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'*

USAGE INSTRUCTIONS

`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 i`t'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 two things will happen:

    5.1. **`CollectItemsSizes()`** will be called (which you can optionally implement to provide your own sizes, if known beforehand

    5.2 **`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 sandalone platforms)

EXAMPLE SCENES & UTILITIES

All the example scenes & the utlity 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

- 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.