Event Minion

Hello!

This is a documentation for a Unity asset Event Minion made by Salday.
You can purchase the asset on assetstore.unity3d.com

For support support@salday.com

Content

The purpose of asset

Many times software developers face a problem of tightly coupled modules where one module depends on existence of another, and thus, is only capable of interacting with a fixed set of predefined components.

To solve this issue, we created Event Minion - system, which could dispatch published events to any registered handler accepting particular event argument. This means, that your event sources should no longer contain references to other components, only to event bus.

This creates a clean separation, between event sources and handlers, and allows to build handler pipelines dynamically in runtime, rather than compile time. So in theory you can even load third-party libraries adopted for this Event Bus and insert their handlers into the pipeline. For example, this behavior can be used to implement plugin systems.

Core concepts

Event bus

The main class what you are going to be dealing with.

Accepts and registers proxy objects, which contain public instance methods marked with Handler attribute.

It is also a place, where all the events get published and later distributed among compatible handlers.

Handler attribute

Every handler needs to be marked with this attribute, it also allows to specify priority: from lowest to highest.

Creating Event argument class

It is the class, which is going to hold any event data. Can be declared like so:

```
public class ChatEvArgs : CancelableEventBase
{
   public string Message { get; set; }
}
```

This one is inherited from `CancelableEventBase` which has Boolean property `Canceled`. This serves as a way to tell other handlers further down the chain, that event was canceled due to inappropriate data, or successfully handled.

There is also a non-cancelable version – `EventBase`.

Core concepts

Proxy object

An instance of class, which contains public instance methods marked with Handler attribute. They will be called, once event bus receives event with matching argument type. Only exact types are supported! e.g. if methods signature looks like this:

```
[Handler(HandlerPriority.Low)]
    public void ChatMessageHandler(ChatEventArg argument)
    {
                            //Handler code goes here
                            //.......
    }
```

this method will be only called when event bus tries to dispatch `ChatEventArg` type of event.

So, proxy object can contain one or more handler methods (even with the same type of accepted parameter).

Proxy classes can either be plain c# objects, or, if you're planning to use them in Unity Editor in a drag-and-drop manner, should be derived from `UnityEventProxyBase`

Core concepts

Registration

A process of registering proxy in the event bus. Can be performed in code like so:

```
ISubscription sub = eventBus.RegisterSubscription(proxyObject);
```

Notice the subscription, can be used to unsubscribe proxy from event bus, or temporarily deactivate it.

If you are using `UnityAdoptedEventBus` and proxies derived from `UnityEventProxyBase`

everything is even more simple – just drag appropriate proxy component into Event Bus proxy array in the inspector.

Publishing Events

Once everything is configured, it's time to publish our first event. This is done in the code like so:

```
eventBus.Publish(eventArgumentObject);
```