

# Zone Actions

Zone Actions are similar to Event Actions but they're not automatically invoked in the same way as EventActions.

These are normally invoked by MessageTarget methods: the target for SendMessage.

Here's an example from TpSlideShow (edited for brevity):

```
protected void Access_ActionToTilePacket(ActionToTilePacket
sentPacket)

{

    if (sentPacket.SourceInstance ==
this)

        return; //avoid
recursion.

    if
(!m_AcceptsClicks)

return;

    if (sentPacket.Delay !=
0)

{

        TpLib.DelayedCallback(this,()=>ChangeSlide(SlideIndex ==
0,false,false),"DelayATTP_Slideshow",
sentPacket.Delay);

    if(m_ZoneAction)
```

```

        m_ZoneAction.Exec(this, this.GetType(), m_ZoneTargetTag,
ObjectPacket.EventsPropagationMode.ZoneEvents,"",SlideIndex);

    }

    else

        ChangeSlide(slideIndex == 0,
true,sentPacket.PostEvent);

    }

```

So this MessageTarget is a recipient of Messages from TpActionToTile which is the S.O. that converts clicks to tile positions. Having found one, it sends this message and (if there's a delay) invokes the ZoneActions's Exec method.

The Exec method declaration:

```

public virtual bool Exec(TilePlusBase
sourceInstance,
                                Type?
targetType,
                                string
tagString,
                                ObjectPacket.EventsPropagationMode eventPropagationMode =
ObjectPacket.EventsPropagationMode.None,
                                string
optionalString
                                =
"",
                                int
optionalInt
                                =
0,
                                bool
optionalBool
                                =
false,
                                object?
optionalObject
                                =
null)

```

As you can see, the parameters have some specific information like:

- targetType (here passed-in as the Type of the invoking TPT tile)
- tagString (here passed-in as a specific value from the invoking TPT tile)

and some general parameters:

- optionalString
- optionalInt
- optionalBool
- optionalObject

These can be whatever you want. In this example, optionalInt is used to pass the current `SlideIndex` which specifies which sprite is being displayed.

It's probably obvious that the invoking tile has to know a what the ZoneActions needs *and* the ZoneAction has to be designed for a specific purpose: there's an unavoidable dependency.

To be clear, a TPT tile could do whatever the ZoneAction does within its own code. The idea of Zone Actions is that often there will be actions that different varieties (Types) of TPT tiles perform. Using a Scriptable Object plugin like this means you don't have to have similar code peppered throughout your project, improving maintainability and making debugging easier.

Similar to Event Actions, avoid having any state variables in the Event Action Scriptable Object since it may be re-used among several different tiles.

Zone Actions may relay messages to other tiles. This is commonly used by the `UI` tiles but there's no restriction. If you examine the base-class Zone Action (`TpTileZoneAction.cs`) you'll see that what it does is relay a message to other tiles within the `BoundsInt` region of whatever tile invoked the Zone Action, with filtering.

The filter has this code:

```
bool FilterFunc(ITpMessaging<ObjectPacket> pkt, TilePlusBase tpb)
{
    if (!tpb)
        return false;

    // ReSharper disable once Unity.NotNullPatternMatching
    if(tpb is not IZoneActionTarget receiver )
        return false;
    if(!receiver.AcceptsZoneAction)
        return false;

    if (!checkTag)
        return true;
```

```
//this can be slow if you have many tags.  
(var count, var tags) = tpb.TrimmedTags;  
return count != 0 && ((IList)tags).Contains(tagString);
```

The filter rejects TPT tiles that don't implement `IZoneActionTarget`. That interface has one property: `AcceptsZoneAction` which is simply a way for t TPT tile to say "leave me alone."

The the filter checks for a tag match and discards TPT tiles without matching tags.

---

Revision #7

Created 22 June 2025 19:03:15 by Vonchor

Updated 12 July 2025 14:00:01 by Vonchor