Foundation Messenger (v4.0) 3/6/2015

Nicholas Ventimiglia | AvariceOnline.com

The Messenger is a static application service for relaying events (messages) in a loosely coupled way. Any object may be "Published" through the messenger and handled by methods (or coroutines) through the app.

- Support for objects, structs, enums and interfaces
- Support for caching. This allows for publishing then subscribing and then receiving a message in that order. This is useful for one time messages like authentication.
- Use the IMessengerObject interface to add a Publish() extension method to your message.
- Supports [Subscribe] annotation. Using this annotation unlocks the Subscribe(object) helper method. This helper method allows for the subscribe of all decorated methods automagically.
- Support for coroutines event handling.

Setup

Make sure the files exist in your project.

Use

Subscribing

Subscribing is the wiring of methods to the messenger. Methods must take a single argument of the message type they are handling. When a message of the appropriate type is publish the method will be called.

```
public class MyHandler: MonoBehaviour
{
    void Awake()
    {
        // Subscribe using the [Subscribe] annotation
```

```
        Messenger.Subscribe(this);

        // Subscribe manually
        Messenger<MessageType>.Subscribe(MyHandler);

        // Subscribe manually
        Messenger<MessageType>.SubscribeCortoutine(MyCoroutineHandler);
    }

    void OnDestroy()
    {
        // Unsubscribe using the [Subscribe] annotation
        Messenger.Unsubscribe(this);

        // Unsubscribe manually
        Messenger<MessageType>.Unsubscribe(MyHandler);

        // Unsubscribe manually
        Messenger<MessageType>.UnsubscribeCortoutine(MyCoroutineHandler);
    }

    [Subscribe]
    public void MyHandler(MessageType arg);

    [Subscribe]
    public IEnumerator MyCoroutineHandler(MessageType arg);
}
```

Publishing

Publishing is the sending a message to subscribed members.

```
public class MyPublisher: MonoBehaviour
{
    void Awake()
    {
        var message = new MessageType();

        // publish using the IMessengerObject Extension Method
        message.Publish();

        // publish Manually
        Messenger.Publish(message);
    }
```

```
}
```

Caching

Messages may be cached. When cached the message will be saved and issued to late subscribers.

```
// Cache the message
[CachedMessage]

// or

// Cache the message and clear the cache of old messages of the same type
[CachedMessage(OnePerType=true)]

public class MessageType : IMessengerObject
{

}
```

More

Part of the Unity3d Foundation toolkit. A collection of utilities for making high quality data driven games. http://unity3dFoundation.com

- **Tasks** : An async task library for doing background work or extending coroutines with return results.
- **Messenger** : Listener pattern. A message broker for relaying events in a loosely coupled way. Supports auto subscription via the [Subscribe] annotation.
- **Terminal**: A in game terminal for debugging !
- **Injector**: Service Injector for resolving services and other components. Supports auto injection using the [Inject] annotation
- **DataBinding** : For MVVM / MVC style databinding. Supports the new uGUI ui library.
- **Localization** : Supports in editor translation, multiple files and automatic translation of scripts using the [Localized] annotation.
- **Cloud** : Parse-like storage and account services using a ASP.NET MVC back end. Need to authenticate your users? Reset passwords with branded emails? Save high scores or character data in a database? Maybe write your own authoritative back end? This is it.
- **Lobby** : The ultimate example scene. Everything you need to deploy for a game, minus the

actual game play.

Donations

I accept donations via papal. Your money is an objective measure of my self esteem.