namespace frame8.Logic.Misc.Visual.UI.ScrollRectItemsAdapter:

ScrollRectItemsAdapter8 <TParams, TItemViewsHolder>

Description

Base abstract class that you need to extend in order to provide an implementation for **GetItem[Height/Width](int index)** and **InitOrUpdateItemViewHolder(TItemViewsHolder newOrRecycled).**

You must first extend **BaseItemViewsHolder**, so you can provide it as the generic parameter **TItemViewsHolder** when implementing **ScrollRectItemsAdapter8.**

Extending **BaseParams** is optional. Based on your needs. Provide it as generic parameter **TParams** when implementing **ScrollRectItemsAdapter8.**

Properties & fields

```
// The params passed in Init(TParams parms)
public TParams Parameters { get; }

// Returns a copy of the current visible items list
public List<TItemViewsHolder> VisibleItemsCopy { get; }

// The params passed in Init(TParams parms)
protected TParams _Params;

// List of instantiated view holders for current visible items
protected List<TItemViewsHolder> _VisibleItems;

// Number of currently visible items
protected int _VisibleItemsCount;
```

Methods

```
// Parameterless, empty constructor
protected ScrollRectItemsAdapter8();

// Initializes the adapter and calls GetItemHeight(int index) for each item immediately
// After a few frames, it calls InitOrUpdateItemViewHolder(TItemViewsHolder newOrRecycled)
// for each visible item in order to initially populate the view
public void Init(TParams parms);

// Call this anytime the data set changes and the views need to update. For example, when
// adding/removing items, when modifying existing items etc. This will trigger another batch
// calls to GetItemHeight(int index) for each item in the same frame
// After a few frames, it calls InitOrUpdateItemViewHolder(TItemViewsHolder newOrRecycled)
// Equivalent to calling ChangeItemCountTo(int, false)
public virtual void ChangeItemCountTo(int itemsCount);


// Same as ChangeItemCountTo(int), but if contentPanelEndEdgeStationary=true, the content panel
// will be resized by moving its start (TOP if vert. ScrollView/LEFT if hor. ScrollView) edge. Useful, for
// example, if you want to add items at the head of the list but also freeze the current visible
// items inside viewport.
public virtual void ChangeItemCountTo(int itemsCount, bool contentPanelEndEdgeStationary);
```

```
// In some cases you may want to modify the views of an item that you're sure it's visible
// Returns null if the item with index indexOfITemToGet is not visible
public TItemViewsHolder GetItemViewsHolderIfVisible(int indexOfITemToGet);

// Same as the other version, but you can use the root of the viewholder, if the item's index is unknown.
public TItemViewsHolder GetItemViewsHolderIfVisible(RectTransform withRoot);

// Will call GetItem[Height|Width](itemIndex) for each other item to have an updated sizes cache
// After, will change the item's size with <newSize> and will shift down/right (or top/left, if
// itemEndEdgeStationary=true) the next ones, if any.
// returns the resolved size. This can be slightly different than <requestedSize> if the number of items
// is huge (>100k)).
// HINT: Very useful for expand/collapse animations!
public float RequestChangeItemSizeAndUpdateLayout(TItemViewsHolder withViewHolder, float requestedSize,
bool itemEndEdgeStationary=false);

// Returns the distance of the iten's left (if scroll view is Horizontal) or top (if scroll view is Vertical)
// edge from the parent's left (respectively, top) edge.
public float GetItemOffsetFromParentStart(int itemIndex);

// This is called automatically when screen size (or the orientation) changes
// But if you somehow resize the scrollview manually, you also must call this
public void NotifyScrollViewSizeChanged();

// Call it when you're done
public virtual void Dispose();

// Only if the ScrollRect's scroll type is vertical: Called at initialization and anytime you call
// ChangeItemCountTo(int itemsCount)
protected abstract float GetItemHeight(int index);

// Only if the ScrollRect's scroll type is horizontal: Called at initialization and anytime you call
// ChangeItemCountTo(int itemsCount)
protected abstract float GetItemWidth(int index);

// This will be called ONLY for the items currently visible and each time a new one will become visible:
// --- use newOrRecycledViewsHolder.itemIndex to get the item index, so you can retrieve its associated data
// model from your data set
// --- newOrRecycledViewsHolder.root will be null if the item is not recycled. So you need to instantiate
your
// prefab (or whatever), assign it and call newOrRecycledViewsHolder.CollectViews()
// --- newOrRecycledViewsHolder.root won't be null if the item is recycled. This means that it's assigned
a
// valid object whose UI elements only need their values changed
// ---update newOrRecycledViewsHolder's views from its associated data model
protected abstract void InitOrUpdateItemViewHolder(TItemViewsHolder newOrRecycled);

// Self-explanatory. The default implementation returns true each time
protected virtual bool IsRecyclable(TItemViewsHolder potentiallyRecyclable, int
indexOfItemThatWillBecomeVisible, float heightOfItemThatWillBecomeVisible)
```

BaseItemViewsHolder:

Description

You *must* extend this class in order to provide it as a generic parameter to your implementation of **ScrollRectItemsAdapter8**, along with **BaseParams** or an implementation of **BaseParams.**

Properties & fields

```
// Used internally. Don't change it!
// formerly "cachedHeight". renamed in v2.0. you should not use it anyway
public float cachedSize;

// The index of the item designated by this view holder
public int itemIndex;

// The root of the item's views
public UnityEngine.RectTransform root;
```

Methods

```
// Call this after you've assigned the first or a new root to the views holder
// Provide your own implementation of it, if you also have other views (not just an empty root).
// And you will, in almost all cases.
public virtual void CollectViews();
```

BaseParams:

Description

You may or may not extend this in order to use it with your implementation of **ScrollRectItemsAdapter8,** depending on what data do you want to provide to **ScrollRectItemsAdapter8.** For example, you may want to provide it with a prefab for the items, if they all share the same view hierarchy, so you can easily access it in **ScrollRectItemsAdapter8.InitOrUpdateItemViewHolder** method in order to instantiate new items.
Mark it as **Serializable** and use it as a **MonoBehaviour's** field, in order to populate it in inspector (just a suggestion. You can easily populate it by code instead)

Properties & fields

```
// How much objects to always keep in memory, no matter what. This includes visible items + items in the
// recycle bin. The recycle bin will always have at least one item in it, regardless of this setting. Set
// to -1 or 0 to detect automatically (Recommended!). Change it only if you know what you're doing (usually,
// it's the estimated number of visible items + 1).
// Last note: this field will only be considered after the number <visible views + views the recyclebin>
// grows past it
public int minNumberOfObjectsToKeepInMemory = -1;

// See BaseParams.UpdateMode enum for full description. The default is ON_SCROLL_THEN_MONOBEHAVIOUR_UPDATE
// and it's the most stable.
public UpdateMode updateMode = UpdateMode.ON_SCROLL_THEN_MONOBEHAVIOUR_UPDATE;

// These 3 fields are assigned in inspector or by code. Just make sure they are all assigned and the
// Scroll View you've created in Unity Editor has the same structure as the default one
// (created by GameObject->UI->Scroll View)
public ScrollRect scrollRect;
public RectTransform viewport;
public RectTransform content;

// UPDATE: removed in v 2.0.
// For internal use only! Please don't change it manually.
// Call ScrollRectItemsAdapter8.ChangeItemCountTo(int) instead, which will trigger the natural
// flow of updating the items' views.
public int itemsCount; // removed in v 2.0

// Self-explanatory (up to v2.1 the padding was taken from a LayoutGroup component attached to the content
// panel. This is not the case anymore)
public RectOffset contentPadding;
```

```
// Self-explanatory (up to v2.1 the spacing was taken from a LayoutGroup component attached to the content
// panel. This is not the case anymore)
public float contentSpacing;
```

Methods

```
// We need an parameterless constructor in the case we need the class to be serializable
public BaseParams();

// Copy constructor
public BaseParams(BaseParams other);

// Sets the scrollrect itself as viewport and content as scrollRect.content (make sure it’s NOT null ☺ )
public BaseParams(ScrollRect scrollRect);

// Used when you’re populating the params by code, instead of serializing this class and assigning values
// in inspector. If viewport is null, sets the scrollrect as viewport. If content is null, sets content as
// scrollRect.content (make sure it’s NOT null ☺ )
public BaseParams(ScrollRect scrollRect, RectTransform viewport, RectTransform content, int itemsCount);

// Make sure to replace null-values with default ones.
// Does the same checks as BaseParams(ScrollRect, RectTransform, RectTransform, int)
internal void InitIfNeeded();

// For internal use
internal float GetAbstractNormalizedScrollPosition();
internal void ScrollToStart();
internal void ClampScroll01();
```

BaseParams.UpdateMode:

Description

How/When to update the items layout

Properties & fields

```
// Updates are triggered by a MonoBehaviour.Update() (i.e. each frame the ScrollView is active) and at each
// OnScroll event.
// Moderate performance when scrolling, but looks best.
ON_SCROLL_THEN_MONOBEHAVIOUR_UPDATE,

// Updates ar triggered by each OnScroll event
// Experimental (somewhat - don’t be afraid to test it). However, if you use it and see no issues, it's
// recommended over ON_SCROLL_THEN_MONOBEHAVIOUR_UPDATE.
// This is also useful if you don't want the optimizer to use CPU when idle.
// A bit better performance when scrolling.
ON_SCROLL,

// Update is triggered by a MonoBehaviour.Update() (i.e. each frame the ScrollView is active)
// In this mode, some temporary (they disappear almost immediately) gaps appear when fast-scrolling. If this
// is not acceptable, use other modes.
// Best performance when scrolling.
MONOBEHAVIOUR_UPDATE
```