Optimized ScrollView Adapter - Manual

! This is the manual for 5.0. If using OSA 6.x+, see OSA 6.0 Manual instead !

Contents

1. Documentation
2. External links
3. OSA wizard
4. Usage
5. Implementation
6. Grid
7. Table
8. Example scenes & utilities (separate manual)
9. Playmaker support
10. Known issues, workarounds
11. Tips
12. FAQ
13. Help improve OSA by leaving a review 😊

1. Documentation

🚩 *Non-english speakers: Install Translate+ for Google Docs. It works surprisingly well!*

In v4.1, the abbreviation **SRIA** (historically, ScrollRectItemsAdapter) was changed to **OSA**: they can be used interchangeably, but in this document, we stick to **OSA**

The code reference is an online resource that can be accessed via '*Tools->OSA->Code reference*' menu item. In previous versions, the menu item was called *frame8*. It changed to *Tools* in v4.3.

For v3.0 and higher, there's a quick start video guide on YouTube, which can be accessed via '*Tools->OSA->Quick start video*'. This video is still relevant even for v4.0-v4.3, although there are a few differences which are pointed out in the migration guide

Starting with v4.3, **OSA** uses assembly definitions, so you can add a reference to the */Scripts/OSA.asmdef* if you also use them.

2. External links

Applications using **OSA**

Migrate from 3.2 to 4.0-4.1, from 4.1 to 4.2, from 4.2 to 4.3, from 4.3 to 5.0, from 5.0 to 5.1, or from 5.x to 6.0. Note that migrations need to be done incrementally. For ex., you can't directly jump from 4.1 to 4.3.

In-browser demo || Android demo APK

Quick start video || Playmaker support video for v4.2+

Thread on Unity forum

Changelog

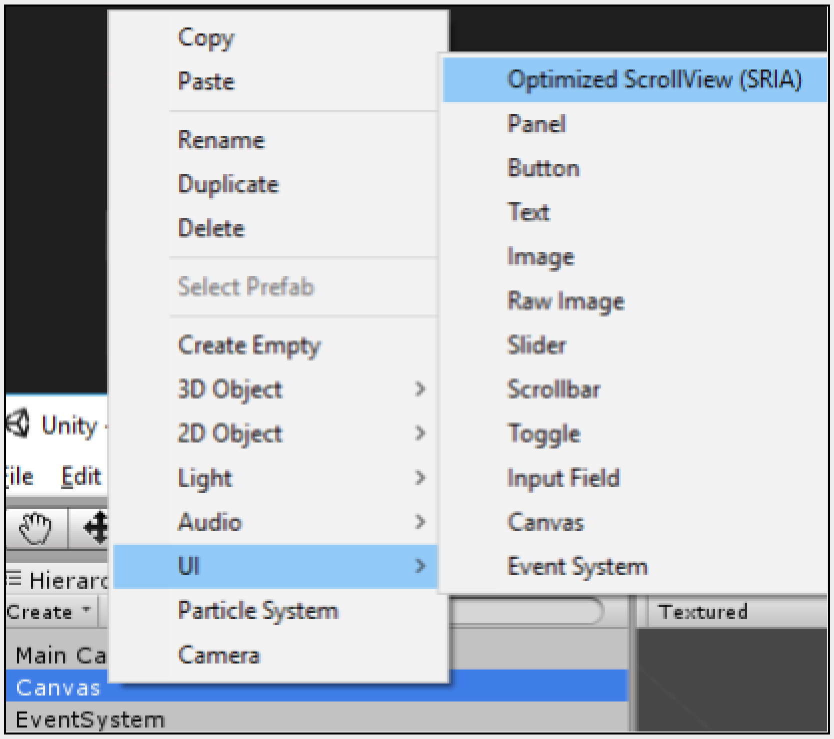
Manual for older versions

3. OSA wizard

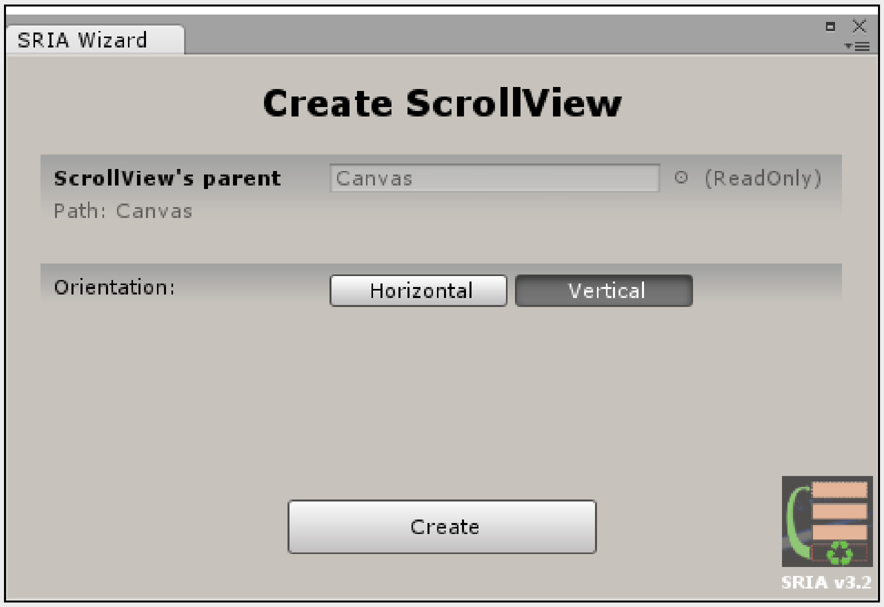
Starting with **v3.2**, a graphical interface is provided to help you generate a ScrollView from scratch, a scrollbar and an **OSA** implementation based on 2 available templates: List or Grid.

See the workflow below

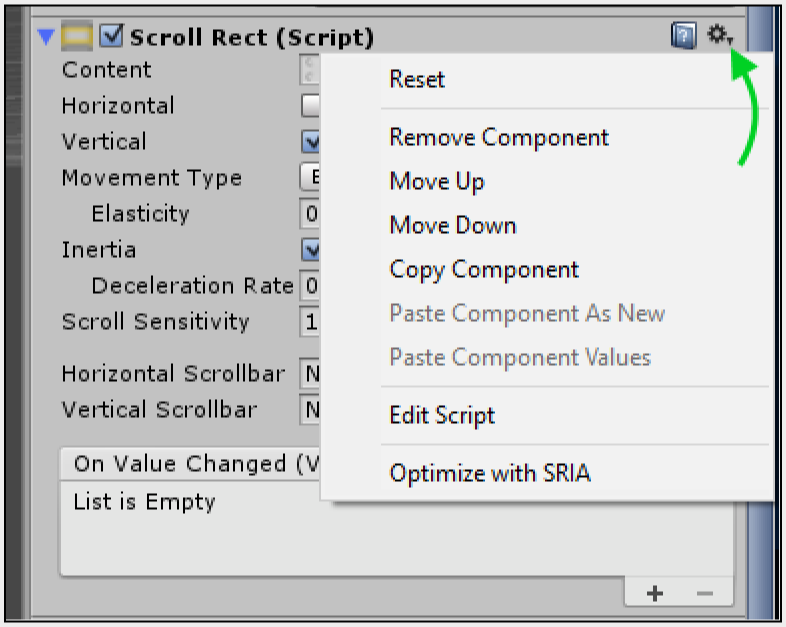
Create from scratch:



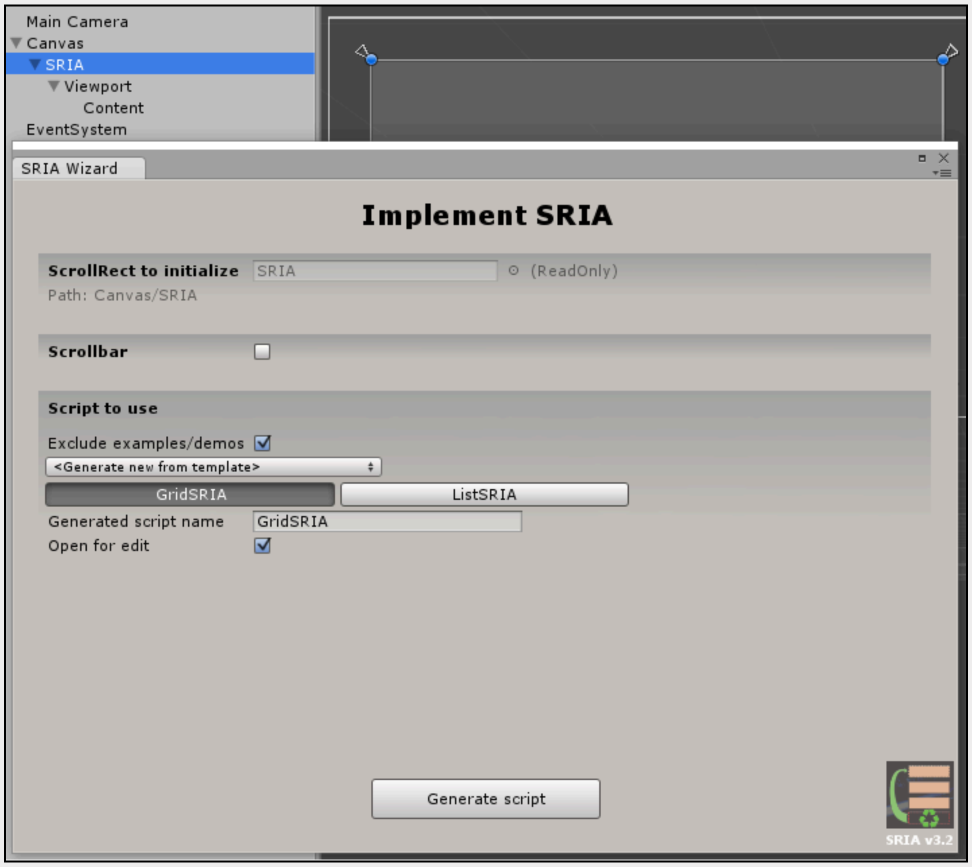
*Choose horizontal or vertical and hit **Create**:*



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



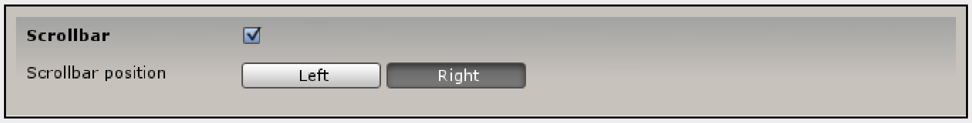
Implement SRIA

ScrollRect to initialize: SRIA (ReadOnly)
Path: Canvas/SRIA

Scrollbar: ☐

Script to use
Exclude examples/demos: ✓
Generated script name: GridSRIA
Open for edit: ✓

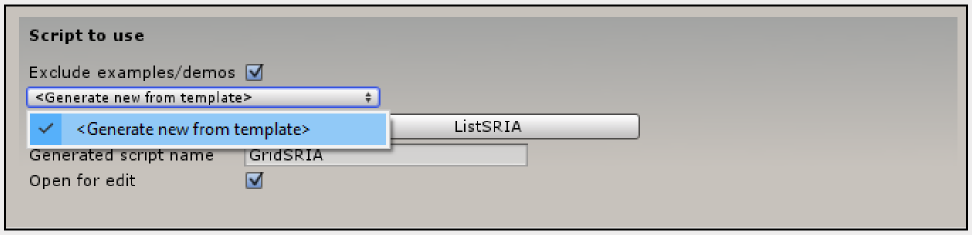
The scrollbar can be Left/Right or Top/Bottom, depending on the ScrollView's orientation:



Scrollbar ✓

Scrollbar position: Left [✓ Right]

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



Script to use

Exclude examples/demos ✓

<Generate new from template>

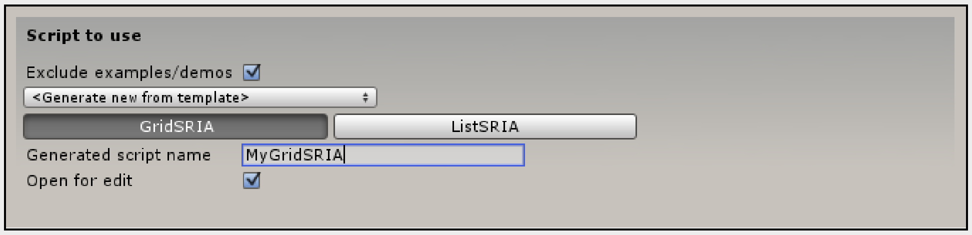
✓ <Generate new from template>

ListSRIA

Generated script name: GridSRIA

Open for edit ✓

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



Script to use

Exclude examples/demos ✓

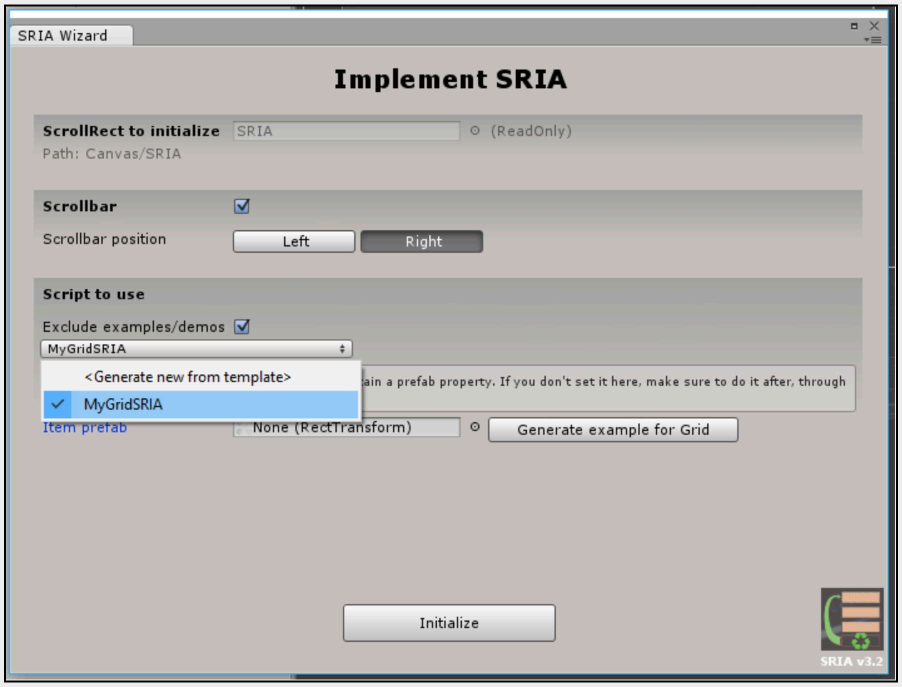
<Generate new from template>

[✓ GridSRIA] ListSRIA

Generated script name: MyGridSRIA

Open for edit ✓

*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):*



SRIA Wizard

Implement SRIA

ScrollRect to initialize: SRIA (ReadOnly)

Path: Canvas/SRIA

Scrollbar ✓

Scrollbar position: Left [✓ Right]

Script to use

Exclude examples/demos ✓

MyGridSRIA

<Generate new from template>

✓ MyGridSRIA

ain a prefab property. If you don't set it here, make sure to do it after, through

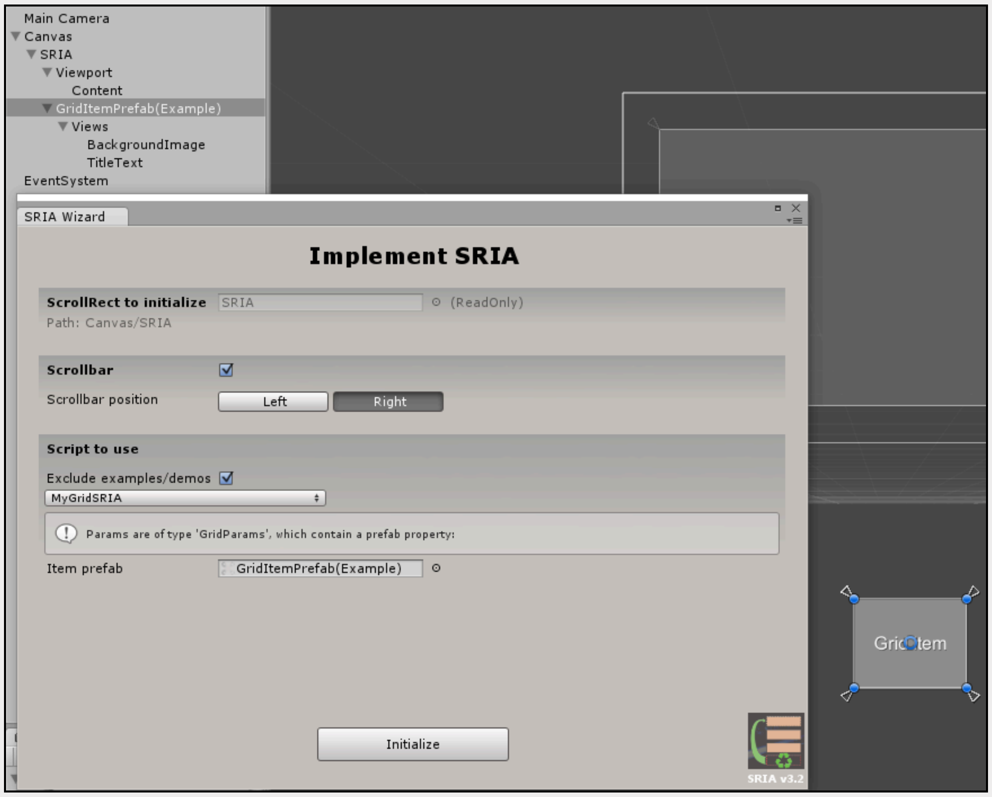
Item prefab: None (RectTransform)

Generate example for Grid

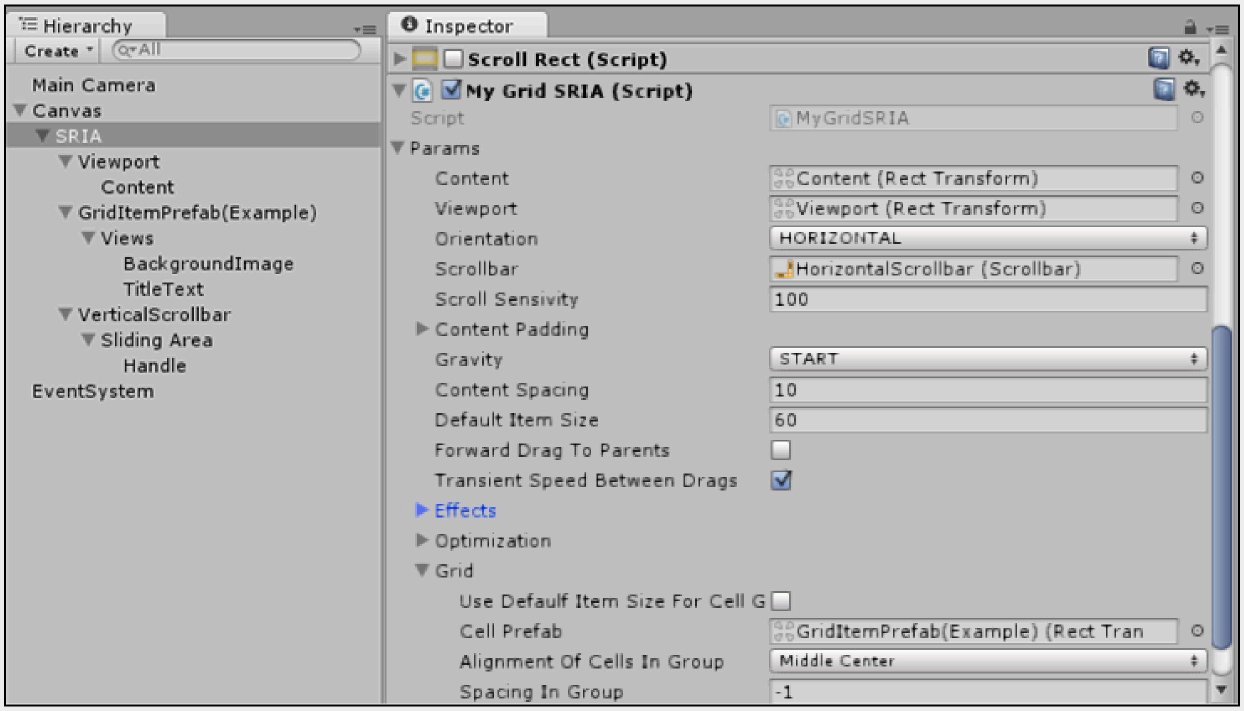
Initialize

SRIA v3.2

*It'll be highlighted in the hierarchy. The **BackgroundImage** and **TitleText** children are just for visualization:*

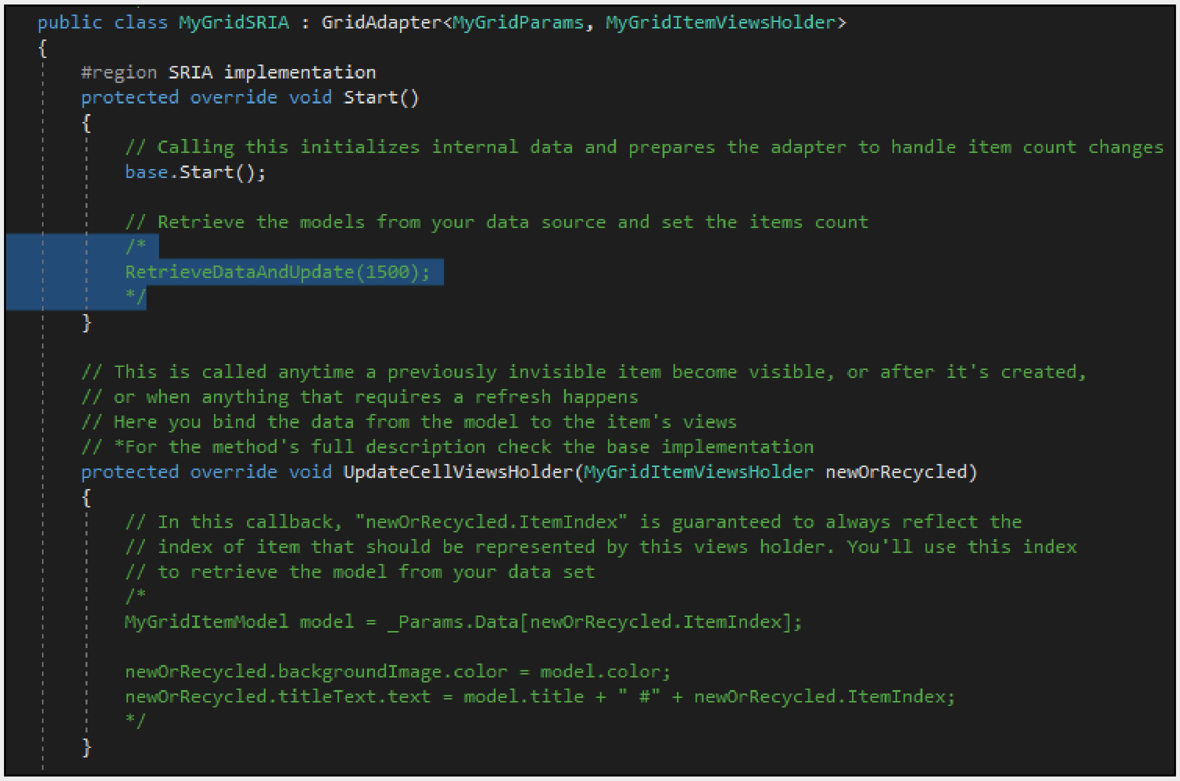


*After you hit Initialize, the ScrollView is configured, the ScrollRect is disabled (you can safely remove it), the scrollbar is generated and/or configured, some initial values are set for the **Params** and you can open the script for editing ("MyGridSRIA" in this case):*



Script	MyGridSRIA
Params	
Content	Content (Rect Transform)
Viewport	Viewport (Rect Transform)
Orientation	HORIZONTAL
Scrollbar	HorizontalScrollbar (Scrollbar)
Scroll Sensivity	100
Content Padding	
Gravity	START
Content Spacing	10
Default Item Size	60
Forward Drag To Parents	☐
Transient Speed Between Drags	✓
Effects	
Optimization	
Grid	
Use Default Item Size For Cell G	☐
Cell Prefab	GridItemPrefab(Example) (Rect Tran
Alignment Of Cells In Group	Middle Center
Spacing In Group	-1

*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()****):*



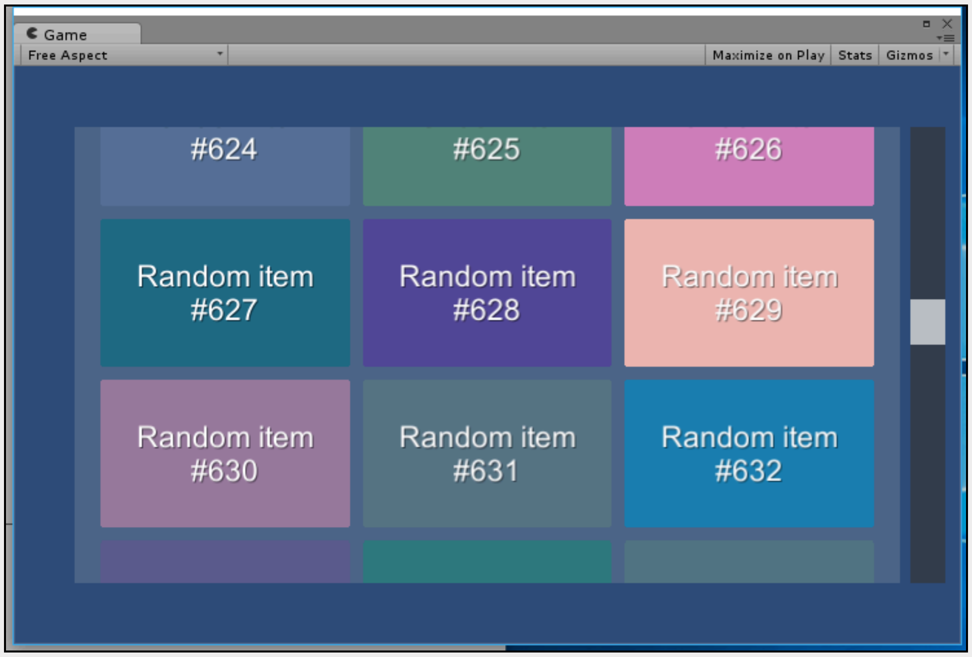
```
public class MyGridSRIA : GridAdapter<MyGridParams, MyGridItemViewsHolder>
{
    #region SRIA implementation
    protected override void Start()
    {
        // Calling this initializes internal data and prepares the adapter to handle item count changes
        base.Start();

        // 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:



4. Usage

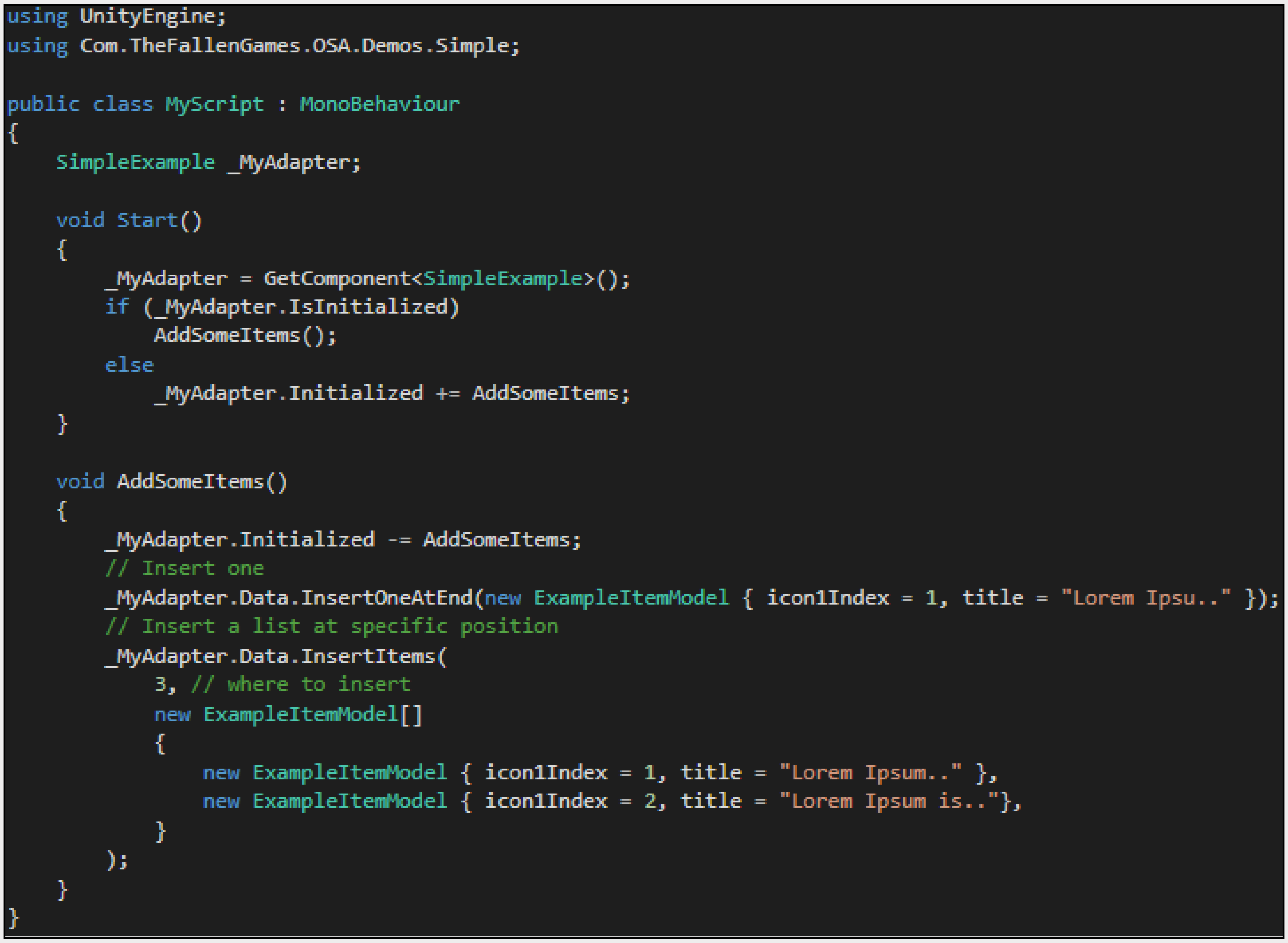
OSA, **BaseParams** and **BaseItemViewsHolder** are the 3 core classes in our small library dedicated to both optimize a Scroll View and programmatically manage its contents.

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

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

It's recommended to manually go through the example code provided in **MainExample.cs** and **SimpleExample.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 very simple scenarios.

Show me the code! After you've implemented the custom **OSA** class managing the views according to your specific case, adding/removing items from outside is more or less the same for any type of **OSA**. Lists have maximum flexibility, while Grids can only use the **ResetItems** method for any data manipulation. The below code uses the **SimpleExample** from the Demos pack to show adding of items in a List **OSA**:



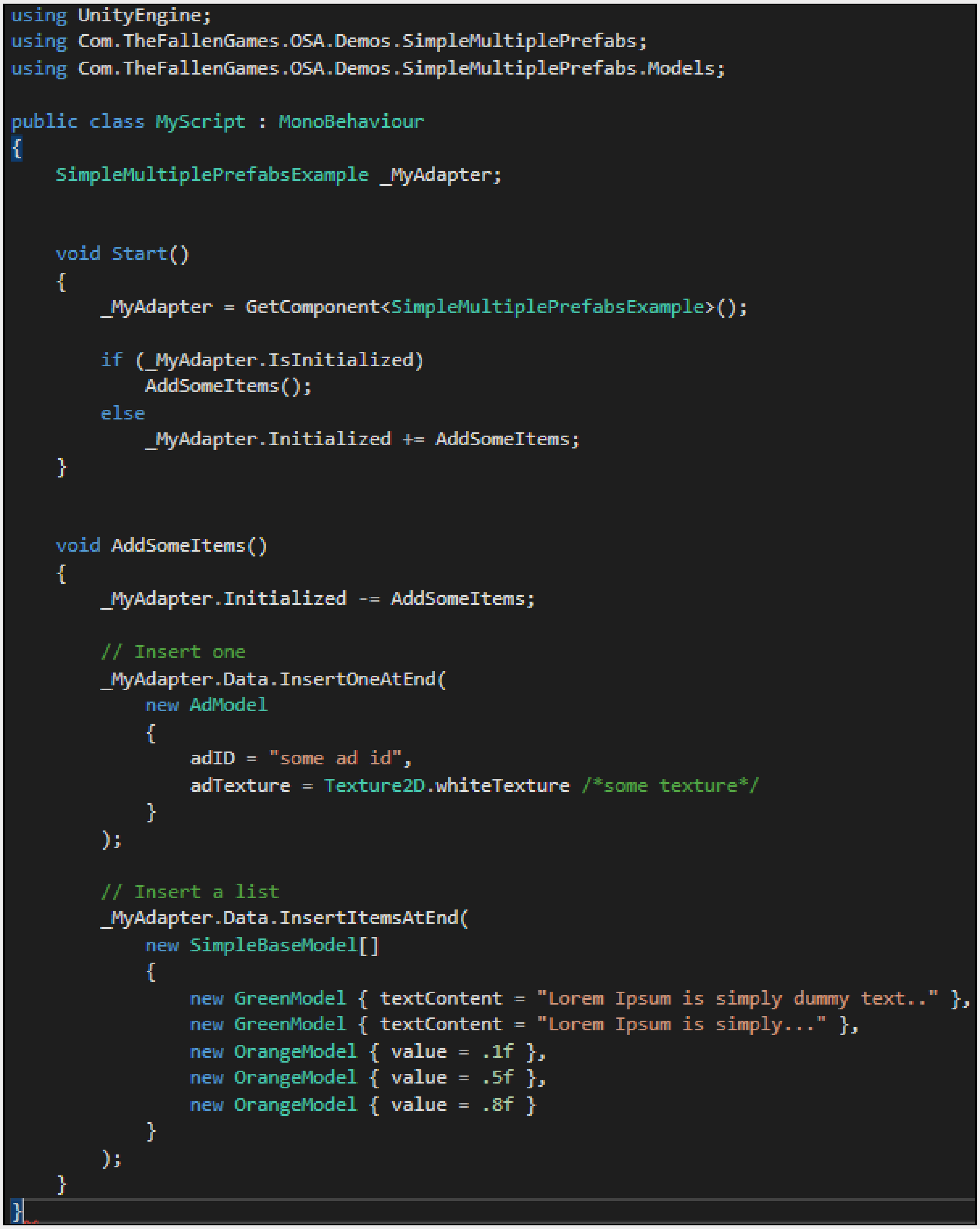
```
using UnityEngine;
using Com.TheFallenGames.OSA.Demos.Simple;

public class MyScript : MonoBehaviour
{
    SimpleExample _MyAdapter;

    void Start()
    {
        _MyAdapter = GetComponent<SimpleExample>();
        if (_MyAdapter.IsInitialized)
            AddSomeItems();
        else
            _MyAdapter.Initialized += AddSomeItems;
    }

    void AddSomeItems()
    {
        _MyAdapter.Initialized -= AddSomeItems;
        // Insert one
        _MyAdapter.Data.InsertOneAtEnd(new ExampleItemModel { icon1Index = 1, title = "Lorem Ipsu.." });
        // Insert a list at specific position
        _MyAdapter.Data.InsertItems(
            3, // where to insert
            new ExampleItemModel[]
            {
                new ExampleItemModel { icon1Index = 1, title = "Lorem Ipsum.." },
                new ExampleItemModel { icon1Index = 2, title = "Lorem Ipsum is.."},
            }
        );
    }
}
```

When you use an **OSA** with multiple prefab types, adding/removing items from outside is similar:



```
using UnityEngine;
using Com.TheFallenGames.OSA.Demos.SimpleMultiplePrefabs;
using Com.TheFallenGames.OSA.Demos.SimpleMultiplePrefabs.Models;

public class MyScript : MonoBehaviour
{
    SimpleMultiplePrefabsExample _MyAdapter;


    void Start()
    {
        _MyAdapter = GetComponent<SimpleMultiplePrefabsExample>();

        if (_MyAdapter.IsInitialized)
            AddSomeItems();
        else
            _MyAdapter.Initialized += AddSomeItems;
    }


    void AddSomeItems()
    {
        _MyAdapter.Initialized -= AddSomeItems;

        // Insert one
        _MyAdapter.Data.InsertOneAtEnd(
            new AdModel
            {
                adID = "some ad id",
                adTexture = Texture2D.whiteTexture /*some texture*/
            }
        );

        // Insert a list
        _MyAdapter.Data.InsertItemsAtEnd(
            new SimpleBaseModel[]
            {
                new GreenModel { textContent = "Lorem Ipsum is simply dummy text.." },
                new GreenModel { textContent = "Lorem Ipsum is simply..." },
                new OrangeModel { value = .1f },
                new OrangeModel { value = .5f },
                new OrangeModel { value = .8f }
            }
        );
    }
}
```

5. Implementation

Follow these steps while constantly looking at how it's done in the example code in ***`SimpleExample.cs`*** and optionally in ***`MainExample.cs`***

Here's the normal flow you'll follow if you've created a Scroll View using **`GameObject->UI->Scroll View`** or through **OSA** wizard:

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 **`OSA<MyParams, MyItemViewsHolder>`**, let's name it **`MyScrollViewAdapter`**
4. Override **`Start(),`** call **`base.Start`**(), after which:
5. Call **`MyScrollViewAdapter.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). This is not the only, nor the recommended way to provide the sizes, especially on large data sets. More on that later.

   - **`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

     – **`newOrRecycled.root`** will be null, so you need to instantiate your prefab), assign it and call **`newOrRecycledViewsHolder.CollectViews().`** Alternatively, you can call its **`AbstractViewsHolder.Init(..)`** method, which can do a lot of things for you, mainly instantiate the prefab and (if you want) call **`CollectViews()`**.

     – after creation, only **`MyScrollViewAdapter.UpdateViewsHolder()`** 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.

   - **`MyScrollViewAdapter.UpdateViewsHolder(MyItemViewsHolder)`** will be called when an item is to be displayed or simply needs updating:

     – use **`newOrRecycled.ItemIndex`** to get the item index, so you can retrieve its associated model from your data set

     – **`newOrRecycled.root`** is not null here (given the ViewsHolder 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 viewsholder 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)

6. Grid

The documentation about implementing grids was initially created around the “Grid, horizontal layout, async items download” demo from the Demos manual, because following an actual example makes it easier to both understand and see how it works.

So not only is that manual describing that particular demo, but it also contains most of the information you’ll ever need about the grids, including common questions.

7. Table

The TableView manual covers the basics of implementing a TableView.

8. Example scenes & utilities

The demos have their own document which can be accessed here.

There you’ll also find a list of all demos and their manuals.

9. Playmaker support

🚩 *Important note for **OSA** 5.0 and below: The README.txt in the **Playmaker** support folder didn't mention an error that you get due to ASMDEFs in Unity 2017.3 and up. The updated README is now found here, where you can see the latest changes even before new **OSA** versions are released.*

In version 4.2, **Playmaker** support was added. All the assets needed to implement **OSA** through **Playmaker** are in a package under */Extra/PluginSupport/Playmaker* for versions older than 4.3, and */PluginSupport/Playmaker for 4.3 and up*.

Check out the README file for how to import it.

Playmaker support only works for **Unity 2017.1.0f3** and up + **Playmaker 1.9.0** and up!

What you can do with **Playmaker:**

- Insert/Remove/Reset/Clear items, Smooth scroll to item index
- `ListViews` with arbitrary data and views structure (defined by you)
- `GridViews`
- Items of variable sizes, using `ContentSizeFitter`. Only available for `ListViews`
- Lazily-initializing data models (if using large amounts of items)
- Displaying large amounts of arbitrary data from XML using DataMaker, although DataMaker is a bit slow for more than 10K items in our examples, so you need something faster than DataMaker if you want to display more items (we support around 2 billion)
- Open our built-in `DateTimePicker` via a simple action and retrieve the picked date and time either in a string variable or each datetime component (year, month, minute etc.) in an integer variable in your `FSM`
- Use the `FSM`s in the existing Controllers and Item prefabs as a guide to create your own, customized ones. It's easy!

Check the Playmaker YouTube tutorial at the beginning of this document for how to get started and useful tips!

10. Known issues, workarounds

- In several Unity versions (2018.4.9, 2018.4.34, 2019.4.10 and probably more), **`RectTransform`** doesn't correctly report its size inside **`Awake()`**. Some only behave like this if the **`RectTransform's`** anchors are not brought together. This causes an issue when trying to initialize OSA early in **`Awake`** as opposed to doing it in **`Start`**. The issue is that the ScrollView itself reports a wrong size and is very subtle as it doesn't impact most of the users. This is because OSA automatically detects if the ScrollView's size changes and calls a rebuild, but in case of grids (and probably other undiscovered edge-cases) **`ScrollTo`** may not behave as expected during the same frame. Solution? Either initialize OSA in **`Start`** (general solution), or use **`SmoothScrollTo`** with a 0 duration (particular solution only for this case).

- (Not related to **OSA**) Some Unity 2019.1 and 2019.2 have problems with setting an Image's sprite property via script. See Forum thread

- (Android, but maybe other platforms have this too): On some (generally older) Unity versions, on some scenes, even if they're simple, Gfx.WaitForPresent() as seen in Profiler eats up between 10 and 20 FPS, and we couldn't find out why. Our fix was to build using the latest Unity at that moment, which was 2019.1. Email to lucian@thefallengames.com if you have any input on this. This is a popular issue with Unity which seems to happen randomly.

- 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 on some Unity versions if its **`Vertical Overflow`** property is set to **`Truncate`**. So, as a 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.

- If you're planning to enable/disable game objects inside the CSF, then keep then disabled initially (in the prefab) if they'll immediately be disabled anyway in the first frame after Init (and vice-versa: any GOs enabled by default in the adapter's code, also keep them initially enabled in the prefab), because otherwise the adapter can't initialize correctly. Concrete use case: Under the item prefab, you have a text child that has its gameObject disabled initially and it needs to activate itself on click and show arbitrarily sized data. See this forum post for more info

- Some Unity versions have a bug with their **`RectMask2D`** implementation, so if you see some demos not clipping the items inside the **`Viewport`**, you should replace it with a **`Mask+Image`** (the old way). We've found this to be the case for WebGL builds of Unity 2018.1 and 2018.2 when the **`Canvas`**' space is not **`Overlay`**, but it's a sure thing that this may also happen on other Unity versions and/or build targets

- Some users will get a call stack size exceeded "RangeError" exception when building with High stripping for WebGL. You should make sure to include the **OSA**'s namespaces manually in your link.xml. See this post (credits to @doctorpangloss)

11. Tips

- Use a **`Canvas`** as a parent of groups of frequently changing UI elements to separate them from the less frequently changing ones. **`Canvas`**es can be nested. More info: Link

- Use a **`RectMask2D`** instead of **`Mask`**. It speeds Scroll Views in Unity when you don't need masking by a specific shape. Try it, but make sure to test it on different platforms, as some Unity versions have a buggy implementation of it, as stated in the *Known issues* section above.

12. FAQ

1. *How to use the examples without the DrawerCommandPanel?*

Update 24.07.2018 for v4.1: **`DrawerCommandPanel`** is now decoupled from the examples, but we still don't recommend using the example scripts in production

See *Example scenes & Utilities* above

2. *Why not use MonoBehaviour as a base class for the Items (AbstractItemViewsHolder)?*

Performance.

The prefab used to instantiate the items can have any number of components on it.

You can re-use your **`MonoBehaviour`**, if there's where you're already retrieving the views (or perhaps setting them via inspector). Look:

```
class MyVH : BaseItemViewsHolder
{
    public YourExistingMonoBehaviourType behaviour;

    public override void CollectViews()
    {
        base.CollectViews();
        behaviour = root.GetComponent<YourExistingMonoBehaviourType>();
    }
}
```

and when you'll bind the data to the views (in **`UpdateViewsHolder`**() in our case), you can simply use your existing behavior the same way as before, just that now it's a field in the views holder:

`viewsHolder.behavior.UpdateViews (<dataModel>)`.

3. *How to do action on click/long-click?*

You can borrow the code from any other scene that includes a specific action when an item is pressed.

The trick is to have a single method that handles the **`onClick`** of all items, but which is also passed the associated **`viewsHolder`**, so you can retrieve the model knowing **`vh.ItemIndex`**.

Example: When creating the views holder:

```
MyViewsHolder vh =...;
vh.button.onClick.AddListener(() => OnItemClicked(vh));
```

Long-clicking is handled the same way, and a utility script is provided for that, in case you don't have your own. See the *SelectionExample* script and its associated scene for this.
Also, in case of grids, cells are created for you, so in order to execute some code, like adding a listener, right after their creation, override **`OnCellViewsHolderCreated`**.

4. *How to make looping work when there are fewer items than what can be shown in viewport?*

Solution 1: Duplicate your models so that the number of items which the adapter sees is always bigger than the minimum needed (this number depends on several factors, like the viewport's size and how your items' size changes based on their anchors, but in practice you'll find that number to be roughly **`(minItemSize+spacing)/viewportSize + 1`**, where **`minItemSize`** is the size of the smallest item that you'll have, in case they can have different sizes).

Example of a duplicated list of 2 items, where the minimum displayed can be 4: [model1, model2, model1, model2, model1, model2]

Solution 2: Enable\disable looping on count change, based on the content-to-viewport ratio. Let's say if content is less than 1.3x the viewport's size, you disable looping.
You can retrieve this ratio using the extensions in IScrollRectProxy.cs.
You should choose a threshold ratio in a way that will always keep at least 1 item out of view.

5. ***Is it possible to add/remove items from UpdateViewsHolder?***

Nope. **`UpdateViewsHolder`** it's not supposed to have the responsibility of deciding whether items should be added/removed. It just binds data to the views. That's all. I'm pretty sure you can achieve whatever you're looking for without the need to add/remove items from **`UpdateViewsHolder`**.

6. ***What's the purpose of separate calls to _Params.Data.Add and InsertItems? Is there any useful case when they can be called without each other?***

Update 24.07.2018 for v4.1: DataHelpers are now provided which do this for you, ***if*** you don't need full flexibility

In short, the biggest advantage is to be able to display items whose model wasn't yet retrieved completely (maybe you're downloading them one by one or just downloading them when needed, i.e. when first shown).

This is needed because **OSA** was from the very beginning designed to be as flexible as possible. The adapter itself doesn't know about your list or how you store your data. It just asks for the items count and, optionally, their sizes (size meaning height or width, depending on ScrollView's orientation). How you set the sizes is exemplified throughout the demo scenes, as there are multiple ways of doing this, depending on how much variation in size you have in your data set + what you prioritize more: CPU, memory, the speed of the **`ChangeItemsCount`** call etc.
And put simply, you tell the adapter how many items there are and maybe their sizes, and that's all it needs to know in order to decide when to call **`[Create/Update]ViewsHolder`** for you. The way you store your data is your implementation detail, but enough utilities are already implemented for you in the **`Utils`** namespace, like the **`BaseParamsWithPrefabAndData`** class. When you call **`_Params.Data.Add`**, you're just adding an item to the list. The adapter should also be notified of this in order to update the list of sizes, update the content size and a lot of other things, that's why a very rigid item adding/removing convention is needed.
Alternatively (and this is what it's done in some of the demos), you can just create a helper method that does both of these things.

7. ***What is frame8? There seems to be a separate set of classes contained in the frame8 namespace***

It's our internal framework which is used in all of our projects (only a subset of it was included, to keep file count at minimum).

For v4.1 and up, its functionality was completely separated under its own folder in */Scripts/frame8* and some non-essential files were removed

8. ***How to scroll in both directions?***

The **`Scrollbar`** solution for **`OSA`** is **`ScrollbarFixer`**, but it only works in the main scrolling direction.

We'll take a vertical List as an example.

Scrolling in both directions is not yet implemented out of the box. The easiest way is to manually increase/decrease the Content's **`localPosition.x`** based on your own calculations. You can have 2 arrow buttons for adding discrete increments/decrements to **`localPosition.x`**.

However, the first quick solution may not be very satisfying, so you can alternatively experiment with adding a classic horizontal **`ScrollRect`** as the parent of **`OSA`**, and make **`OSA`**'s width large enough (at edit time, so constant) to encompass enough of its children's width.
Then add a usual horizontal **`Scrollbar`**, which will always be visible if you don't manually disable/enable it through code (because the **`OSA`**'s width will be constant).
Make sure **`OSA`** has **`ForwardDragToPrents`** toggled.
Make sure the parent **`ScrollRect`**'s content is **`OSA`**, and the viewport it's none (or it's set to **`ScrollRect`** itself).
Check Unity's docs if you're not very familiar with the built-in **`ScrollRect.`** The difference between the docs and our case is that you won't have a Viewport between the top **`ScrollRect`** and **`OSA.`**

Another solution is to use something very similar to the **`EdgeDragger`** script we include in some scenes (ContentSizeFitter example, horizontal async example). This is a draggable button that allows manually dragging a UI element's edge, so the user will choose how large it'll be. Of course, **`EdgeDragger`** may not have production-quality code, because it was only intended to be used in the demos, but it's very close to a final solution.

9. ***Why can't I access Time.time or Time.deltaTime ? (Error: 'float' does not contain a definition for 'time')***

Since v5.0, you can set **`Parameters.UseUnscaledTime`** to choose whether **`OSA`** can work even if the app's time is paused or not, so **`Time`** and **`DeltaTime`** properties were introduced and you should use those instead, for your own time-related tasks/animations.
Also, Time property has this name intentionally to prevent you from using **`UnityEngine.Time`** by accident when typing just **`Time`** . You should now use **`UnityEngine.Time`** explicitly if that's your intention

10. ***CollectItemsSizes doesn't work***

You're probably using **`BaseParamsWithPrefab`**. You need to either make your prefab's size to be smaller than any of the items you'll ever have, or simply use **`BaseParams`** and manually add the Prefab property which you'll use in **`CreateViewsHolder`**. **`BaseParamsWithPrefab`**'s main purpose is to make the prefab control the **`DefaultItemSize`**, but if you're overriding it anyway, **`BaseParamsWithPrefab`** is not needed, maybe just as a convenience for storing the **`ItemPrefab`** property for you.

In v5.1 (no ETA for publishing it yet) there's a **`PrefabControlsDefaultItemSize`** property in **`BaseParamsWithPrefab`** that you'd disable to achieve the same effect. I attached the new file here (need to be a *Verified OSA User*) which should be fine if you replace it in an **OSA** 5.0 project. And I actually recommend this instead. If the file is gone, request it via DM.

11. ***How can I force a ListView to redraw a ViewsHolder whose model's data has changed? i.e. force-call***

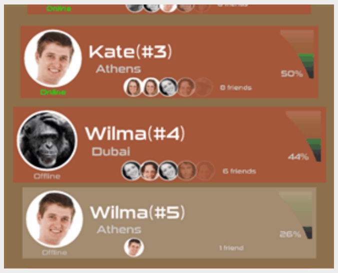
UpdateViewsHolder (which normally is called only when the holder becomes visible / is created)

Problem with relying on **`UpdateViewsHolder`** for this purpose is that if you're using **`ScheduleComputeVisibilityTwinPass`** in it (like the **`ContentSizeFitter`** example does), it'll not work. So for this reason there's no built-in method that you can call to have it call **`UpdateViewsHolder`** for you. But If you don't use **`ScheduleComputeVisibilityTwinPass,`** then you'll be fine (a) calling **`UpdateViewsHolder`** manually, or (b) even have an **`UpdateViews`** method in the **`ViewsHolder`** which you call directly.

If you use the **`ContentSizeFitter`** approach to have different item sizes, the solution is to use (b) and then call **`ForceRebuildViewsHolderAndUpdateSize`** on it.

In v5.1, there's **`ForceUpdateViewsHolderIfVisible`**, which takes care of everything, even rebuilding a single item's size when you use **`ScheduleComputeVisibilityTwinPass`** in **`UpdateViewsHolder`**.

12. ***How can I both scroll to- and resize an item at the same time? Like resizing the item with its center as pivot (currently, only the top and bottom edges can be stationary) AND centering it inside viewport:***



This is only available to v5.1 and up (see other FAQs above for how to grab one if it's not released yet).

You should be using **`ExpandCollapseAnimationState`** for the resizing animation, the same way it's exemplified in *MainExample.cs* (from the Demos extra package). You'll also see here what extra fields to add to your model in order to make it expandable, as well as the methods used in the code snippet at the end of this answer.

Either through inspector or in code, set **`Parameters.Animation.Cancel.SmoothScroll.OnSizeChanges = false`**. This makes sure a **`SmoothScroll`** in progress won't be cancelled by a size change in progress, which makes them run simultaneously.

Use the following code (it uses other methods from *MainExample.cs*, which presumably you've already copied in your **`OSA`** implementation when added the resizing):

```
void ToggleItemFocus(int index)
{
        float vpSize = (float)GetViewportSize();
        var model = <get model at index>;
        float itemNewSize = model.ExpandedAmount == 1f ? _Params.DefaultItemSize : GetModelExpandedSize(model);
        // Get half of the remaining viewport space as percentage
        float normalizedOffsetFromViewportStart = ((vpSize - itemNewSize) / vpSize) / 2f;
        SmoothScrollTo(index, <duration, same as resizing>, normalizedOffsetFromViewportStart, 0f, null, null, true);

        // "Click" the item through code
        var vh = GetItemViewsHolderIfVisible(index);
        OnExpandCollapseButtonClicked(vh);
}
```

13. ***How to calculate the ideal Image Pool capacity (FIFOCachingPool)?***

There are several approaches. It requires hard-coding some values that may need to change in a year or 2. This is usually done by each developer based on the info they have.

You can google the different ways you can get the estimated available free ram in bytes X, and then here's one suggestion:

```
float safePoolSizeBytes = Mathf.Min(X / 4f /*take only a quarter of the available ram*/, 250 * 1024 * 1024 /*And limit it to 250mb because we're good citizens*/);
float avgBytesPerImage = (4f/*PNG has 4 bytes per pixel*/ * averageWidth * averageHeight);

int safePoolCapacity = (int)(safePoolSizeBytes / avgBytesPerImage);
```

14. ***OSA initializes itself in Start(). Can I initialize it sooner? For example, I want to instantiate OSA from a prefab and then immediately set the data on the same frame.***

You can call **`OSA.Init()`** manually before setting the data. If you're calling it later or simply don't know if **OSA** did already initialize itself, check **`OSA.IsInitialized`** before.

A second solution would be to override **`Awake()`** and call **`Init()`** there. **OSA** will detect that and skip auto-initialization in its **`Start()`**. Note that a **`GameObject`**'s **`Awake()`** is called only if it's active, or at the moment of first activation.

15. ***How to have more items [created] than the number of visible ones? I.e. more than the OSA's calculated <visibleCount + 1>***

OSA makes sure it only creates the minimum necessary objects to display your data. However, if you want to create more, one way of doing it is simply enlarging the Viewport and also increasing the padding in those directions by the same value. That way, your first item will still *appear* first, but it won't immediately be recycled when goes out of the view (assuming you're using the default setup with a **`Mask`** on the ScrollView object), because that's actually done when the item goes out of the *viewport*; and since the viewport is now larger, items need to travel a greater distance before being recycled, leading to what you want: more game objects are created than the visible ones.

Concrete example: Consider a vertical ScrollView. Viewport is initially identical to the ScrollView itself. Now move the Viewport's Top edge 200 units up and its Bottom edge 200 units down. Increase the ContentPadding Top/Bottom by the same amounts. Voila! Look in the hierarchy at runtime to see the results.

16. ***Is there a way to pre-instantiate a set number of list items (ie. 20-50) on Start() and have the items persist even when I call ResetItems(0)?***

OSA v5.0 has **`CreateBufferredRecycleableItems()`** and **`AddBufferredRecycleableItem()`**, which allow you to pre-instantiate some items and keep them in memory to avoid any subsequent object creation/destruction. For this, a special "bufferred recycleable items" internal list was created, apart from the "recycleable items", and it contains items that won't be directly destroyed, unless they're "promoted" to the "recycleable items" list.

You can pass a custom index for the views holders: useful when having multiple prefabs and you want to distinguish between them, since **`CreateViewsHolder(int)`** is called for these. OSA calls it with -1 by default, but in case of multiple prefabs, you'll use a different negative value for each prefab type.

Something similar can also be achieved by setting a higher BaseParams.optimization.RecycleBinCapacity, which previously didn't work, but was fixed in v5.0.

As for persisting those game objects between important events like count changes, there are some parameters to control this under **`Params.optimization`**: **`KeepItemsPoolOnLayoutRebuild`**, **`KeepItemsPoolOnEmptyList`**. As of 19.07.2020, grids ignore **`KeepItemsPoolOnLayoutRebuild`** if the layout has rebuilt results in a different number of cells per group.

Ok, but how about pre-instantiating items at edit-time and using those, i.e. no instantiate allocations to make at runtime?

This is something that'll be added in 5.1.3+ (not yet released ATM of writing - 06.11.2020). You'll basically be able to supply the instance of your root `GameObject` for a particular ViewsHolder inside `CreateViewsHoler()`, instead of the prefab. This is done by using `vh.InitWithExistingRoot()`:

```
[SerializeField]
List<RectTransform> _PreMadeViews = null;

/*some code here*/

protected override void OnInitialized()
{
        base.OnInitialized();

        // Add our pre-made views
        var vhs = CreateBufferredRecycleableItems(_PreMadeViews.Count);
        AddBufferredRecycleableItems(vhs);

        // We don't want the buffered items to be destroyed during the OSA's lifetime
        _Params.optimization.KeepItemsPoolOnEmptyList = _Params.optimization.KeepItemsPoolOnLayoutRebuild = true;
}

protected override MyItemViewsHolder CreateViewsHolder(int itemIndex)
{
        var instance = new MyItemViewsHolder();
        if (itemIndex < 0)
        {
                // Called because of our CreateBufferredRecycleableItems() call

                var preMadeRoot = _PreMadeViews[_PreMadeViews.Count - 1];
                _PreMadeViews.RemoveAt(_PreMadeViews.Count - 1);
                instance.InitWithExistingRoot(preMadeRoot, _Params.Content, itemIndex);
        }
        else
        {
                // Called by OSA when it needs more views to be created

                // You can decide to throw an exception or simply fallback to instantiating a new one at runtime

                //instance.Init(_Params.ItemPrefab, _Params.Content, itemIndex);
                throw new InvalidOperationException("Increase the number of pre-instantiated items");
        }

        return instance;
}
```

17. *How to get the currently snapped/focused item when using Snapper8?*

OSA 5.1 added `OSASnapperFocusedItemInfo` util script. The code is really simple and you can copy it directly into your OSA implementation, but it's provided as a separate script for convenience.

18. *How to 'decorate' the ScrollView with an additional view that's not necessarily related to an item?*

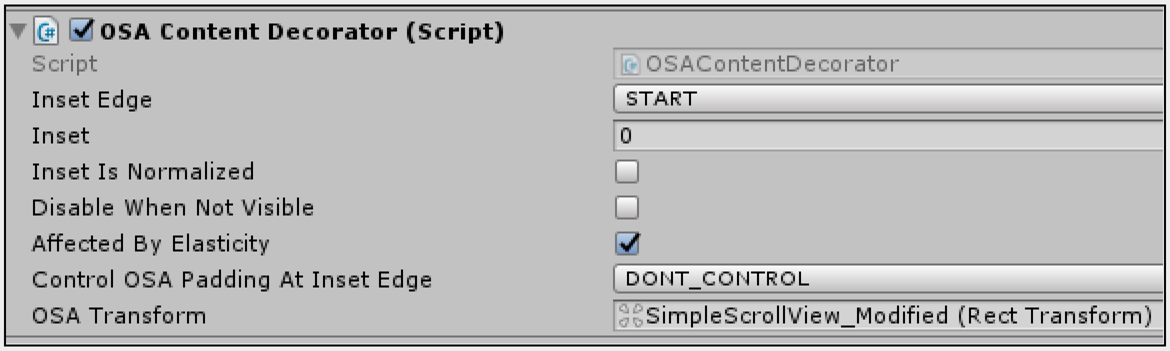
For example, show an advertisement-like view that scrolls with the content

The way it works is you'll have an empty item in your data set that will act as the space required for the ad. The ad will make use of the **`OSAContentDecorator`**, which is a versatile component.

Steps:

Add your ad as a child of Viewport;

Add **`OSAContentDecorator`** to it with this params:



OSA Content Decorator (Script)

Script	OSAContentDecorator
Inset Edge	START
Inset	0
Inset Is Normalized	☐
Disable When Not Visible	☐
Affected By Elasticity	✓
Control OSA Padding At Inset Edge	DONT_CONTROL
OSA Transform	SimpleScrollView_Modified (Rect Transform)

Keep a reference to the decorator component in your OSA implementation - for ex., assign it to a field.

Keep your ad object deactivated from edit-time. Activate it at runtime (in code) before the first time you set the items. This way, you won't see it just sitting there. Alternatively, you can set the Inset to a value like -1000.

Let's say the ad should appear below item 32 (33th item) as item 33 (34th item). This means that the original item 33 in your set will become item 34, because we'll insert the empty item at 33. If you keep your data in a plain C# list, use **`list.Insert(adIndex, model)`**. If you use DataHelpers, use **`dataHelper.Insert(adIndex, model)`**. It's better to add the empty model in the C# list before submitting the change to the DataHelper, so OSA will just refresh once at init.

After OSA was initialized, we want the placeholder item to be resized so that the ad will fit into it. This can be done after you've set the items the first time:

```
var adRT = decorator.transform as RectTransform;
var adSize = adRT.rect.height; // or width, if using a horizontal scrollview
RequestChangeItemSizeAndUpdateLayout(adIndex, adSize)
```

Override **`OnScrollPositionChanged`** and set keep the decorator's inset updated like this:

```
protected override void OnScrollPositionChanged(double normPos)
{
        base.OnScrollPositionChanged(normPos);

        if (GetItemsCount() <= adIndex)
                // Items not set yet or just too few items exist, which don't include the ad
                return;

        float adInset = (float)GetItemVirtualInsetFromParentStart(adIndex);
        decorator.SetInset(adInset);
}
```

If you're using variable-sized items, you'll need to skip executing the size-changing code for the ad. Not only this, but any change done to the placeholder is unnecessary, so you'll need to not execute it at all:

```
protected override void UpdateViewsHolder(MyItemViewsHolder newOrRecycled)
```

```
{
        if (newOrRecycled.ItemIndex == adIndex)
                return;

        // Code for normal items
        ...
}
```

You may further want to disable the children of your item, if it's an ad, and enable them otherwise (you cannot directly disable/enable the item because OSA manages that). This is needed if your ad doesn't cover the placeholder item completely.

If **`OSAContentDecorator.SetInset()`** doesn't exist, you'll need to update OSA or (if the live version is not yet available) contact us on discord to get a preview package as this method was added after v5.1.2.

That should be it.

19. ***How to jump to the next/previous items when using snapping?***

So snapping usually means you'll have a **`Snapper8`** script on your OSA object. This means that there's always a snapped or "focused" item. A good example is a gallery-like horizontal ScrollView where you're focusing the middle item and you have other items around it. A very common need is to be able to programmatically snap to the next item and that's where you'll use the well-known **`SmoothScrollTo()`**, or just **`ScrollTo()`** if you want it to be immediate.

For example, you're waiting in **`Update()`** for a key press, then get the currently focused item, and scroll to the next, if it exists:

```
protected override void Update()
{
        base.Update();

        if (Input.GetKeyUp(KeyCode.RightArrow))
        {
                var snapper = GetComponent<Snapper8>();

                float _;
                var snappedVH = snapper.GetMiddleVH(out _);
                if (snappedVH == null)
                {
                        // Probably there are no items
                        return;
                }

                int nextItemIndex = snappedVH.ItemIndex + 1;
                if (nextItemIndex < GetItemsCount())
                {
                        // No more items after the current one
                        return;
                }

                bool cancelCurrentAnimationIfAny = true;
                // We're passing snapper.viewportSnapPivot01 and snapper.itemSnapPivot01 to preserve the snap position 
specified in the snapper (which may not be in the middle, if the user chooses to)
                SmoothScrollTo(
                        nextItemIndex,
                        .3f,
                        snapper.viewportSnapPivot01,
                        snapper.itemSnapPivot01,
                        null,
                        null,
                        cancelCurrentAnimationIfAny
                );
        }
}
```

20. ***How do I use the "New Input System"?***

Unity documentation is fine, but we had trouble finding it. In any case, we tested OSA 5.3.1 with Unity 2019.4 LTS and it works fine with either input system. You just need to carefully read the Installation and the first part of UI Support sections. Once the documentation was found, it took 5 mins to set it up.

Note: for maximum compatibility, the demo scenes weren't changed to support the new system, but you can easily modify any of them - you only need to migrate the EventSystem object in them using the above instructions.

Small note: We initially tested it with 2019.1, but this time Unity didn't automatically create/assign the default Input Actions asset for us, so if you're stuck with this version, check further down the UI Support page so set up your own.

21. ***How to prevent the interaction with the items while scrolling/dragging? I want to prevent button clicks while the content is moving.***

A solution would be adding an invisible Image at the same level in the hierarchy as the Content game object and only keeping it enabled while **`OSA.Velocity > 0`**. The result is the image will block any interaction with the Content object while it's scrolling, but will still forward drag events to OSA so that dragging will work fine.

I imagine **`CutMovementOnPointerDown`** can be left ON, but if you have problems in your particular case, try turning it OFF.

13. Help improve OSA by leaving a review

Your review will lead to more users purchasing the asset, and this indirectly provides the necessary resources to improve it and keep up with your feature-requests!

And you'll have my everlasting gratitude. 😊

-- Lucian