Optimized ScrollView Adapter

(historically, "ScrollRectItemsAdapter" or SRIA)

Documentation

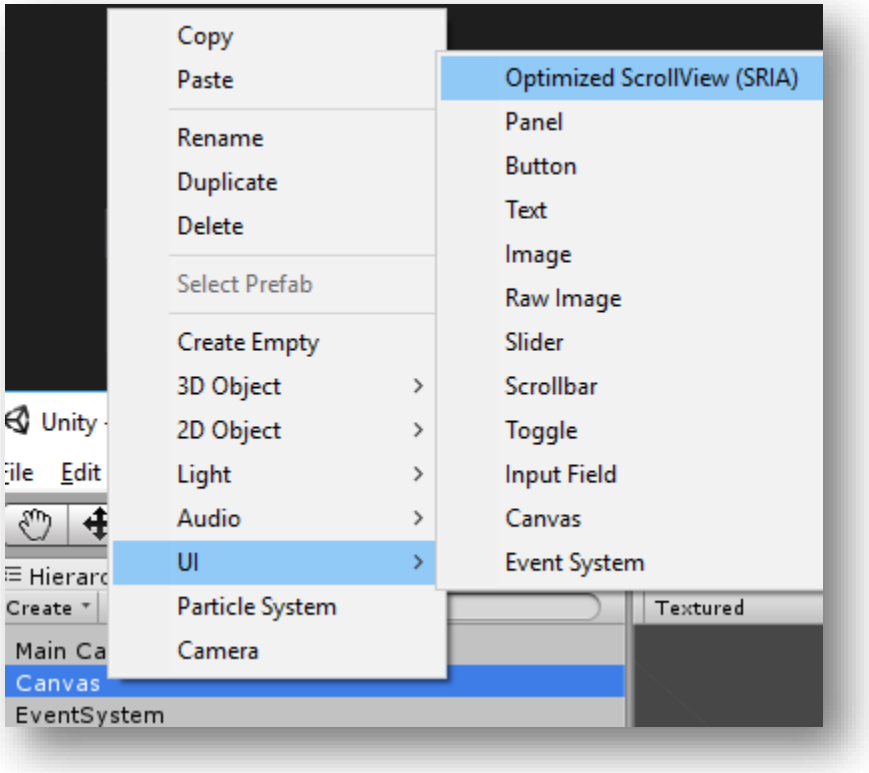
The code reference is an online resource that can be accessed via '*frame8->Optimized ScrollView Adapter-> Code reference*' menu item.

For v3.0 and higher, there's a quick start video guide on youtube, which can be accessed via '*frame8->Optimized ScrollView Adapter-> Quick start video*'

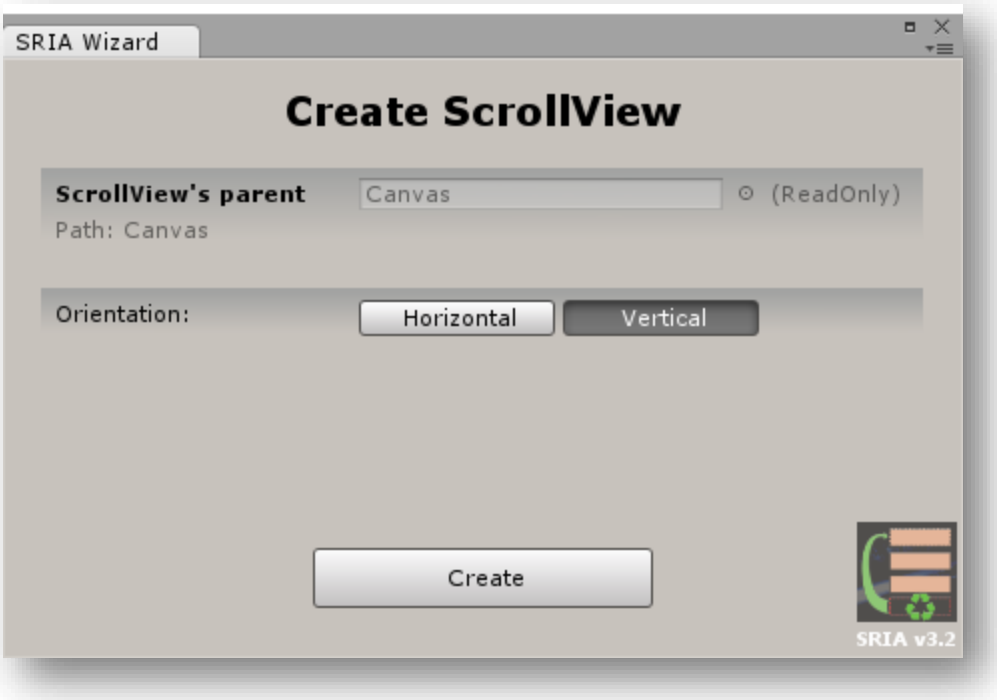
SRIA wizard

Starting with **v3.2**, a graphical interface is provided to help you generate a ScrollView from scratch, a scrollbar and a SRIA implementation based on 2 available templates: List or Grid

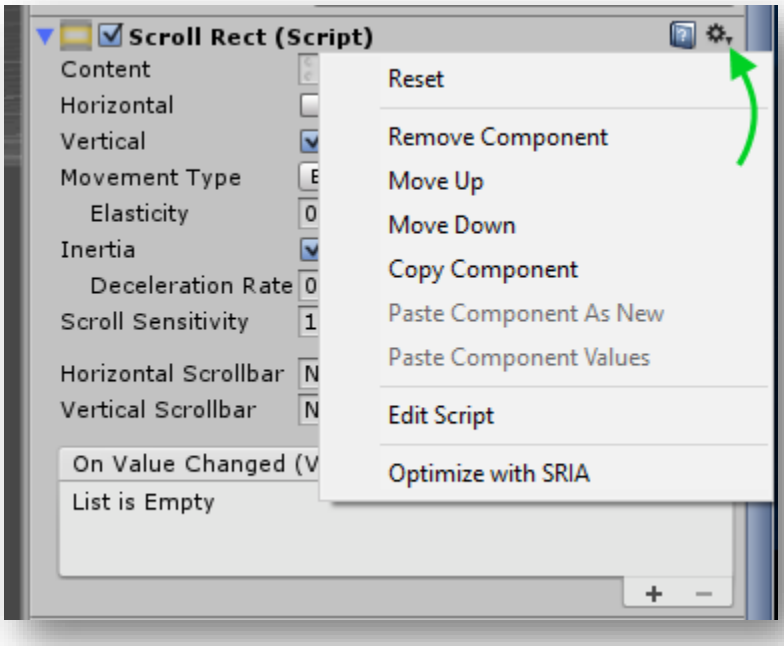
Create from scratch:



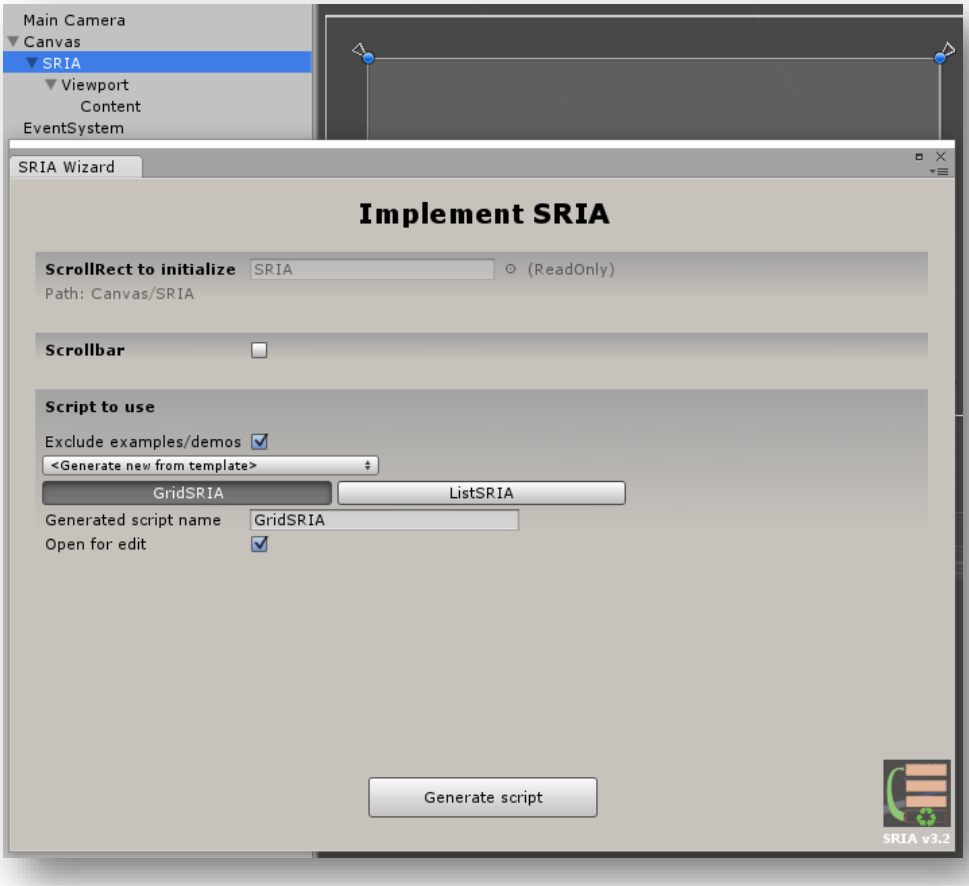
*Choose horizontal or vertical and hit **Create**:*



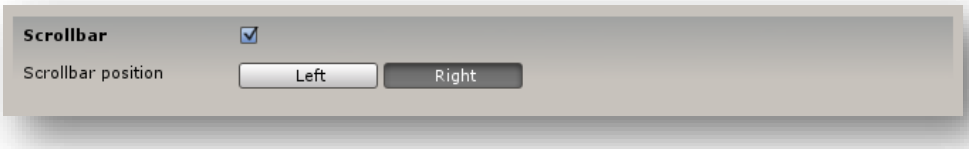
*If you already have a **ScrollRect** in the scene, click here to open the "Implement SRIA" window:*



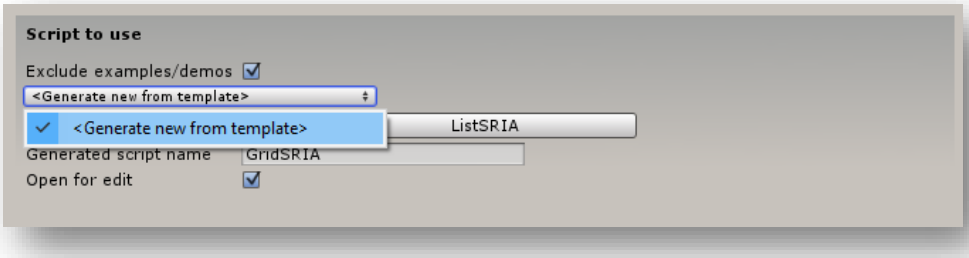
Choose whether to use a scrollbar and which existing implementation to use (or from which template to generate a new one). If a scrollbar already exists, it'll be detected & linked automatically (you can still generate a new one and disable the old one, if you want):



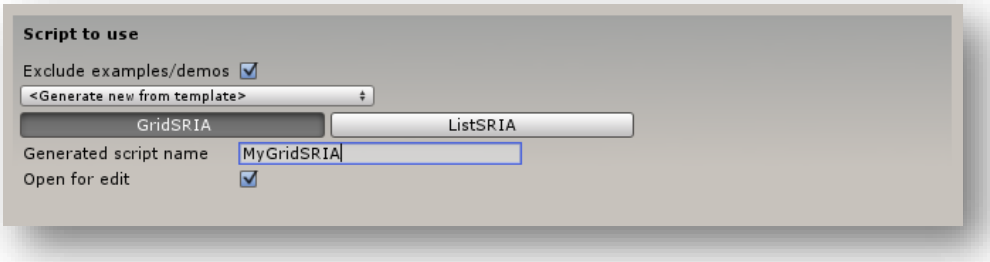
The scrollbar can be Left/Right or Top/Bottom, depending on the scrollview's orientation:



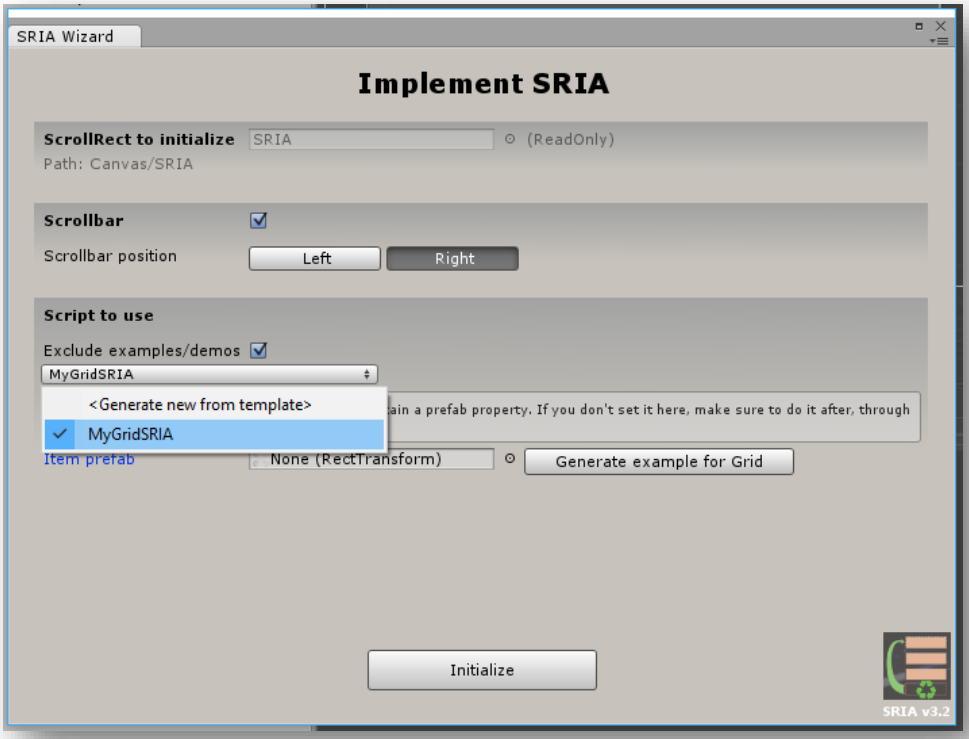
Initially, there's no implementation that can be used in production, only example implementations. Click here to create a new one:



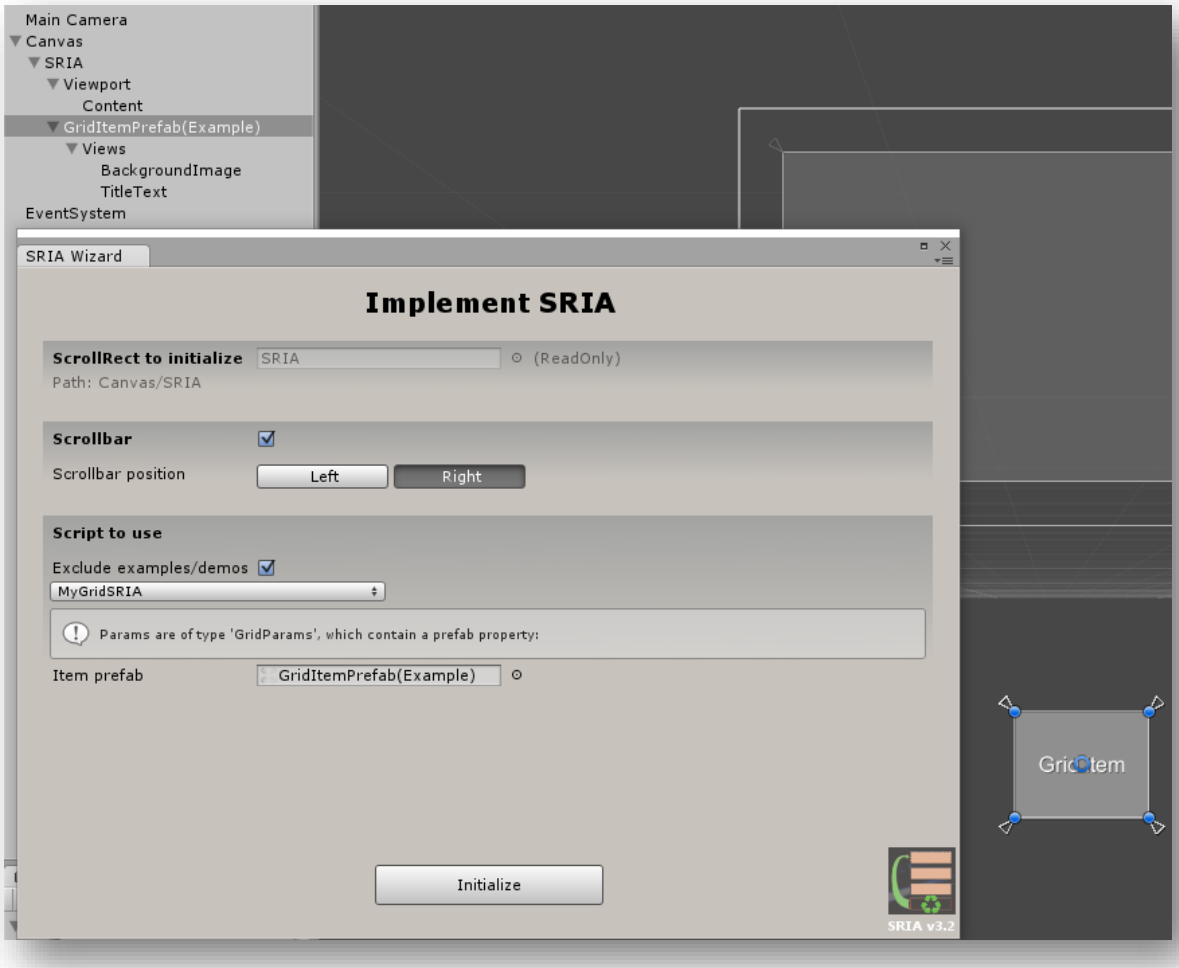
*Type a unique name and hit **Generate**:*



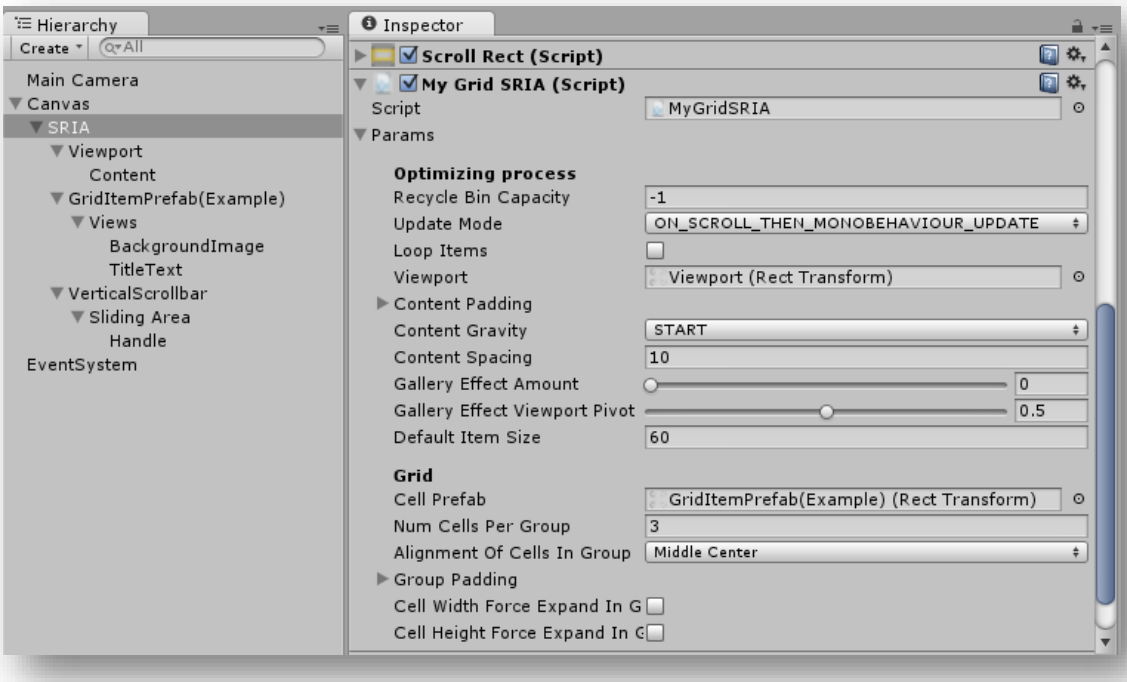
*The newly generated implementation will be auto-selected and, if a prefab property is detected on the used **Params** class, it'll be exposed here as "Item prefab". If you just want a quick start and don't already have a prefab for the items, or if you want to see what an example item prefab should look like, hit "Generate example for **X**")the prefabs for lists and grids differ from each other):*



*Here's what the prefab looks like. It'll be highlighted in the hierarchy. The **BackgroundImage** and **TitleText** children are just for visualization:*



*After you hit Initialize, the ScrollRect is configured, the scrollbar is generated and/or configured, some initial values are set for the **Params** and you can open the script for editing("MyGridSRIA" in this case):*



*You can un-comment sections marked with /**/ or remove them if they won't be used (if you want to test the code, find the following methods and remove or just keep them commented: **OnBeforeRecycleOrDisableCellViewsHolder()**, **GetViews()**, **MarkForRebuild()**):*

```
public class MyGridSRIA : GridAdapter<MyGridParams, MyGridItemViewsHolder>
{
    #region SRIA implementation
    protected override void Start()
    {
        // Calling this initializes internal data and prepares the adapter to handle item count changes
        base.Start();

        // Retrieve the models from your data source and set the items count
        /*
        RetrieveDataAndUpdate(1500);
        */
    }

    // This is called anytime a previously invisible item become visible, or after it's created,
    // or when anything that requires a refresh happens
    // Here you bind the data from the model to the item's views
    // *For the method's full description check the base implementation
    protected override void UpdateCellViewsHolder(MyGridItemViewsHolder newOrRecycled)
    {
        // In this callback, "newOrRecycled.ItemIndex" is guaranteed to always reflect the
        // index of item that should be represented by this views holder. You'll use this index
        // to retrieve the model from your data set
        /*
        MyGridItemModel model = _Params.Data[newOrRecycled.ItemIndex];

        newOrRecycled.backgroundImage.color = model.color;
        newOrRecycled.titleText.text = model.title + " #" + newOrRecycled.ItemIndex;
        */
    }
```

If you followed all the steps until now correctly, hit Play and you should see it in action:

Usage

SRIA, **BaseParams** and **BaseItemViewsHolder** are the 3 core classes in our small library dedicated to both optimize a Scroll View and programmatically manage its contents.

You can use it both for a horizontal or vertical ScrollView.

SRIA it's an abstract, generic MonoBehaviour, which you need to extend and provide at least the implementation of **SRIA.CreateItemViewHolder()** and **SRIA.UpdateItemViewHolder()**.

It's recommended to manually go through example code provided in **ScrollRectItemsAdapterExample.cs** and **SimpleTutorial.cs** in order to fully understand the mechanism. You'll find detailed comments in core areas. You may even use this script directly without implementing your own, in some simple scenarios.

(Some may find it more easy to consult the example code+scene directly or the quick start video tutorial, without reading this document)

Implementation

*(Follow these steps while constantly looking at how it's done in the example code in **SimpleTutorial.cs*** and optionally in ***ScrollRectItemsAdapterExample.cs**)*

Here's the normal flow you'll follow after you've created a Scroll View using **GameObject->UI->Scroll View:**

1. create your own implementation of **BaseItemViewsHolder**, let's name it **MyItemViewsHolder**

2. create your own implementation of **BaseParams** (if needed), let's name it **MyParams**

3. create your own implementation of **SRIA<MyParams, MyItemViewsHolder>**, let's name it **MyScrollRectItemsAdapter**

4. override **Start()**, call **base.Start()**, after which:

5. call **MyScrollRectItemsAdapter.ResetItems(int count)** once (and any time your dataset is changed) and the following will happen:
   - **CollectItemsSizes()** will be called (which you can optionally implement to provide your own sizes, if known beforehand
   - **CreateViewsHolder(int)** will be called for each view that needs creation. Once a view is created, it'll be re-used when it goes off-viewport
     - **newOrRecycledViewsHolder.root** will be null, so you need to instantiate your prefab), assign it and call **newOrRecycledViewsHolder.CollectViews().** Alternatively, you can call its **AbstractViewHolder.Init(..)** method, which can do a lot of things for you, mainly instantiate the prefab and (if you want) call **CollectViews()**
     - after creation, only **MyScrollRectItemsAdapter.UpdateItemViewHolder()** will be called for it when its represented item changes and becomes visible

  - this method is also called when the viewport's size grows, thus needing more items to be visible at once
- 5.3. `MyScrollRectItemsAdapter.UpdateViewsHolder(MyItemViewsHolder)` will be called when an item is to be displayed or simply needs updating:
  - use `newOrRecycledViewsHolder.ItemIndex` to get the item index, so you can retrieve its associated model from your data set (most common practice is to store the data list in your `Params` implementation)
  - `newOrRecycledViewsHolder.root` is not null here (given the view holder was properly created in `CreateViewsHolder(..)`). It's assigned a valid object whose UI elements only need their values changed (common practice is to implement helper methods in the view holder that take the model and update the views themselves)

`ResetItems()` is also called when the viewport's size changes (like for orientation changes on mobile or window resizing on standalone platforms)

Example scenes & utilities

All the example scenes & the utility scripts are provided on an "as-is" base. Their main purpose is to demonstrate the feature-set and show you the basic code-flow when implementing the adapter, following the recommended best-practices & conventions.

Known issues & workarounds

- **If building for Universal Windows Platform**, you must replace `StandardInputModule` with `SRIAStandardInputModule` or, if you're already using a custom input module, subclass it from this instead of StandardInputModule. If this is not possible, `ISRIAPointerInputModule` should be implemented by it (more details in ISRIAPointerInputModule.cs). Similarly, for older Unity versions where `TouchInputModule` should also be present, `SRIATouchInputModule` was provided. These modules contain a single method which gives access to some internal data needed by SRIA

- Not actually related to the plugin itself, but worth mentioning: some lower-end devices have terrible performance with Open GL 3 and/or Auto Graphics API. If you experience oddly low FPS, untick Auto Graphics API and use Open GL 2 instead.

- In the ContentSizeFitter example scene: the prefab's Text will be oddly truncated if its "Vertical Overflow" property is set to "Truncate". So a as general rule, set it to "Overflow" when you have similar scenarios. Likewise, if you have a horizontal ScrollView, the "Horizontal Overflow" property is the one to be modified.