Messages Between Objects

TpMessages has a secondary, separate typed-subscription system using the same MessagePacket scheme.

Recall that messages to TPT tiles requires implementation of explicit interfaces. Here, you need to subscribe to a Type of message packet.

`public static ulong RegisterMessageTarget<T>(Object? parent, Action<object> target) where T : MessagePacket<T>`

You register your target to a particular variety of MessagePacket (which can be any arbitrary blob of data as long as the class is derived from MessagePacket). If the parent is UnityEngine.Object such as a Monobehaviour then null-checks ensure that if the Object becomes null then it is automatically deregistered. Leave parent null when using a static class instance (which can't become null);

The target is an Action which receives a c# object which you have to cast back to the Type of the MessagePacket.

For example (not complete and just for illustration):

```
public class Example :  Monobehaviour
{

        public void Start()
        {
            Messages.RegisterMessageTarget<PositionPacket>(this, PlayerMoveCompleteTarget));
            return;



            void PlayerMoveCompleteTarget(object obj)
            {
                var packet = (PositionPacket) obj;
                Debug.Log($"Player moved to {packet.m_Position});
             }

        }
```

If this Monobehaviour's parent GameObject becomes null then the 'registration' is auto-deleted.

Calls to RegisterMessageTarget return a ulong ID which can be used to unregister.

There are two Unregister methods:

```
public static bool UnregisterMessageTarget(Type parentType, ulong id)
public static bool UnregisterMessageTarget(ulong id)
```

The first method is faster. Both return false if the operation failed.

Sending a Message

`public static bool SendToTargets<T>(MessagePacket<T> packet) where T : MessagePacket<T>`

This generic method sends the message packet to all targets for that particular message packet Type. In the context of the above example:

```

var packet = new PositionPacket();
packet.m_Position = new Vector3Int(1,2,3);
Messages.SendToTargets(packet);
```

This is a simple but powerful way to have event-driven messages controlling your gameplay systems without lots of dependencies.

Revision #7
Created 2026-07-09 15:53:45 UTC by Vonchor
Updated 2026-07-09 17:05:46 UTC by Vonchor