Removing an Event from a WinForm ListView control using reflection

The objective was to remove an event from a WinForm control that we don't have the source code for (and can't recompile and remove the event on directly on the source code).

This problem happened originally when dynamicaly consuming Cat.NET's guis (outside VisualStudio) and there were a couple controls (like a ListView) that hooked event handlers that triggered functions that had DTE dependencies (which triggered an exception since we were running Cat.NET outside visualstudio (and the DTE2 object was null).

There doesn't seem to be an easy way to do this (and google didn't find a good solution) so using O2's powerful reflection APIs I was able to find a solution which is now available as these extensionmethods:

```
//for a UserControl (in fact any control that implements System.ComponentModel.Component)
var userControl = new UserControl();
//we can get the current mapped event handlers
userControl.eventHandlers();
//its signature
userControl.eventHandlers_MethodSignatures();
//remove one by using the static field name
userControl.remove_EventHandler("EVENT_SELECTEDINDEXCHANGED");
//or use this one specifically mapped to the SelectedIndexChanged event
userControl.remove_Event_SelectedIndexChanged
```

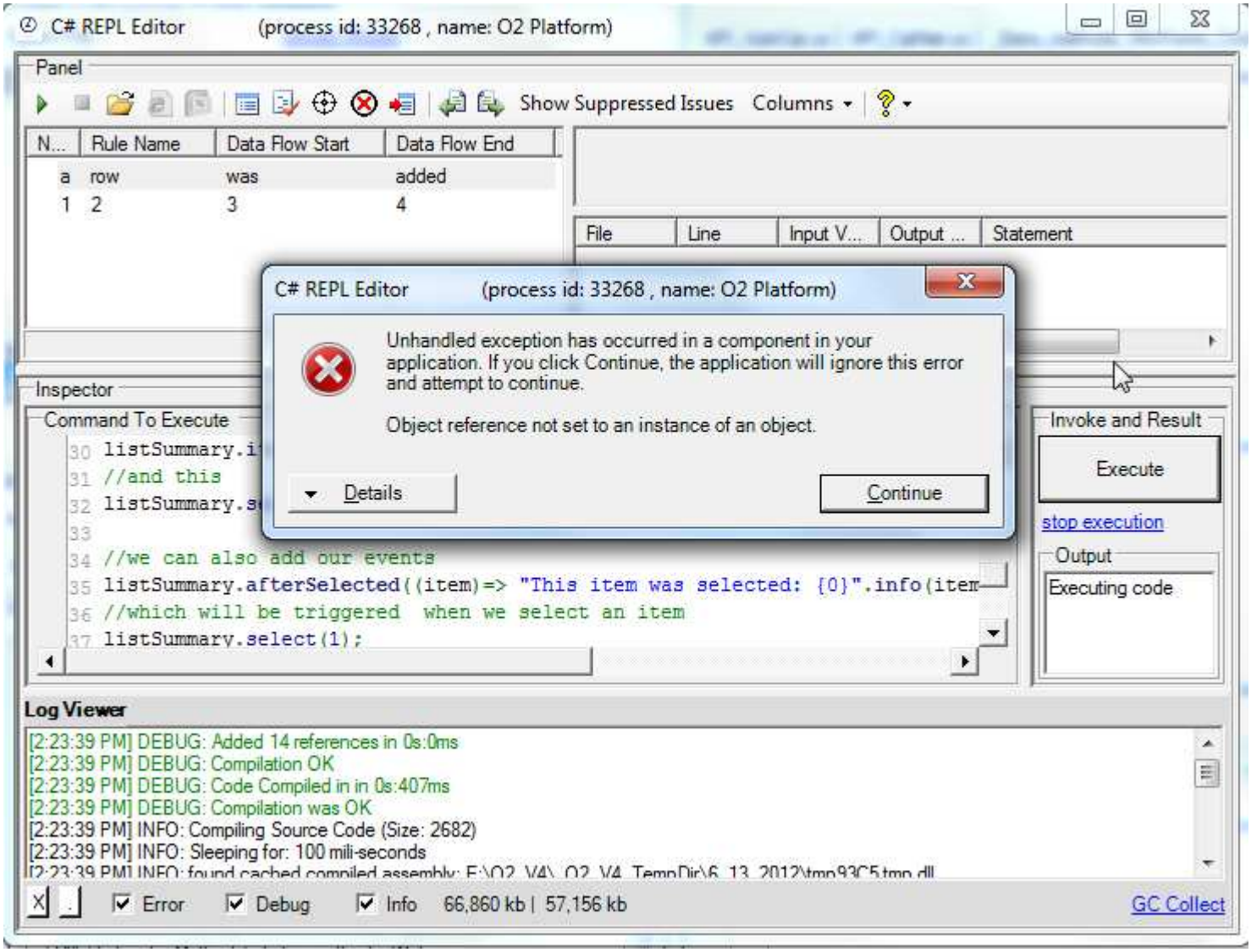
The next scripts shows how these extension methods were created

First version of script that removed the handler

```
//var topPanel = "Remove event PoC".popupWindow();
var topPanel = panel.clear().add_Panel();
//create an instance of Cat.Net SummaryView control
var summaryView = topPanel.clear().add_Control<SummaryView>();
//get the private field _lvSummary
var listSummary = (ListView)summaryView.field("_lvSummary");
//add a couple items to the list
listSummary.add_Row("a","row","was","added")
                .add_Row("1","2","3","4");

//with the event still in place we get an exception when a row is selected, which can be triggered
using:
listSummary.items().first().select();

//SOLUTION: remove the original SelectedIndexChanged event

//this is a generic object used as a key for the SelectedIndexChanged event
var listView = typeof(System.Windows.Forms.ListView);
var EVENT_SELECTEDINDEXCHANGED =  listView.ctor().field("EVENT_SELECTEDINDEXCHANGED");

//get private 'Events' property
var events = (EventHandlerList)listSummary.prop("Events");
//invoke private method 'find' in order to get the  SelectedIndexChanged event entry
var listEntry = events.invoke("Find", EVENT_SELECTEDINDEXCHANGED);
//get the private field 'handler'
var handler = (EventHandler)listEntry.field("handler");
//now that we have the EventHandler object we can remove it normaly
listSummary.SelectedIndexChanged -= handler;

//with the event removed, this will now work
listSummary.items().first().select();
//and this
listSummary.select(2);

//we can also add our events
listSummary.afterSelected((item)=> "This item was selected: {0}".info(item.values().toString()));
//which will be triggered  when we select an item
listSummary.select(1);
```

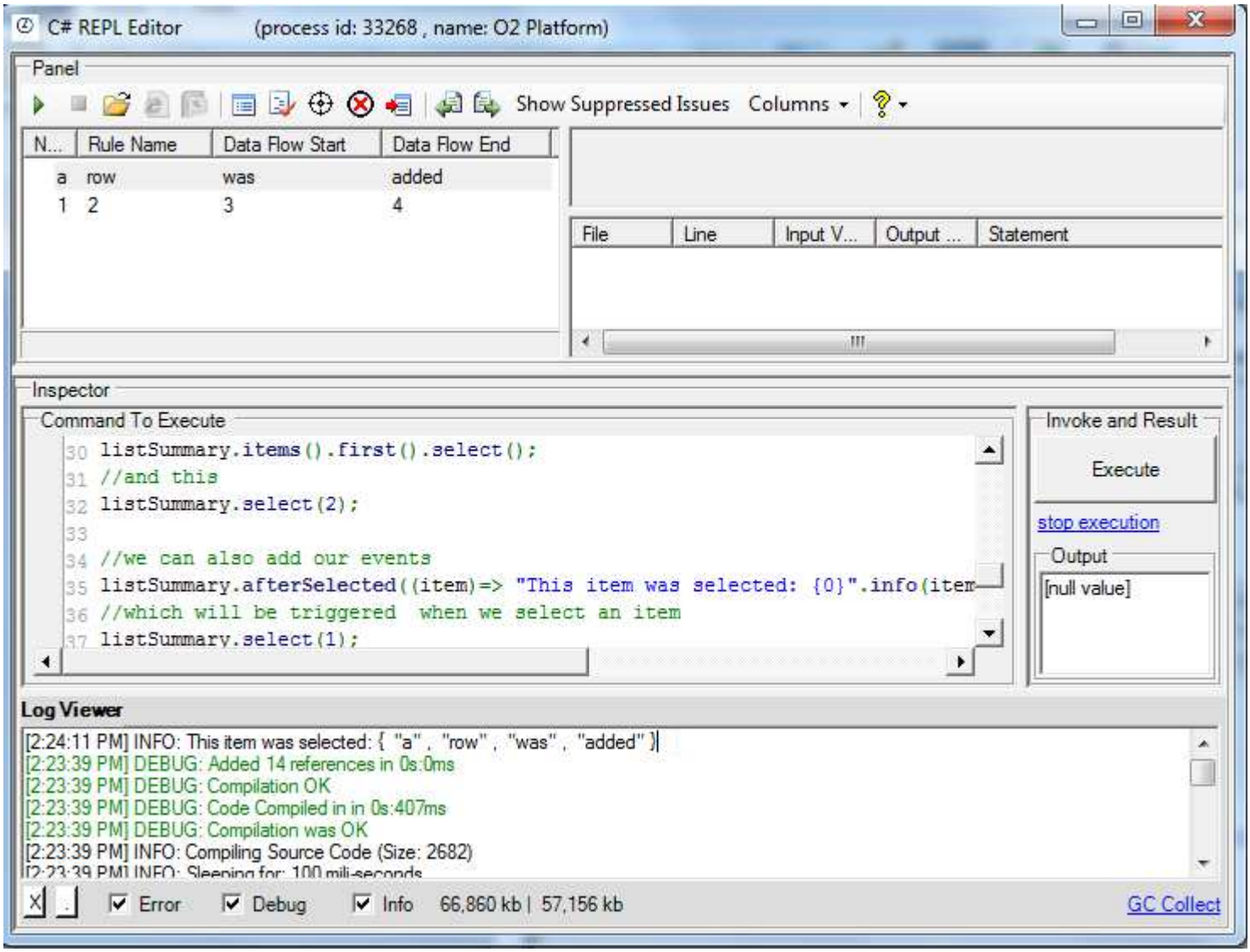
```
//using System.ComponentModel
//using Microsoft.ACESec.CATNet.UI
//O2Ref:CatNet_1.1/SourceDir/Microsoft.ACESec.CATNet.UI.VSAddIn.dll
```

error we get before the event is removed:



What happen after even is removed

Note how after the event was removed (and the new one added) we get the selected row values in the log viewer:



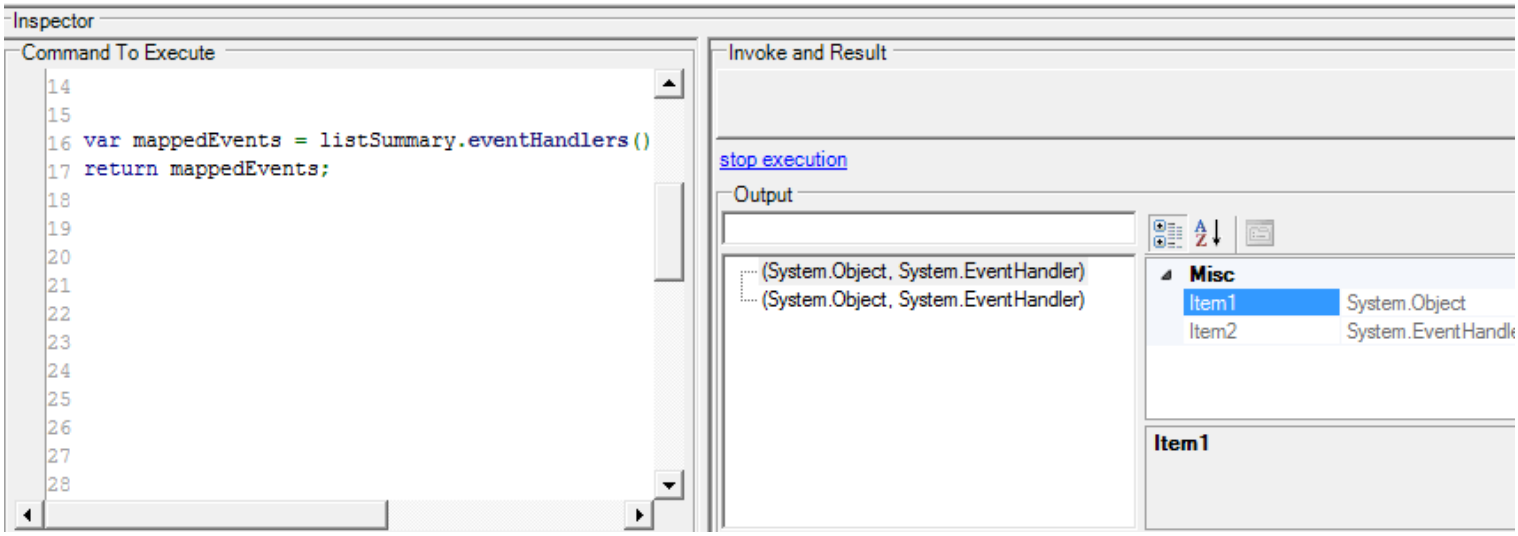
Next step is to refactor the code using a couple extension methods,

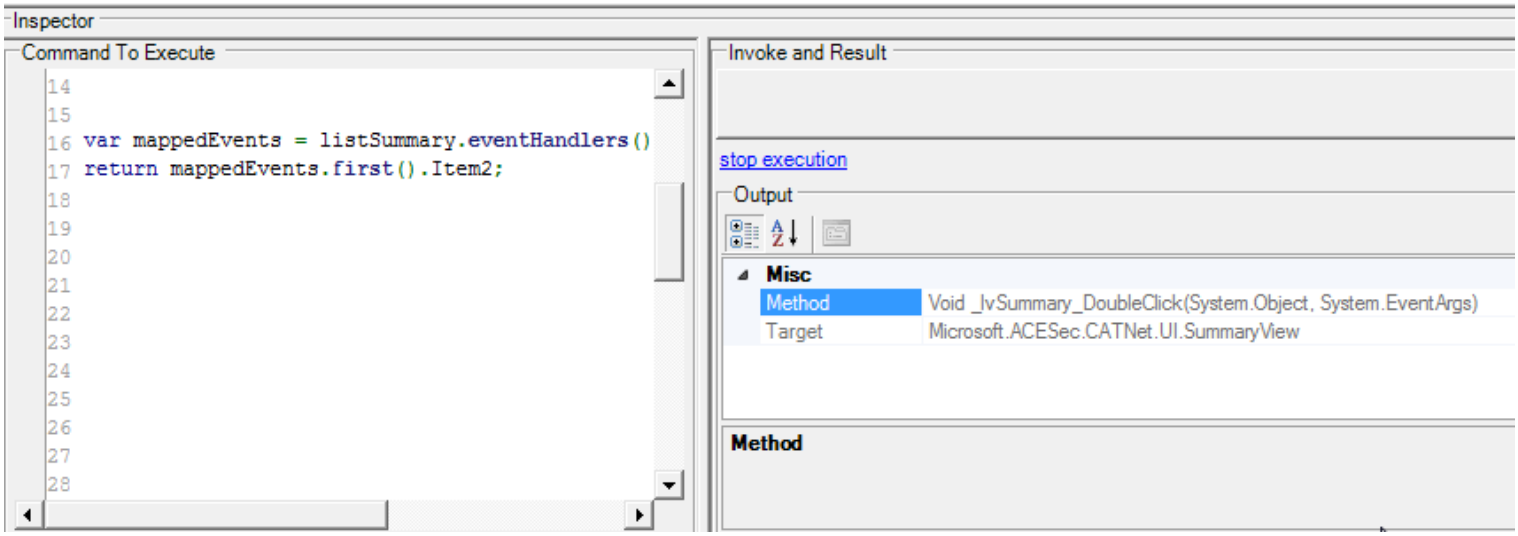
The objective is to simplify this code:

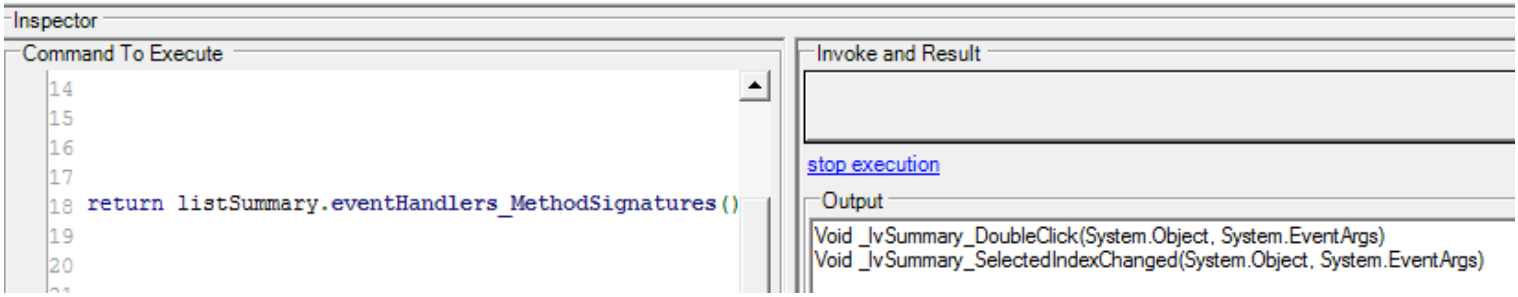
```
//this is a generic object used as a key for the SelectedIndexChanged event
var listView = typeof(System.Windows.Forms.ListView);
var EVENT_SELECTEDINDEXCHANGED =  listView.ctor().field("EVENT_SELECTEDINDEXCHANGED");

//get private 'Events' property
var events = (EventHandlerList)listSummary.prop("Events");
//invoke private method 'find' in order to get the  SelectedIndexChanged event entry
var listEntry = events.invoke("Find", EVENT_SELECTEDINDEXCHANGED);
//get the private field 'handler'
var handler = (EventHandler)listEntry.field("handler");
//now that we have the EventHandler object we can remove it normaly
listSummary.SelectedIndexChanged -= handler;
```

We start by creating an helper method to list all current events:

```
var mappedEvents = listSummary.eventHandlers();
return mappedEvents;
```



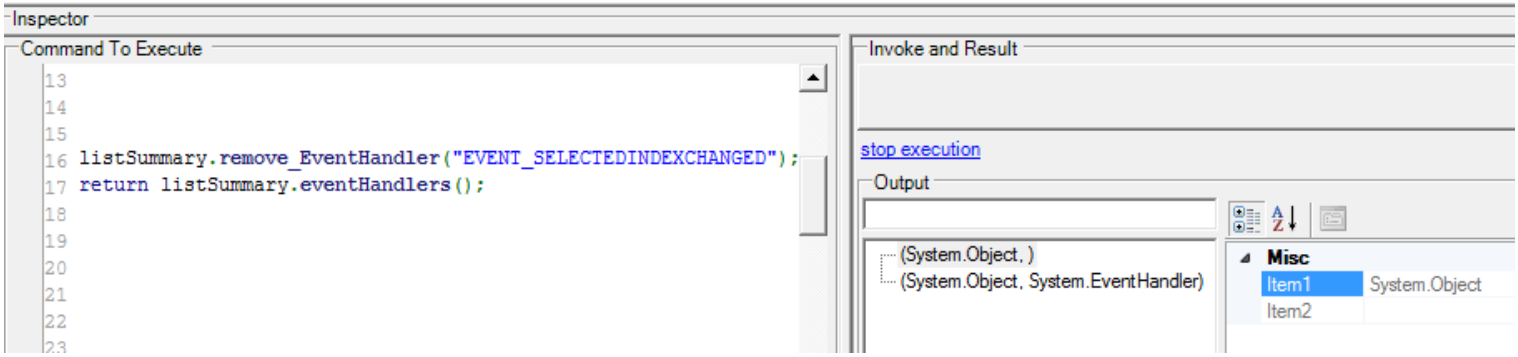




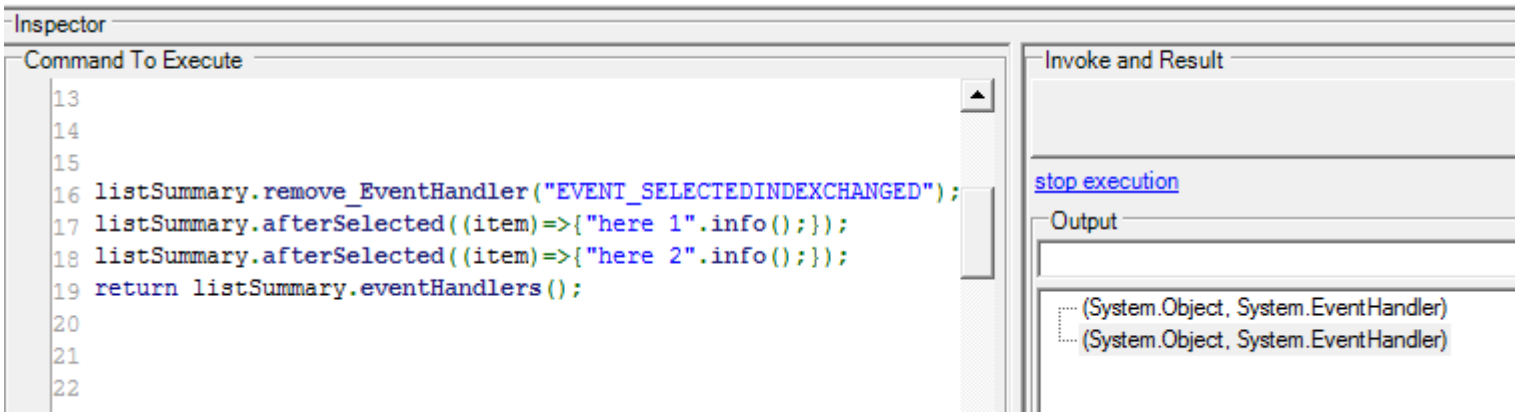
And then one to remove a specific event:

```
listSummary.remove_EventHandler("EVENT_SELECTEDINDEXCHANGED");
```

Note that the handler value (Item2 from Tuble) of the Event is now null (altough it doesn't seem to have major side effects)



Adding our own event will restore the handler:



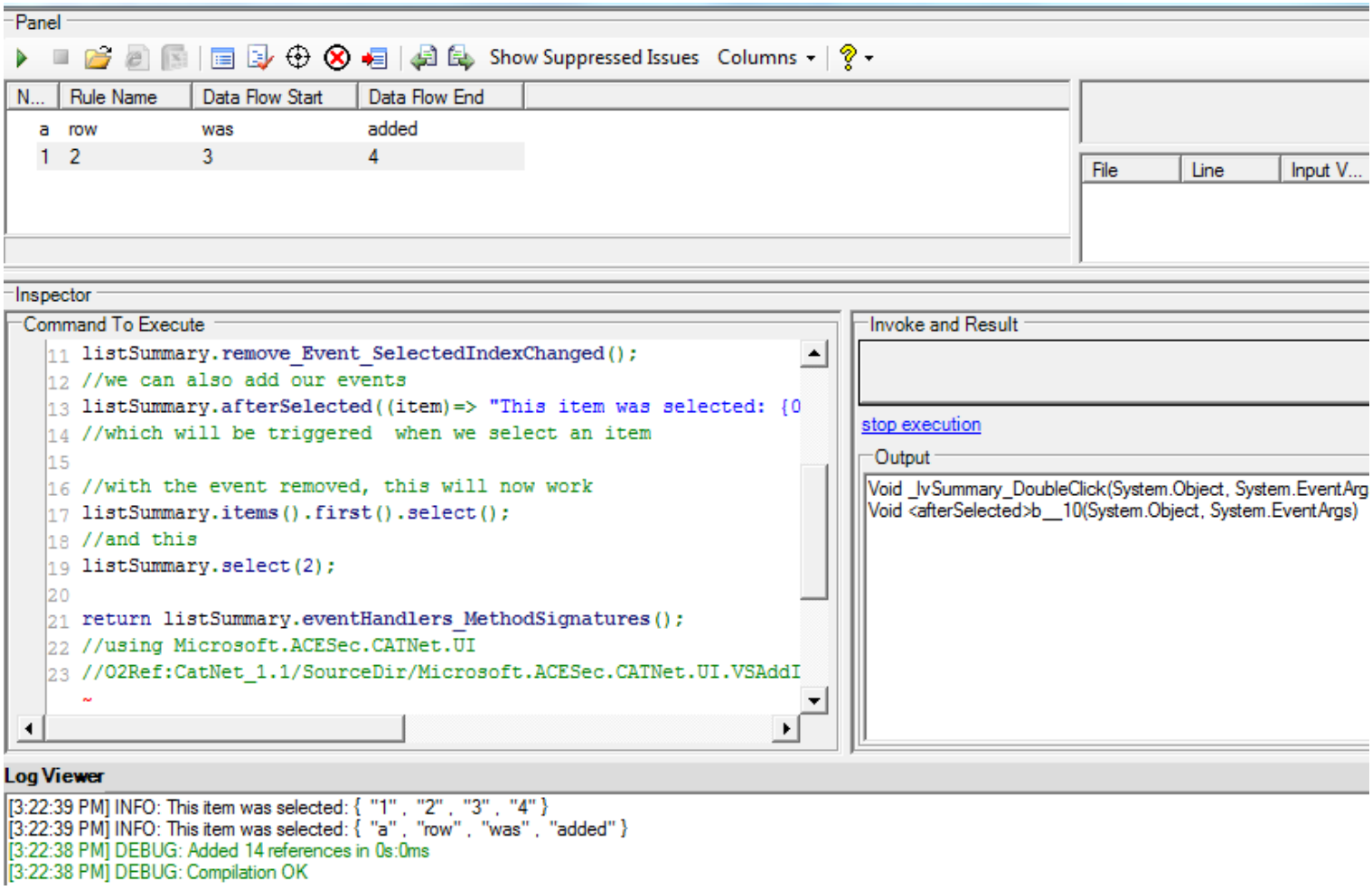
we can create a dedicated method for this:

```
listSummary.remove_Event_SelectedIndexChanged();
return listSummary.eventHandlers();
```

And here is what the original code now looks like:

```
//var topPanel = "Remove event PoC".popupWindow();
var topPanel = panel.clear().add_Panel();
//create an instance of Cat.Net SummaryView control
var summaryView = topPanel.clear().add_Control<SummaryView>();
//get the private field _lvSummary
var listSummary = (ListView)summaryView.field("_lvSummary");
//add a couple items to the list
listSummary.add_Row("a","row","was","added")
                .add_Row("1","2","3","4");

//remove event (using new ExtensionMethods)
listSummary.remove_Event_SelectedIndexChanged();

//we can also add our events
listSummary.afterSelected((item)=> "This item was selected: {0}".info(item.values().toString()));
//which will be triggered  when we select an item

//with the event removed, this will now work
listSummary.items().first().select();
//and this
listSummary.select(2);

return listSummary.eventHandlers_MethodSignatures();
//using Microsoft.ACESec.CATNet.UI
//O2Ref:CatNet_1.1/SourceDir/Microsoft.ACESec.CATNet.UI.VSAddIn.dll
```

which when executed will throw no errors

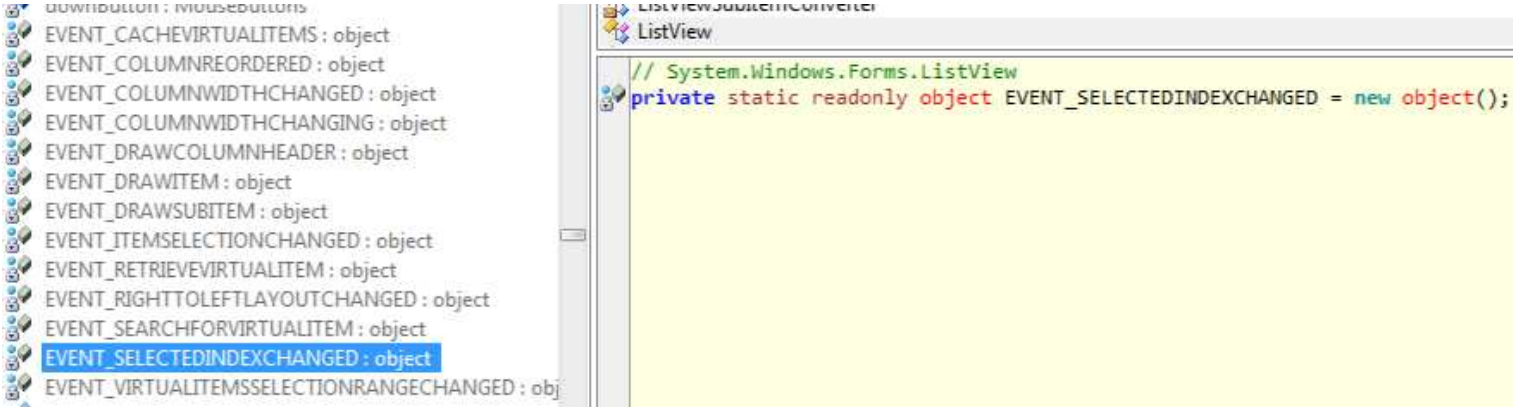


ExtensionMethods Created

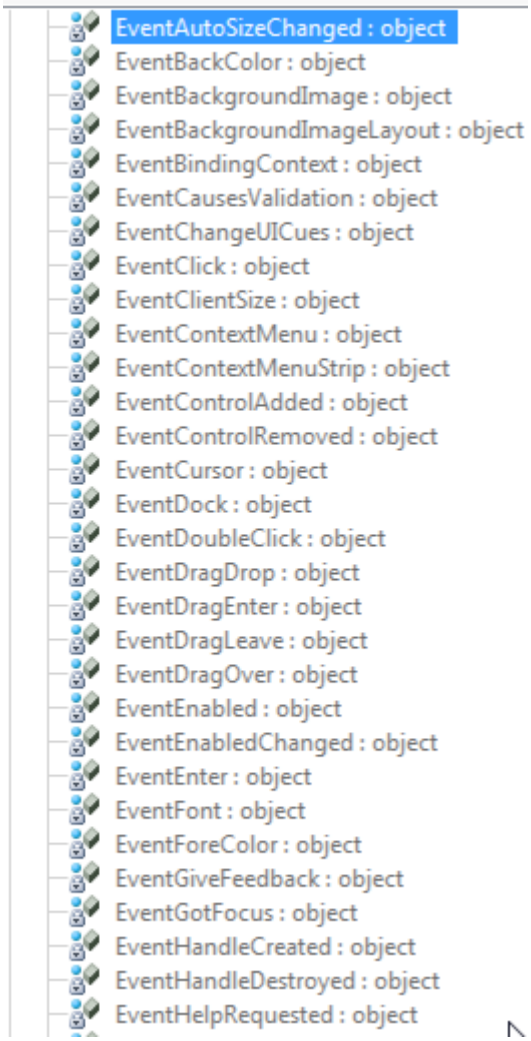
```
public static class _Extra_extensionMethods_Component
{
	public static List<Tuple<object,Delegate>> eventHandlers(this Component component)

	{
		var mappedEvents = new List<Tuple<object,Delegate>>();
		var events = (EventHandlerList)component.prop("Events");
		var next = events.field("head");
		while(next!=null)
		{
			var key = next.field("key");

			var handler = (Delegate)next.field("handler");
			mappedEvents.Add(new Tuple<object,Delegate>(key,handler));
			next = next.field("next");
		}
		return mappedEvents;
	}

	public static string eventHandlers_MethodSignatures(this Component component)

	{
		var signatures = "";
		foreach(var eventHandler in component.eventHandlers())
			if (eventHandler.Item2 != null)
				signatures += eventHandler.Item2.Method.str().line();
		return signatures;
	}

	public static T remove_EventHandler<T>(this T component, string eventId)
		where T : Component
	{
```

```
                var eventIdObject =  typeof(T).ctor().field(eventId);

                //get private 'Events' property
                var events = (EventHandlerList)component.prop("Events");
                //invoke private method 'find' in order to get the  SelectedIndexChanged event entry
                var listEntry = events.invoke("Find", eventIdObject);
                //get the private field 'handler'
                var handler = (EventHandler)listEntry.field("handler");
                //now that we have the EventHandler object we can remove it normaly
                events.invoke("RemoveHandler", eventIdObject, handler);
                return (T)component;
        }

        public static T remove_Event_SelectedIndexChanged<T>(this T component)
                where T : Component
        {
                return component.remove_EventHandler("EVENT_SELECTEDINDEXCHANGED");
        }

}
```

For reference here are the ListView static variables used for events



and here are the ones from Forms.Control (not marked as readonly and with different capitalization)




```
// System.Windows.Forms.Control
static Control()
{
    Control.EventAutoSizeChanged = new object();
    Control.EventKeyDown = new object();
    Control.EventKeyPress = new object();
    Control.EventKeyUp = new object();
    Control.EventMouseDown = new object();
    Control.EventMouseEnter = new object();
    Control.EventMouseLeave = new object();
    Control.EventMouseHover = new object();
    Control.EventMouseMove = new object();
    Control.EventMouseUp = new object();
    Control.EventMouseWheel = new object();
    Control.EventClick = new object();
    Control.EventClientSize = new object();
    Control.EventDoubleClick = new object();
    Control.EventMouseClick = new object();
    Control.EventMouseDoubleClick = new object();
    Control.EventMouseCaptureChanged = new object();
    Control.EventMove = new object();
    Control.EventResize = new object();
    Control.EventLayout = new object();
    Control.EventGotFocus = new object();
    Control.EventLostFocus = new object();
```

.

.

.

.

.

.

.