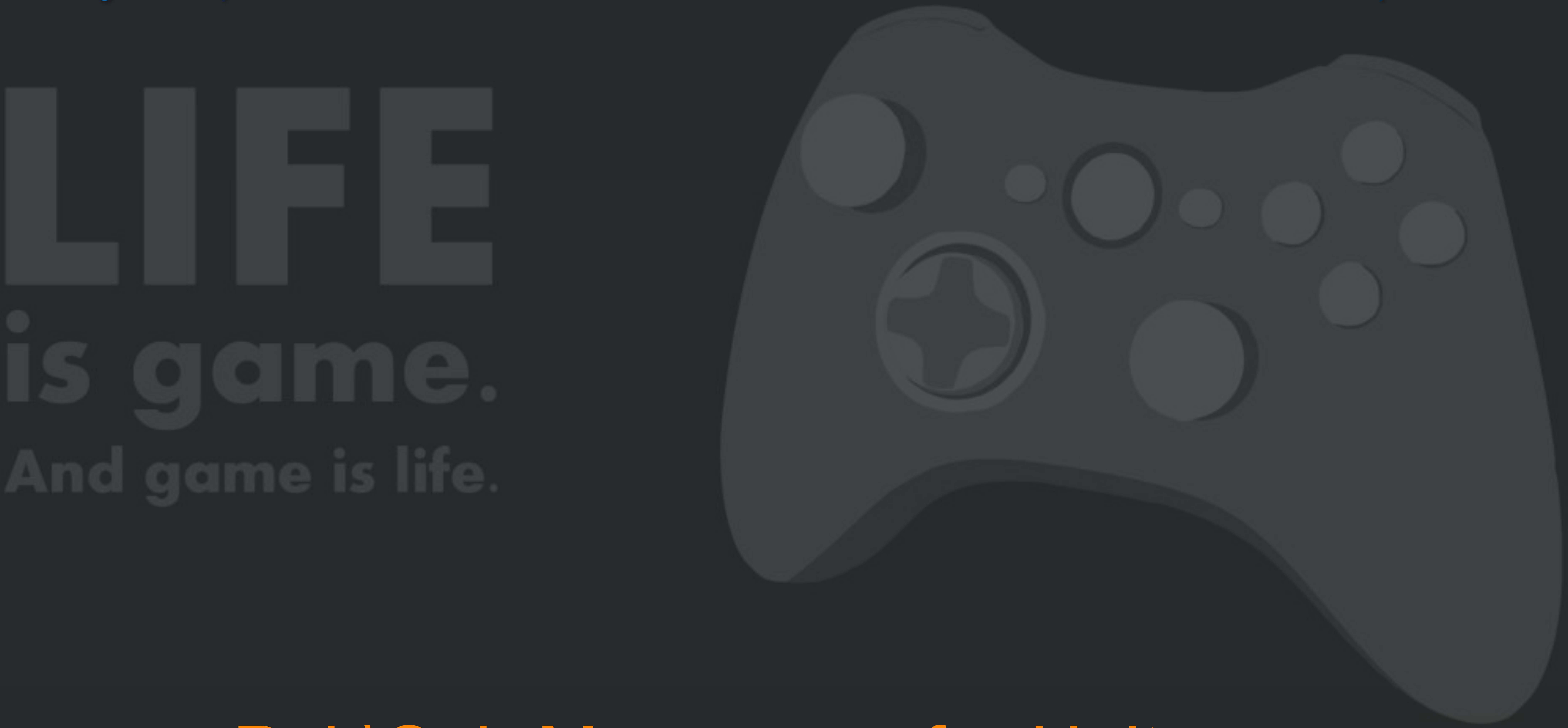
https://supermax.github.io/pubsub

SuperMaxim.com



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.

- In commonly used C# events or delegates, classes are “familiar” with each other and this prevents good modularity. This is NOT following SOLID Principles and Objects are not Encapsulated.

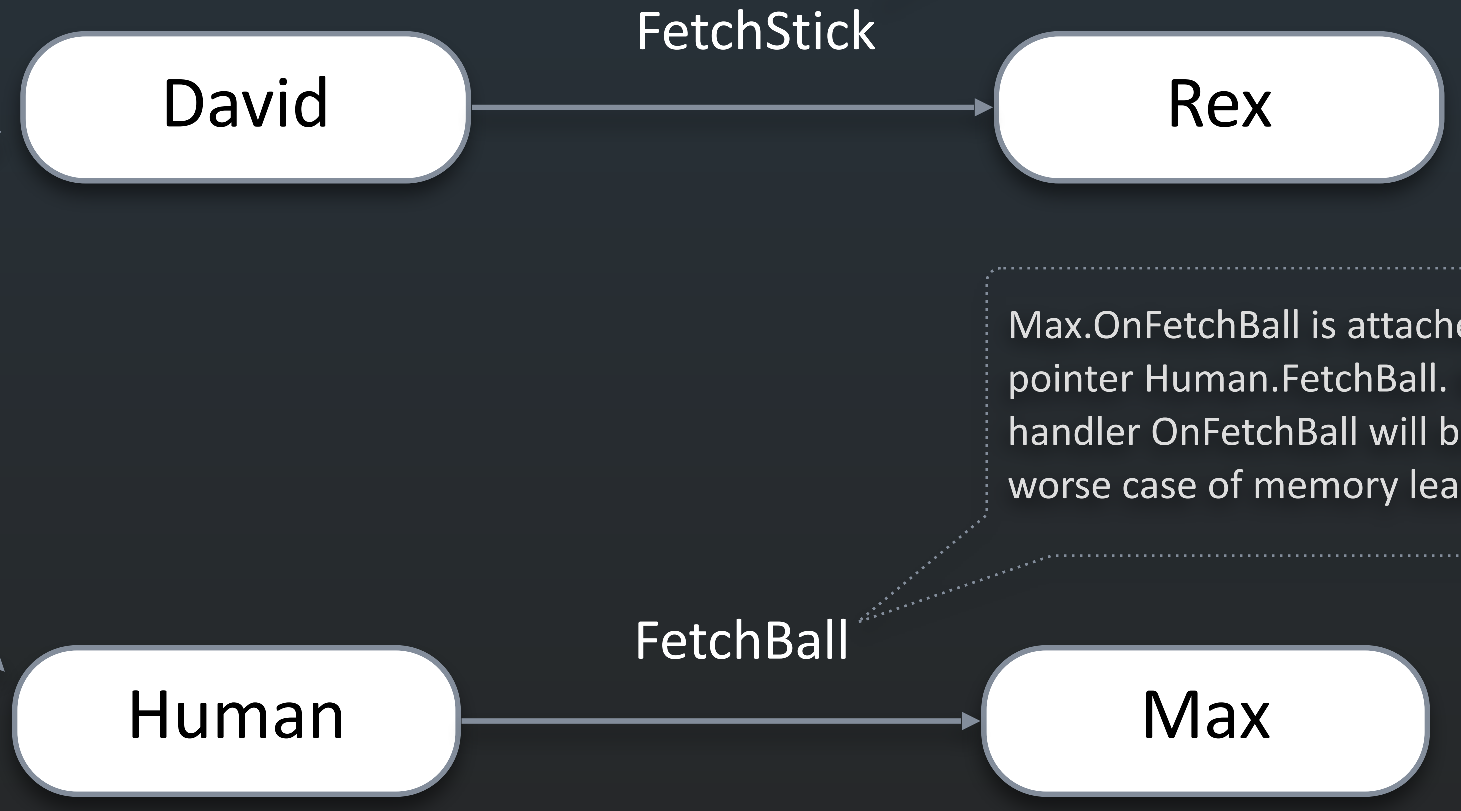
The Problem

Example of common usages of events in C#:

```
public class Human
{
    // event that passes instance of Stick when it is invoked
    public event Action<Stick> FetchStick;

    // static event that passes instance of Ball when it is invoked
    public static event Action<Ball> FetchBall;
}
```

```
public class Dog : Animal
{
    // event handler - method that is invoked by event and receives Stick instance
    public void OnFetchStick(Stick stick) { /*TODO*/ }
}


public class Cat : Animal
{
    // event handler - method that is invoked by event and receives Ball instance
    public void OnFetchBall(Ball ball) { /*TODO*/ }
}
```

```
public class Playground
{
    public Human David { get; set; }
    private Dog Rex { get; set; }
    private Cat Max { get; set; }

    public void RegisterEvents()
    {
        David.FetchStick += Rex.OnFetchStick;

        Human.FetchBall += Max.OnFetchBall;
    }
}
```



Rex.OnFetchStick is attached to David.FetchStick and David instance has reference to Rex instance. Rex instance will not be removed from the memory until it's handler OnFetchStick will be detached from FetchStick event or until David will be removed from the memory.

Max.OnFetchBall is attached to Human.FetchBall and Max instance is referenced by static pointer Human.FetchBall. Max instance will not be removed from the memory until it's handler OnFetchBall will be detached from static event FetchBall. Static references are worse case of memory leaks.

The Solution

- ✓ **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.

- ✓ Instead of passing objects or modules, pass small **Payloads** (Data/Messages) that are relevant for the specific cases/events.

- ✓ Classes/Modules will not be "familiar" with each other, this will allow **better encapsulation and less dependencies**.

- ✓ In case of subscriber's destruction, it will be removed automatically from Messenger's list, since it was referenced via **Weak Reference**.

- ✓ Pub/Sub can be a great pattern in combination with **Dependency Injection** (DP) and with **Inversion of Control** (IoC), both part of SOLID.

The Solution

Usage of Messenger as Pub/Sub mechanism:

```
// publisher
public class Human
{
    public void PublishFetchStickPayload()
    {
        // publish new payload with specific data
        Messenger.Default.Publish(
            new FetchStickPayload
                {
                    StickType = StickTypes.PlasticStick,
                    Position = new Vector3(1, 1, 0)
                });
    }
}
```

```
public class FetchStickPayload
{
    // stick type for filtering
    public StickTypes StickType { get; set; }

    // the position of stick in the space
    public Vector3 Position { get; set; }
}
```

```
// subscriber
public class Dog : Animal
{
    // callback - method that is invoked by Messenger and receives payload instance
    public void OnFetchStick(FetchStickPayload payload) { /* TODO handle stick fetching */ }
}
```

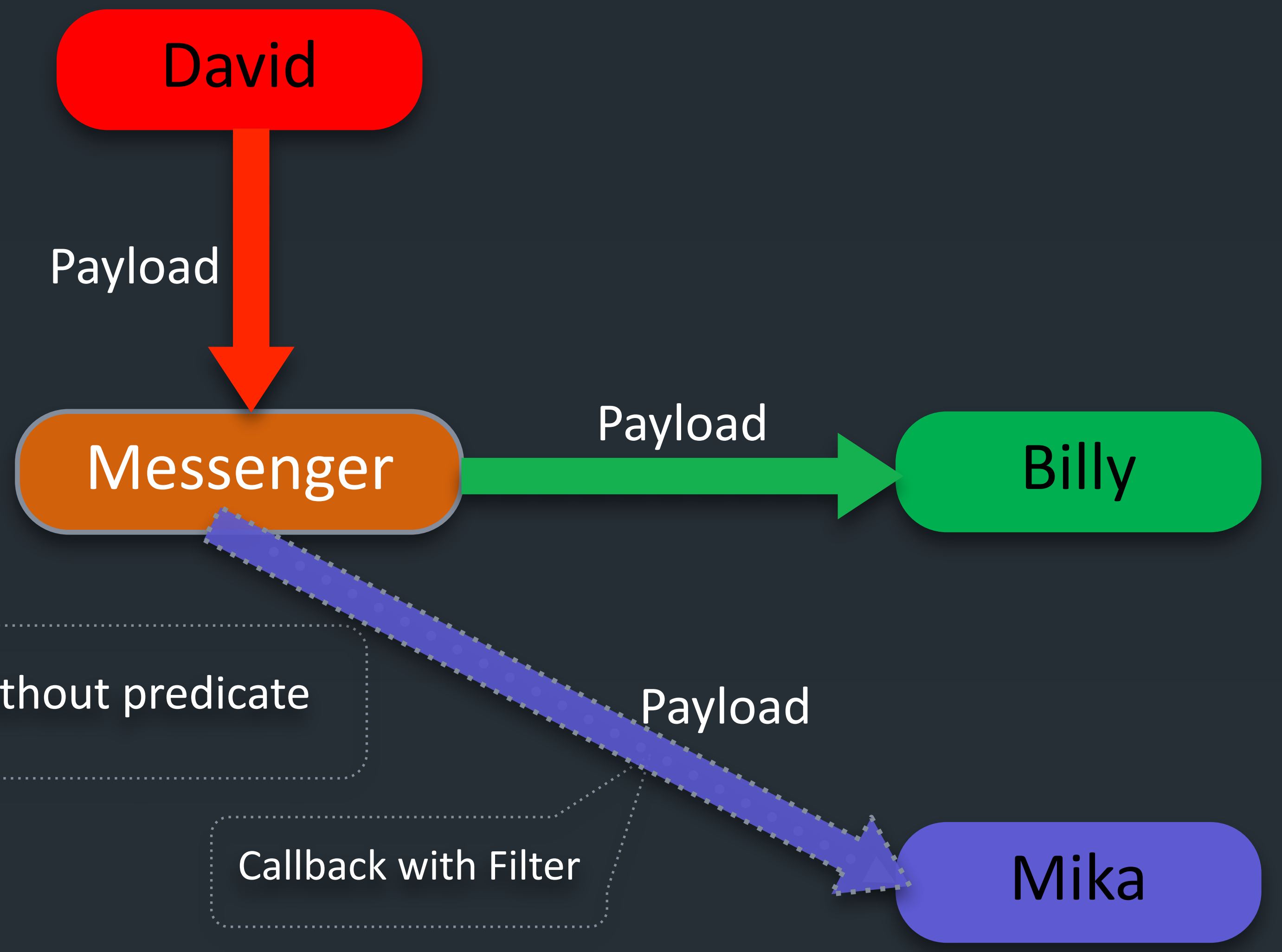
The Solution

Usage of Messenger as Pub/Sub mechanism:

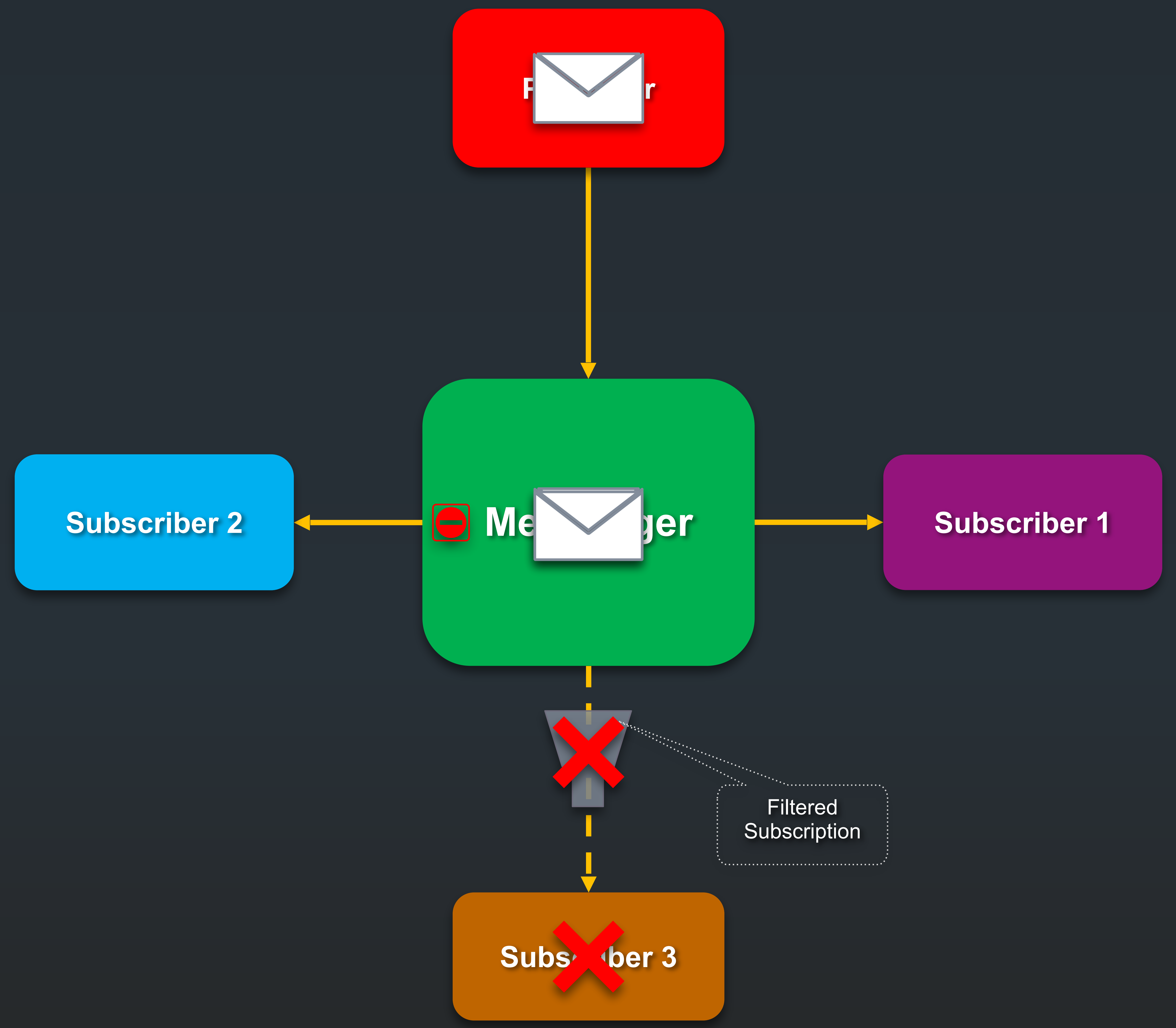
```
public class Playground
{
    public Human David { get; set; }
    public Dog Billy { get; set; }
    public Dog Mika { get; set; }

    public void Subscribe()
    {
        // subscribe callback Billy.OnFetchStick to FetchStickPayload
        Messenger.Default.Subscribe<FetchStickPayload>(Billy.OnFetchStick);

        // subscribe callback Mika.OnFetchStick to FetchStickPayload with filter/predicate
        Messenger.Default.Subscribe<FetchStickPayload>(Mika.OnFetchStick, CanFetchStick);
    }

    private bool CanFetchStick(FetchStickPayload payload) { /* TODO filter unwanted stick types */ }
}
```

Billy subscribed without predicate

Mika subscribed with predicate



Message Routing by Pub\Sub Messenger



Use Cases for Pub\Sub Messenger

√**Pass Payload** between disconnected parts of code.

√**Thread Safe** invocation of callbacks between disconnected parts of code.

√**Asynchronous** invocation of callbacks between disconnected parts of code.

√**Filtered** invocation of callbacks between disconnected parts of code.

√**Obfuscated** invocation of callbacks between disconnected parts of code.

Messenger API

✓ **Messenger** implements this interface:

```
// Messenger Interface
public interface IMessenger
{

    // Subscribe callback to receive a payload
    // Predicate to filter the payload (optional)
    void Subscribe<T>(Action<T> callback, Predicate<T> predicate = null);

    // Unsubscribe the callback from receiving the payload
    void Unsubscribe<T>(Action<T> callback);


    // Publish the payload to its subscribers
    void Publish<T>(T payload);

}
```

Messenger API

✓ **Messenger**

Access to default Messenger instance via:

```
SuperMaxim.Messaging.Messenger.Default
```

✓ **Publish**

```
// Generic Parameter <T> - here is a <Payload> that will be published to subscribers of this type
Messenger.Default.Publish<Payload>(new Payload{ /* payload params */ });

// In most cases there is no need in specifying Generic Parameter <T>
Messenger.Default.Publish(new Payload{ /* payload params */ });

// Generic Parameter <T> - here is a <IPayload> that will be published to subscribers of this type
Messenger.Default.Publish<IPayload>(new Payload{ /* payload params */ });

                    class Payload : IPayload
                    {

                    }
```

Messenger API

√ **Subscribe**

```
// Payload – the type of Callback parameter
// Callback – delegate (Action<T>) that will receive the payload
Messenger.Default.Subscribe<Payload>(Callback);

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

√ **Subscribe with Predicate**

```
// Predicate – delegate (Predicate<T>) that will receive payload to filter
Messenger.Default.Subscribe<Payload>(Callback, Predicate);

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

✓ **Unsubscribe - Variant #1**

```
// Payload – the type of Callback parameter that was subscribed
// Callback – delegate (Action<Payload>) that was subscribed
Messenger.Default.Unsubscribe<Payload>(Callback);

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

✓ **Unsubscribe - Variant #2**

```
// IPayload – the type of Callback parameter that was subscribed
// Callback – delegate (Action<Payload>) that was subscribed
Messenger.Default.Unsubscribe<IPayload>(Callback);

// Payload class implements IPayload interface
private static void Callback(Payload payload)
{
    // Callback logic
}
```

Correct Usage (tips)

- **DON'T** use Messenger if you have a direct access to shared code parts to invoke events/methods in the same module/class.

✓**ALWAYS** ensure that you unsubscribe when you're done with the consuming of payloads.

- **DON'T** publish payloads in endless or in a long running loops.

✓**PREFER** using Filtered subscriptions.

MainThreadDispatcher API

Main Thread dispatcher is responsible for the synchronisation of callbacks between Main and other threads.

✓ **MainThreadDispatcher implements this interface**

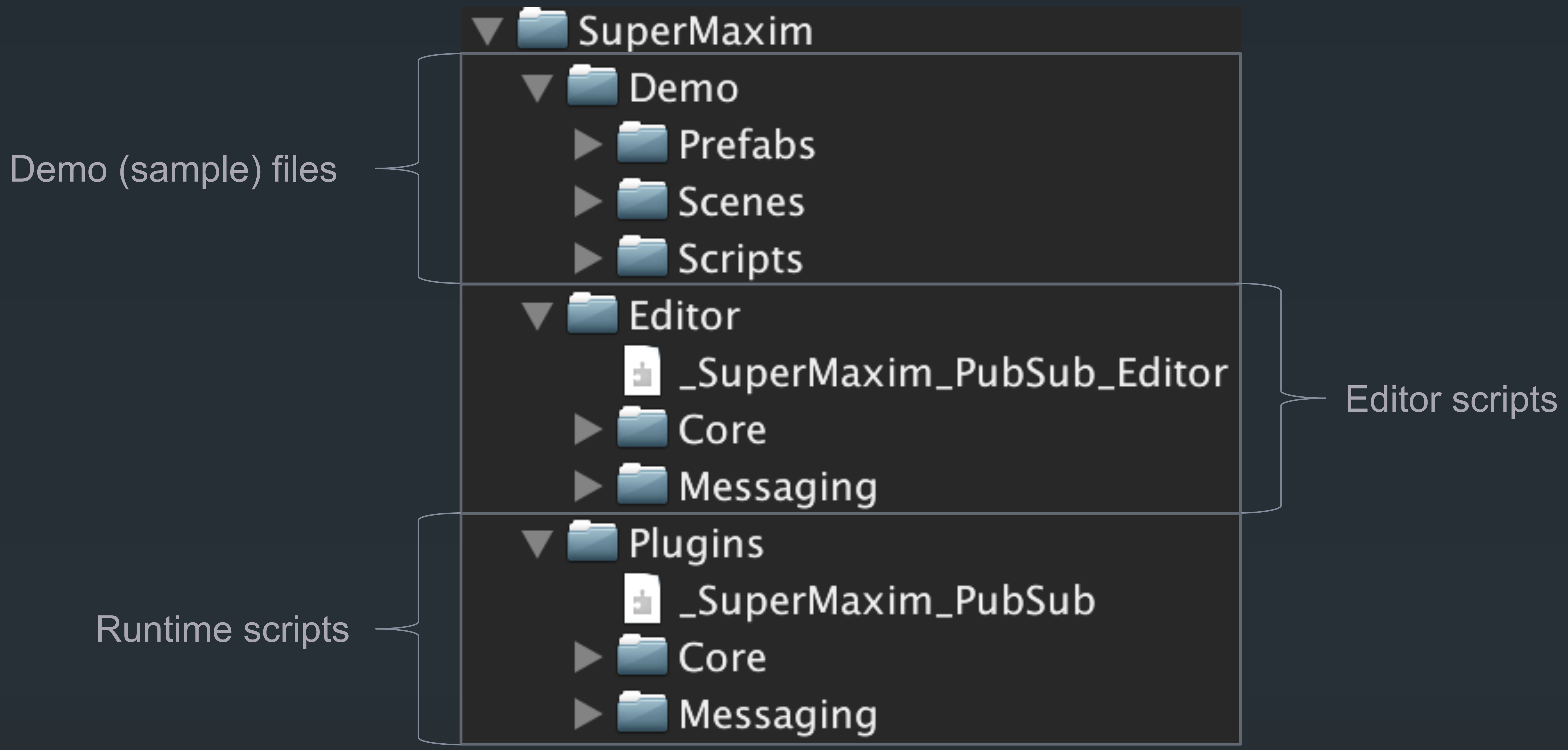
```
public interface IThreadDispatcher
{
    // managed Thread ID
    int ThreadId { get; }

    // dispatch – adds callback delegate into dispatcher’s queue
    // action – delegate reference to method that should be invoked on main thread
    // payload – the data that should be passed to the method/callback
    void Dispatch(Delegate action, object[] payload);
}
```

✓ **Dispatch Method - example**

```
MainThreadDispatcher.Default.Dispatch(Callback, new object[] { payload, state });
```

Package Structure

✓Folders



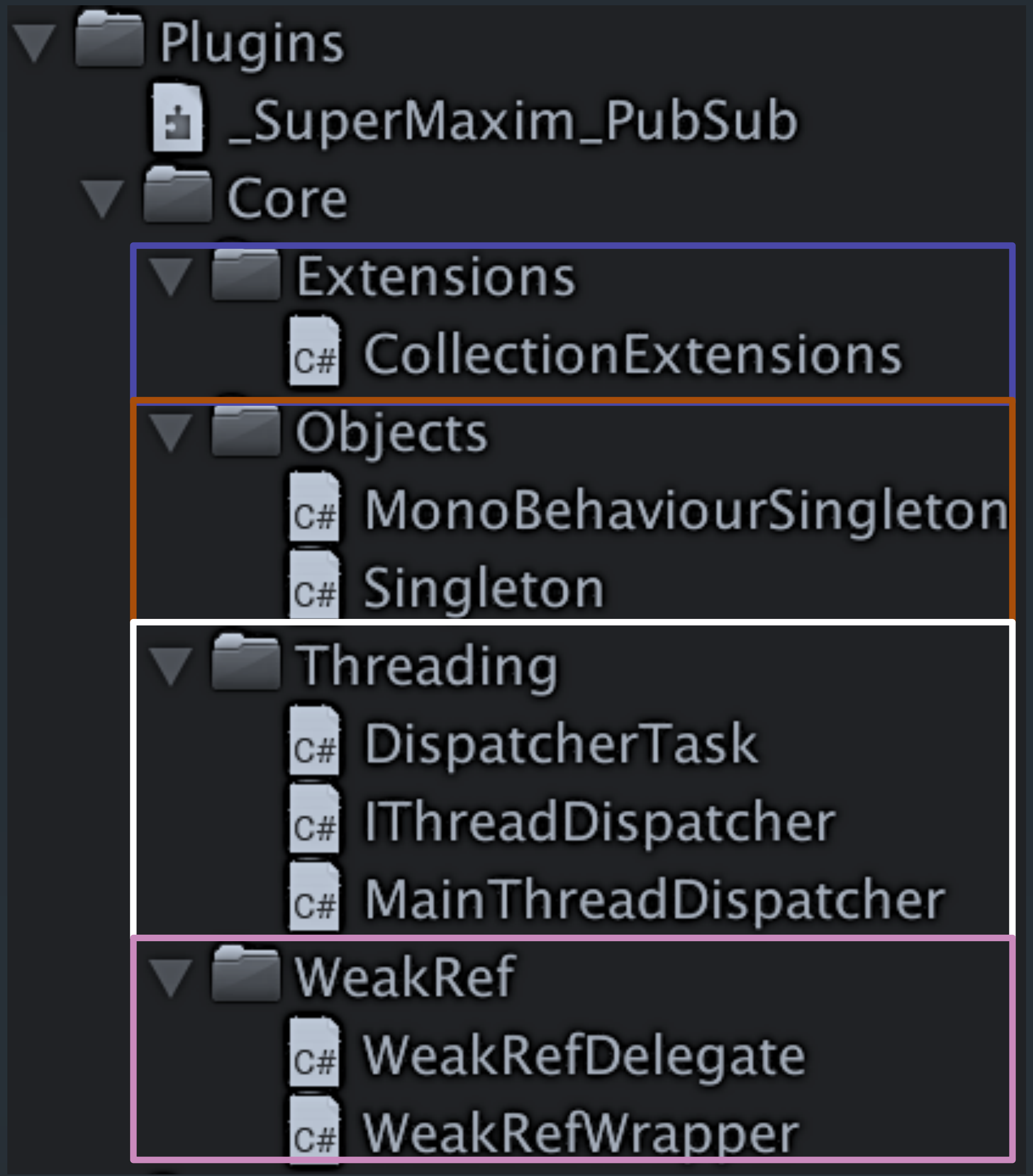
```
▼ SuperMaxim
    ▼ Demo
        ▶ Prefabs
        ▶ Scenes
        ▶ Scripts
    ▼ Editor
          _SuperMaxim_PubSub_Editor
        ▶ Core
        ▶ Messaging
    ▼ Plugins
          _SuperMaxim_PubSub
        ▶ Core
        ▶ Messaging
```

Notes:

Folders with name “Core” contain base classes and scripts that are shared across different modules.
Folders with name “Messaging” contain classes and scripts that are specific for the Messenger.

Package Structure

✓**Plugins / Core**



```
▼ Plugins
    _SuperMaxim_PubSub
  ▼ Core
    ▼ Extensions
        CollectionExtensions
    ▼ Objects
        MonoBehaviourSingleton
        Singleton
    ▼ Threading
        DispatcherTask
        IThreadDispatcher
        MainThreadDispatcher
    ▼ WeakRef
        WeakRefDelegate
        WeakRefWrapper
```

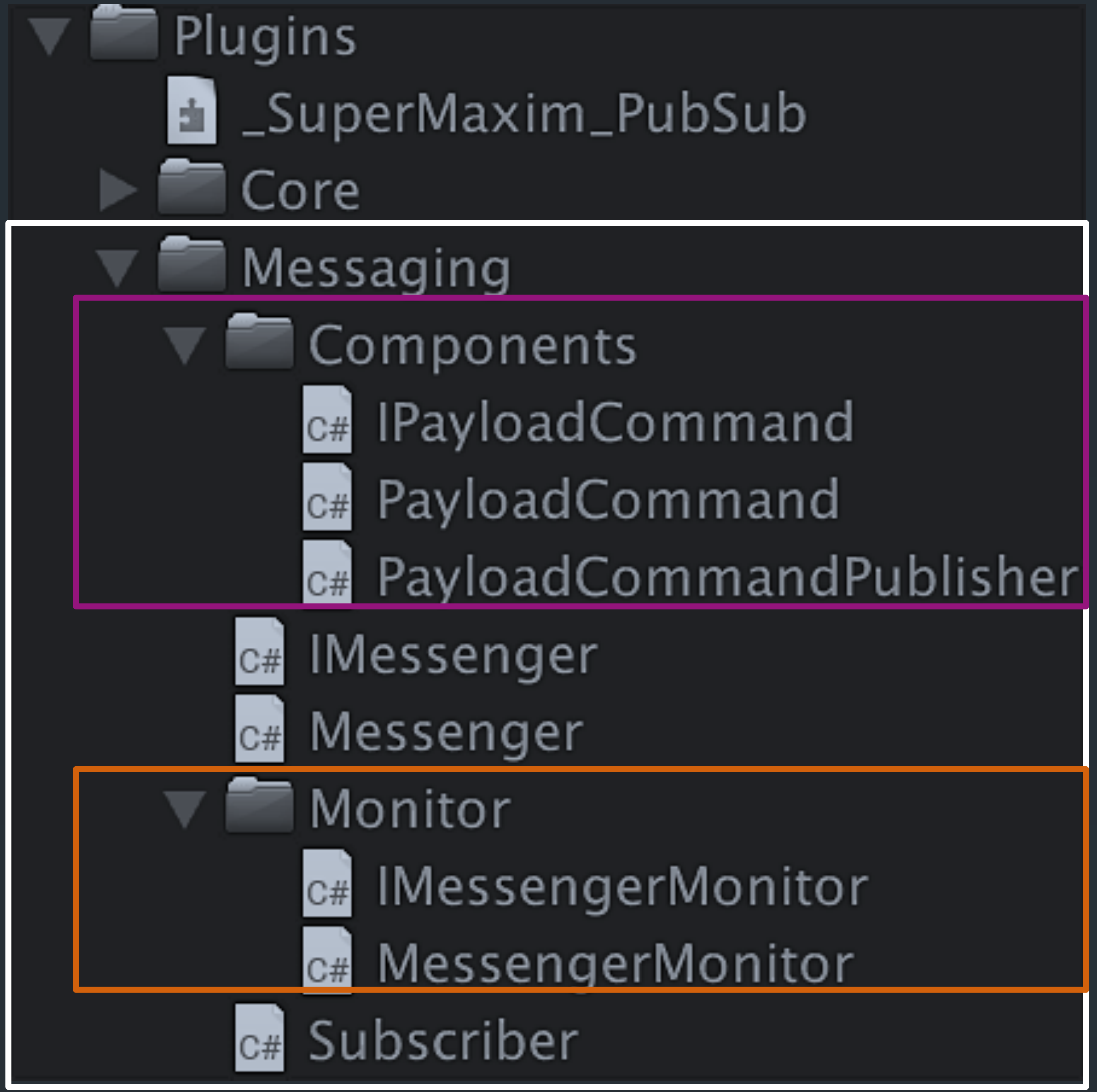
Folder	File/Class/Script	Description
Extensions	CollectionExtensions	Static class with collection extension functions
Objects	MonoBehaviourSingleton	Abstract base class for MB singletons
Objects	Singleton	Abstract base class for classic singletons
Threading	DispatcherTask	Task unit with weak ref. (pointer) to delegate to execute on thread
Threading	IThreadDispatcher	Interface for Thread Dispatcher API
Threading	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)
WeakRef	WeakRefWrapper	Weak reference wrapper (disposable)

Folders:

Extensions – extensions classes
Objects – core base classes
Threading – multithreading related classes/scripts
WeakRef – weak reference handling classes

Package Structure

✓**Plugins / Messaging**



```
▼ Plugins
    _SuperMaxim_PubSub
  ▶ Core
  ▼ Messaging
    ▼ Components
        IPayloadCommand
        PayloadCommand
        PayloadCommandPublisher
      IMessenger
      Messenger
    ▼ Monitor
        IMessengerMonitor
        MessengerMonitor
      Subscriber
```

Folder	File/Class/Script	Description
Messaging / Components	IPayloadCommand	Interface for Payload Command – a generic payload with Id and Dic. of string key value pair Data
Messaging / Components	PayloadCommand	Implementation of generic Payload Command, implementing IPayloadCommand interface
Messaging / Components	PayloadCommandPublisher	MB Script with Publish method that publishes a generic PayloadCommand – can be used with UGUI events or other ways
Messaging	IMessenger	Interface for Messenger API
Messaging	Messenger	Implementation of IMessenger interface with Pub-Sub logic; contains container of subscribers; singleton that is accessible via Messenger.Default
Messaging	Subscriber	Represents single item for subscribers' container. Including weak ref. pointing to callback method and weak ref. pointing to predicate.
Messaging / Monitor	IMessengerMonitor	Interface for MessengerMonitor API
Messaging / Monitor	MessengerMonitor	Debugging and monitoring tool for Messenger

Folders:

Components – useful unity components
Monitor – debugging and monitoring tools

Unit Tests

Coverage:

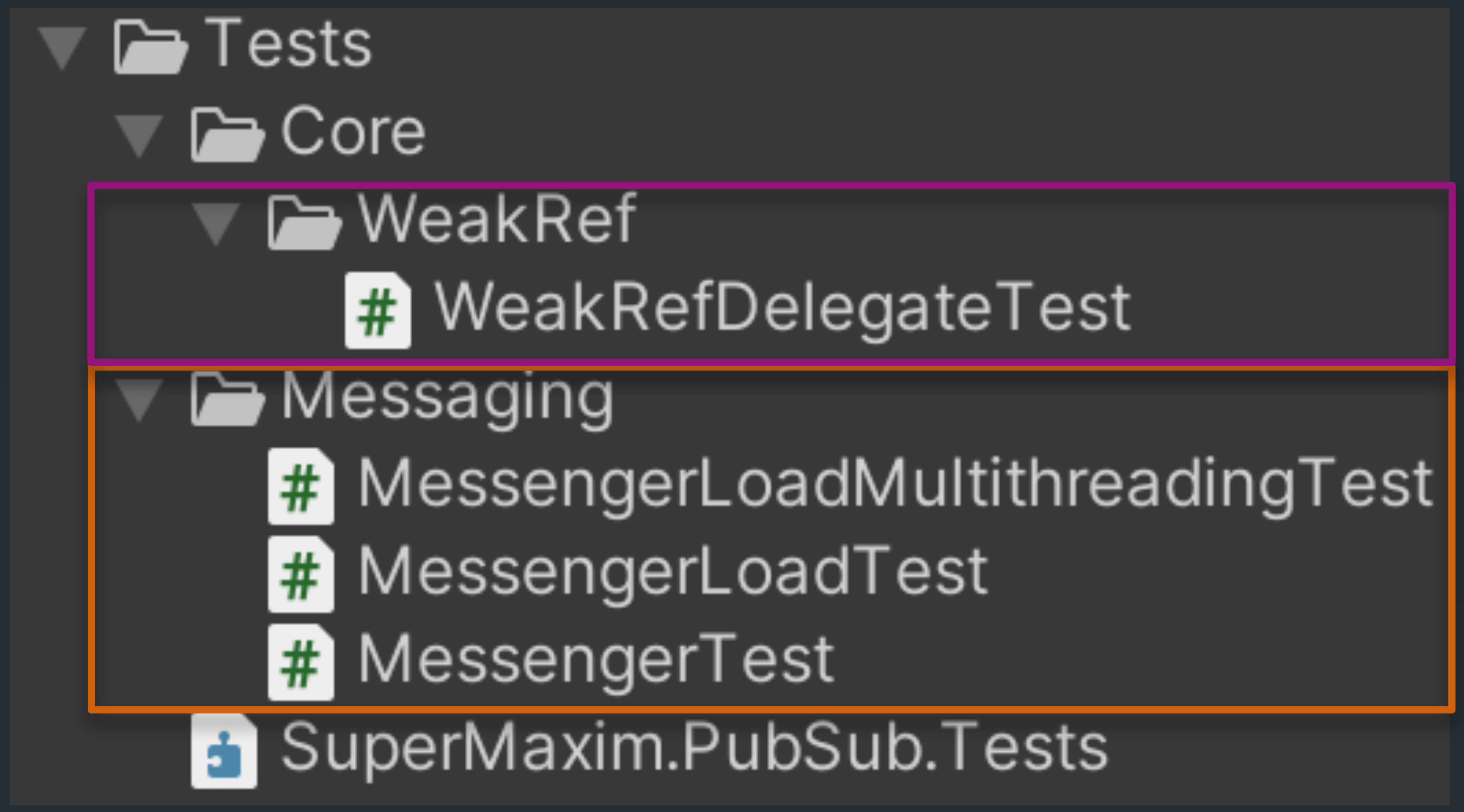
✓Messenger Tests

- Functional
- Load
- Multithreading

✓Weak Reference Tests

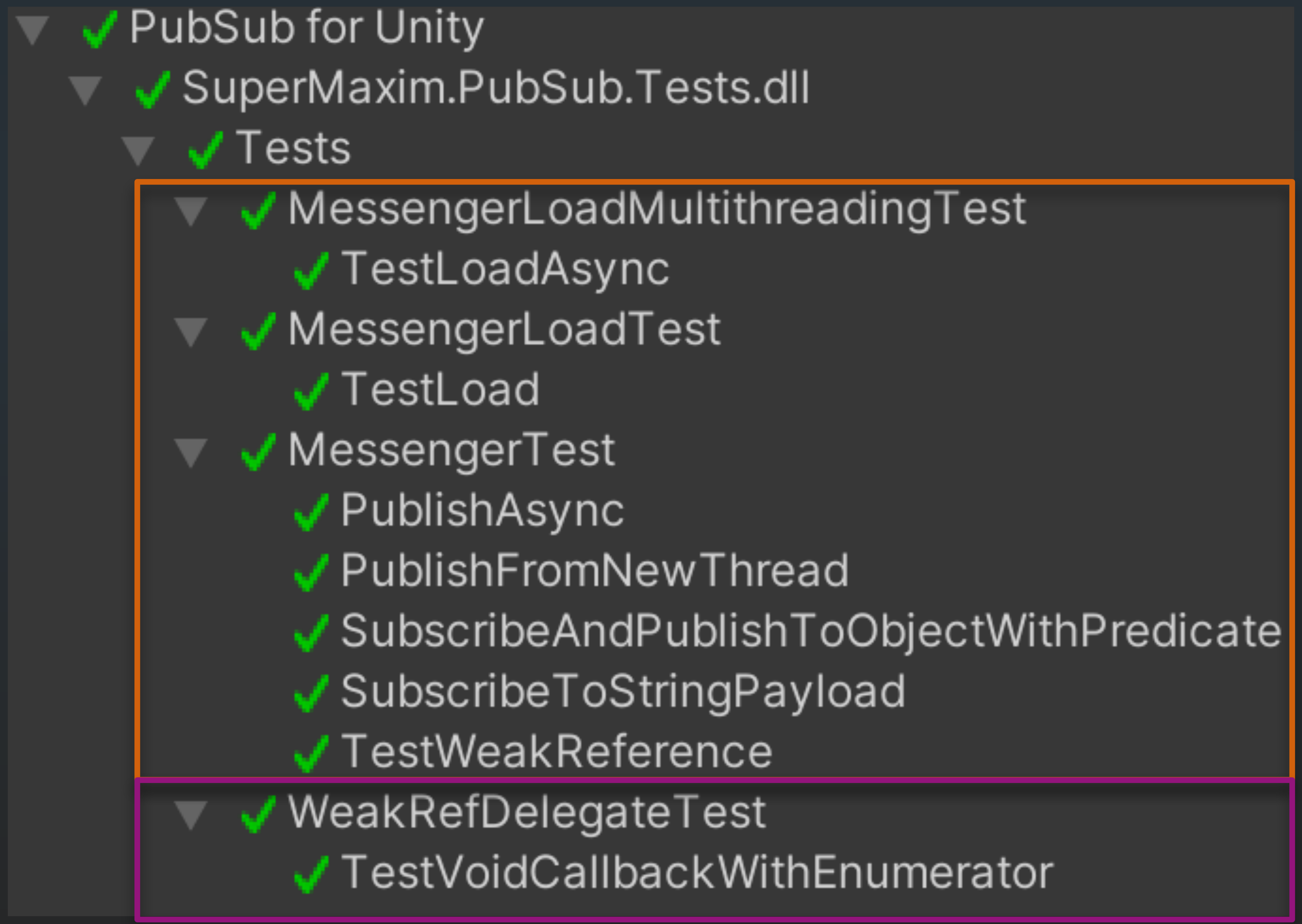
- Functional

Tests Folder:



```
Tests
  Core
    WeakRef
      WeakRefDelegateTest
    Messaging
      MessengerLoadMultithreadingTest
      MessengerLoadTest
      MessengerTest
    SuperMaxim.PubSub.Tests
```

Playmode Tests:



```
✓ PubSub for Unity
  ✓ SuperMaxim.PubSub.Tests.dll
    ✓ Tests
      ✓ MessengerLoadMultithreadingTest
        ✓ TestLoadAsync
      ✓ MessengerLoadTest
        ✓ TestLoad
      ✓ MessengerTest
        ✓ PublishAsync
        ✓ PublishFromNewThread
        ✓ SubscribeAndPublishToObjectWithPredicate
        ✓ SubscribeToStringPayload
        ✓ TestWeakReference
      ✓ WeakRefDelegateTest
        ✓ TestVoidCallbackWithEnumerator
```

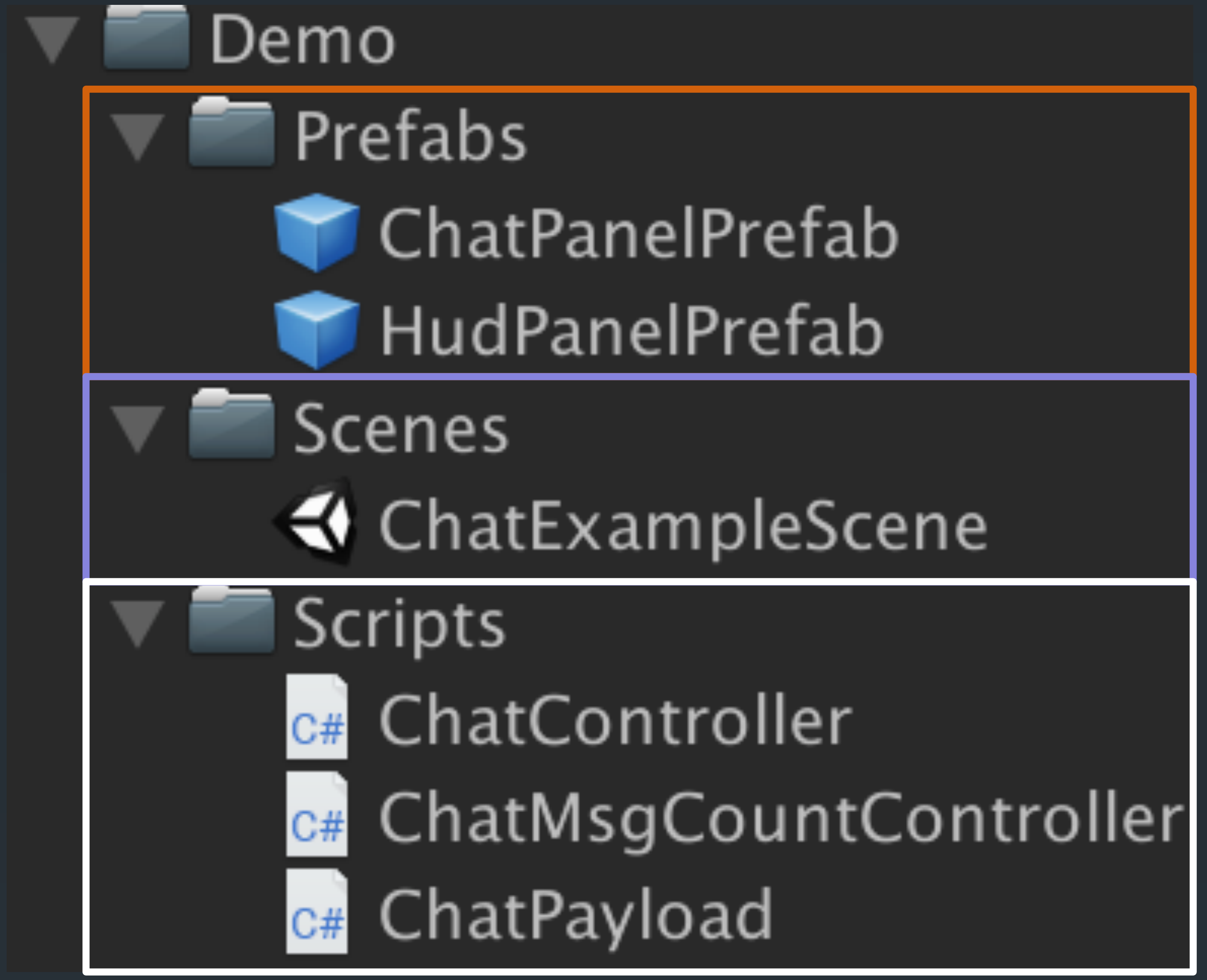
DEMO – Game Chat

Goals:

- ✓Show how to use Messenger
- ✓Use real world example – Chat between players
- ✓Include examples of message filtering and multithreading
- ✓Show “Best Practices” approach in implementation

Demo Structure

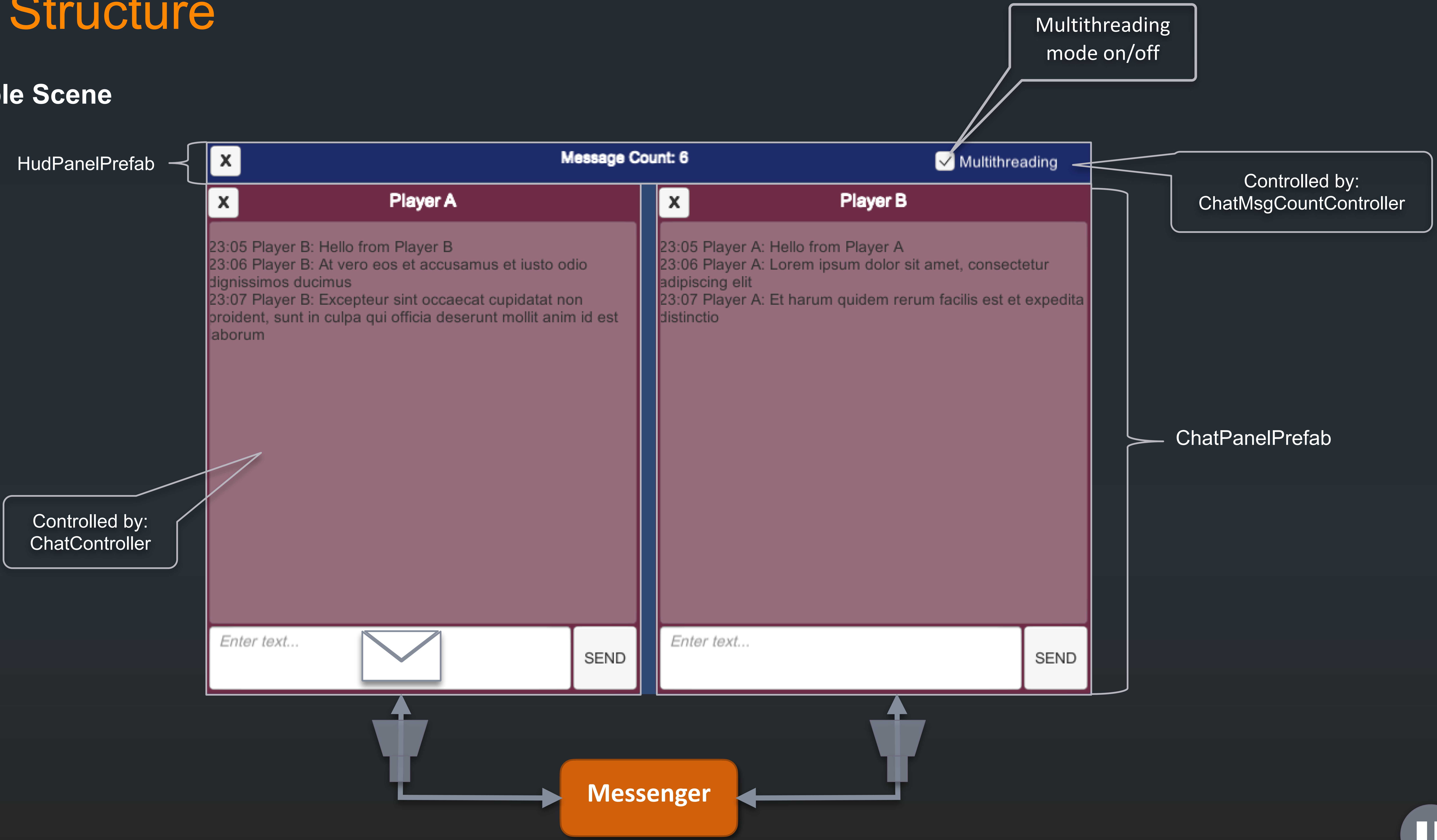
✓Folders / Files


Demo
Prefabs
ChatPanelPrefab
HudPanelPrefab
Scenes
ChatExampleScene
Scripts
ChatController
ChatMsgCountController
ChatPayload

Folder	File/Class/Script	Description
Prefabs	ChatPanelPrefab	Prefab that contains all elements for player chat UI (see prefab in editor)
Prefabs	HubPanelPrefab	Prefab that contains top HUD elements (see prefab in editor)
Scenes	ChatExampleScene	Example scene with chat UI (see scene in editor)
Scripts	ChatController	Script that controls ChatPanelPrefab (message input, publish and subscribe)
Scripts	ChatMsgCountController	Script that 'listens' to chat messages, counts them and displays count in HUD
Scripts	ChatPayload	Payload class that is used to pass messages between chat panels

Demo Structure

✓**Sample Scene**



23:05 Player B: Hello from Player B
23:06 Player B: At vero eos et accusamus et iusto odio dignissimos ducimus
23:07 Player B: Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

23:05 Player A: Hello from Player A
23:06 Player A: Lorem ipsum dolor sit amet, consectetur adipiscing elit
23:07 Player A: Et harum quidem rerum facilis est et expedita distinctio

Source Code and Materials

https://github.com/supermax/pubsub

Subscribe https://tinyurl.com/supermaxim

Thanks for watching :)