EnhancedScroller User Manual

The EnhancedScroller is a plugin extension to Unity UI's ScrollRect. It offers many features, including:

- Recycling of game objects without garbage collection for better memory and processor performance
- Optional infinite loops
- Jumping to an index
- A true MVC solution, separating the concerns of data, controllers, and display
- Dynamically data driven so you don't have to know or have your data prepared at design time
- Object count grows dynamically with the size of the scroller, adding elements as necessary
- Optionally snapping to any location within the scroller's view area, additionally centering the cell at any point in the cell's span.

Setup

Please refer to the Quick Start guide to see how to set up the EnhancedScroller in your project. Also, the plugin comes with several demos that are fully source documented to help you see the different ways it can be used.

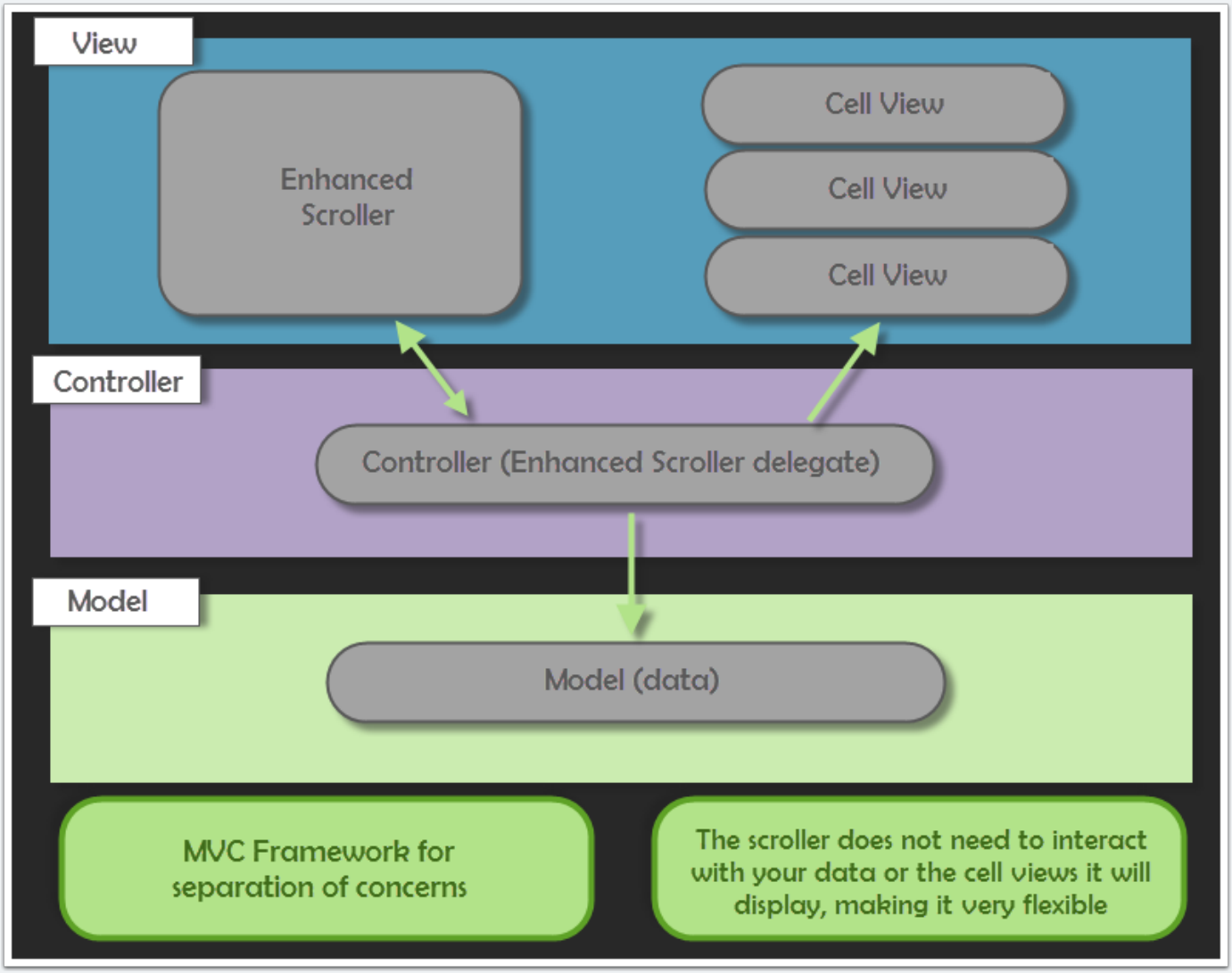
How it works

The EnhancedScroller is an MVC solution that allows you to control your game logic, data, and cell views outside of the scroller, letting you have more freedom over how you can apply the plugin. The scroller expects a delegate controller which it will rely upon to provide it information about your data and views. Specifically, the three required handlers needed in your controller are:

- **GetNumberOfCells**: This handler will tell the scroller how many cells you will need. The scroller can calculate the correct bounds and handle positioning based on this count

- **GetCellViewSize**: This handler tells the scroller the size (height for vertical scrollers, width for horizontal scrollers) for each cell you will display. Note you can return any value here, whether each cell has a different size, or they all share the same size.
- **GetCellView**: Here you will pass a prefab cell view to the scroller. If the scroller has already created a cell of this type before and has placed the cell in the recycled container, it will reuse the cell. This cuts down on garbage collection which can be a costly process on low end devices like mobiles. Once the cell has been created, you can call functions on the view to set up its connection to the data.

The scroller only communicates with your controller. Your controller will handle the data and cell views, keeping a nice separation of concerns in an MVC manner.



Additional delegates

In addition to the three required handlers in your controller, you can also tap into other delegates the scroller will publish to:

- **cellViewVisibilityChanged**: This delegate fires whenever one of the cells in the scroller is either shown or hidden. When the cell is shown, it is either created anew or recycled from the cell objects that already exist. When the cell is hidden it is sent to the recycled container.
- **scrollerScrolled**: This delegate fires whenever the scroller's ScrollRect changes its scroll position. This can be useful if you want to make changes to the display of your cells based on where the scroll position is.
- **scrollerSnapped**: This delegate fires whenever the scroller snaps to a location you specify.
- **scrollerScrollingChanged**: This delegate fires whenever the scroller starts or stops scrolling (the scrollRect's velocity changes from zero to positive or vise versa). This does not fire when tweening changes.
- **scrollerTweeningChanged**: This delegate fires whenever the scroller starts or stops tweening.

Usage

You will usually want to set up your data in your controller, but this is not a requirement so long as your controller has access to the data. Once you have access to the data, your controller will call the ReloadData on the EnhancedScroller. This lets the scroller allocate space for all the elements, displaying only the ones it needs to in any given moment.

When the scroller requests GetCellView from your controller, you will pass it a prefab of the cell view you need. You can have multiple prefabs for your scroller. For instance, you may want to break your data into groups with headers, rows, and footers, requiring three cell prefabs. You could even have a different prefab for each cell, though that would

probably be unlikely. Most scrollers will probably only handle a single prefab, but that is entirely up to you.

Once you have the cell that is created by the scroller, you can do whatever you need to it. This is usually when you set up the cell's view state, updating UI elements based on the data for the cell.

***Note**: if you are using coroutines in your cell view's setup, you will receive errors if you try to call this setup from the GetCellView handler. This is because the coroutine is launched on a separate thread and your cell will try to set elements before they are made available by the scroller's recycle mechanism. If you need to use coroutines in your cell view setup, you will need to move your cell setup from the GetCellView to the cellVisibilityChanged handler. This will ensure the cell has become fully available by the scroller before you try to make UI modifications.*

Please see the fully documented demos that come with the plugin for examples of how use the EnhancedScroller in different situations.

Settings

The EnhancedScroller comes with some settings that can be applied in different situations

- **scrollDirection**: This will determine whether the scroller is vertical or horizontal
- **spacing**: This adds in a spacing element between each cell view. If looping is off, spacing will start after the first element and end before the last element, otherwise it will be between all elements.
- **padding**: This offsets the cell views from the edges of the scroller: left, right, top, bottom.
- **Loop**: If on, this will create an infinite loop of elements.
- **ScrollbarVisibility**: This determines how the scrollbar should be displayed. If no scrollbar is attached to the scroller, then this setting is ignored.

    - **OnlyIfNeeded**: The scrollbar will show if there are more elements than can be displayed in the viewable area, unless looping is turned on.
    - **Always**: The scrollbar will always show even if it doesn't need to.
    - **Never**: The scrollbar will always be hidden.

Special features

The EnhancedScroller has some built-in optional features you can use

- **Looping**: This will create an infinite list of objects that will wrap around to the start when it reaches the end of your data. Behind the scenes, the scroller is actually creating room for three sets of the data, attempting to keep the visible range within the middle set, jumping back or forward as necessary.
- **Snapping**: This will stop the scroller from moving when a certain velocity threshold has been met. When the scroller stops, it will take the cell located at the scroller offset and snap it to that location. In addition, you can specify where in the cell the snapping should center on. This lets you do unique things like centering a cell in the middle of the scroller.

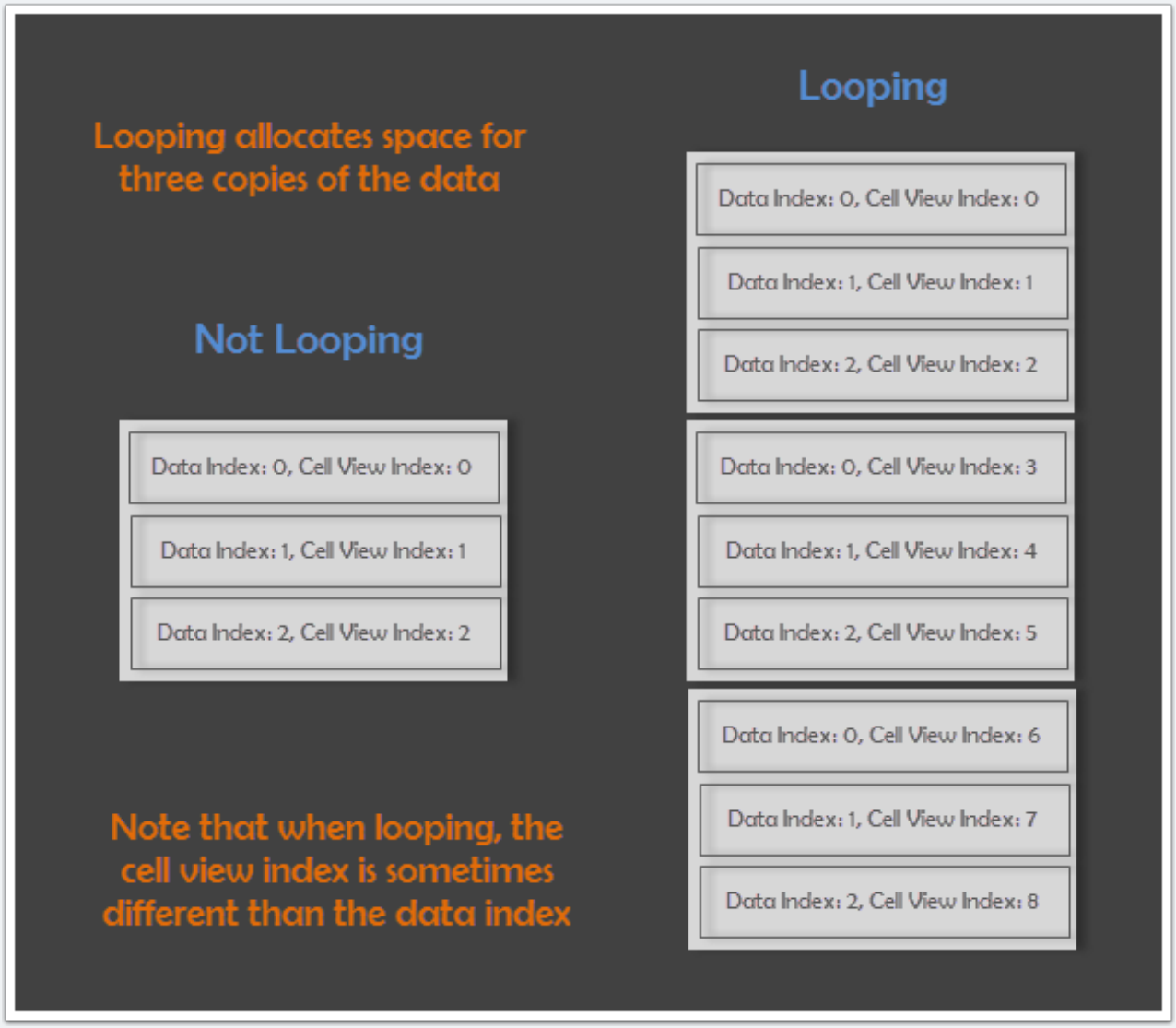
Looping

When the EnhancedScroller is looping it actually creates slots to accommodate three sets of data. Note that it does not duplicate the data or cell views, it only allocates space to fit them. When looping is on, the cell view index will indicate the index of any given cell with the additional slots. The data index will be repeated three times.

You can easily calculate the dataIndex from any cellViewIndex by the following formula:

dataIndex = cellViewIndex % numberOfCells

The dataIndex is the modulus of the cellViewIndex and the numberOfCells.



Snapping

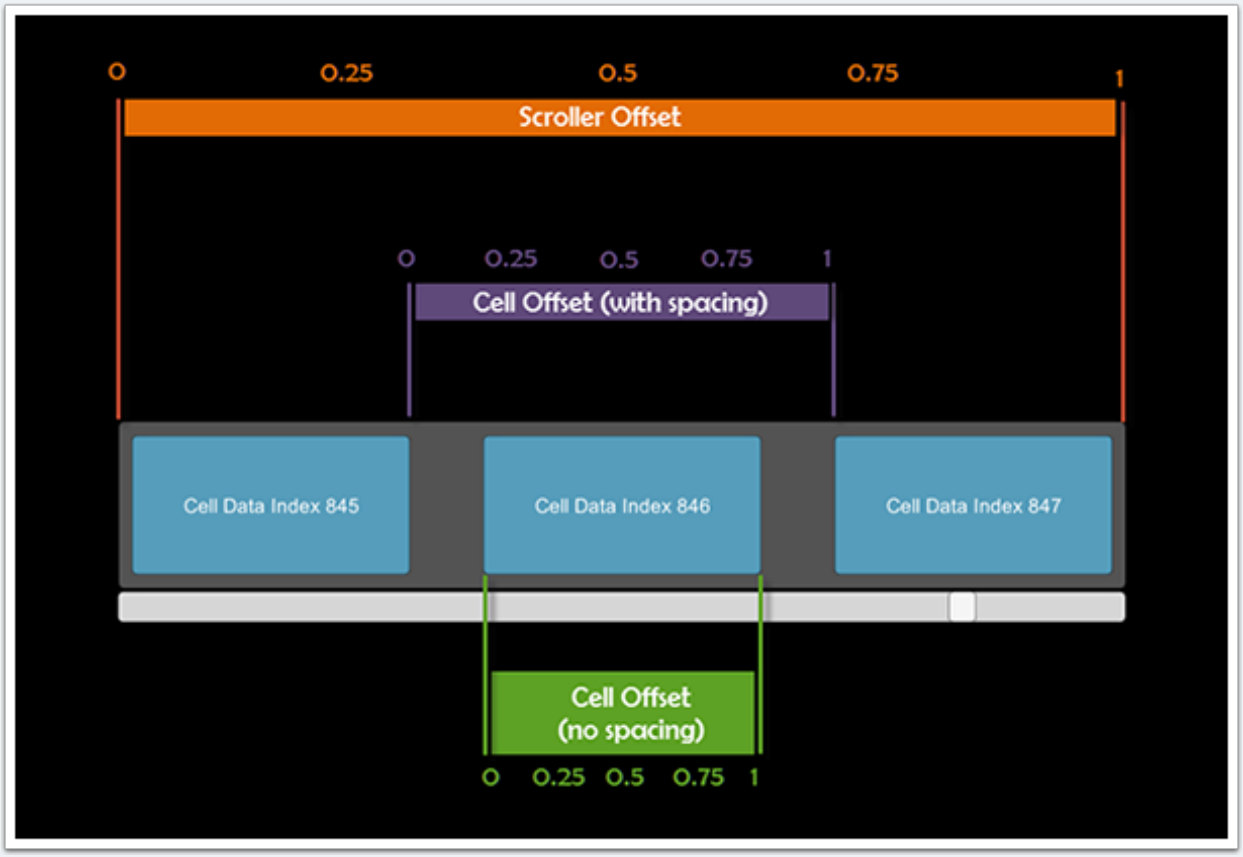
***Note**: snapping does not work when trying to scroll using a scrollbar. It will only work when you are dragging on the ScrollRect or applying a velocity manually to the scroller.*

If you want to turn off snapping when the scrolling is being dragged, please search the FAQs on the EnhancedScroller's forum which can be found at echo17.com.

Snapping will automatically occur when snapping is turned on and the scroller's velocity drops below the threshold. You can also call Snapping directory through the **Snap** function of the EnhancedScroller class.

Snapping has several options elaborated here

- **snapping**: This is a toggle that turns on and off snapping. When it is on, the scroller will snap based on the other settings.
- **snapVelocityThreshold**: This is the speed the scroller will use to start the snap. When the scroller's velocity drops below this threshold, it will initiate the snapping.
- **snapWatchOffset**: This is the location in the scroller that is used to determine which cell to snap to. This value is typically in the range 0..1, with 0 being the top / left of the scroller and 1 being the bottom / right. For instance if you wanted to watch for cells in the middle of the scroller, you would set this value to 0.5. When the scroller slows down to its snapping speed, the watch offset will be used to determine which cell needs to be snapped. (See the *Scroller Offset* in the diagram below)
- **snapJumpToOffset**: This is the location in the scroller that is used to determine where the snapped cell should be moved to. This is typically the same as the snapWatchOffset, since you would most likely be snapping to the same location you are watching. These two values are separated in case your application needs that functionality. (See the *Scroller Offset* in the diagram below)
- **snapCellCenterOffset**: Once the cell has been snapped to the snapJumpToOffset location, the center offset will be used to center the cell on that location. This can be useful if you want your cell to be centered, top/left aligned, bottom/right aligned, or some other alignment. Like the snapWatchOffset and snapJumpToOffset, this value is typically in the range 0..1 with 0 being the top/left of the cell and 1 being the bottom/right. (See the *Cell Offset* in the diagram below)
- **snapUseCellSpacing**: If this value is true, the snapCellCenterOffset will include the spacing between cells when determining where to center the cell. (Note the two different *Cell Offsets* in the diagram below)
- **snapTweenType**: What function to use when interpolating between the current scroll position and the snap location. This is also known as easing. If you want to go immediately to the snap location you can either set the snapTweenType to immediate or set the snapTweenTime to zero.
- **snapTweenTime**: The time it takes to interpolate between the current scroll position and the snap location.



Jumping

You can jump to a particular data index in your EnhancedScroller by calling **JumpToDataIndex** with some settings

- **dataIndex**: The index of the data set to jump to.
- **scrollerOffset**: The location in the range 0..1 where the data index should be centered on. See diagram in Snapping section for an explanation.
- **cellOffset**: The location in the range 0..1 where the cell should be centered on. See diagram in Snapping section for an explanation.
- **useSpacing**: Whether to use spacing when calculating the cell center offset.

- **tweenType**: What easing to use when interpolating between the current position and the jump location.
- **tweenTime**: How long to interpolate the jump.
- **jumpComplete**: This delegate is called when the jump has completed its tween.

Other properties and methods

- **ScrollPosition**: This property gives the current position of the scroller based on the top (for vertical scrollers) or left (for horizontal scrollers). The scroll position is in pixels for most canvas settings.
- **Velocity**: This is the 2D value for how fast the scroller is moving. Typically, only one axis will be moving at a time, so it is usually better to use the LinearVelocity.
- **LinearVelocity**: This is the 1D value for how fast the scroller is moving along its one axis.
- **IsScrolling**: This is true if the velocity is greater than zero (does not apply when tweening).
- **IsTweening**: This is true if the scroller is tweening between two positions.
- **StartCellViewIndex**: The first cell view index being displayed. The cell view index may be different than the data index if looping is occurring because there are more cell slots when looping. (See *Looping* section)
- **EndCellViewIndex**: The last cell view index being displayed. The cell view index may be different than the data index if looping is occurring because there are more cell slots when looping. (See *Looping* section)
- **StartDataIndex**: The first data index being displayed. The cell view index may be different than the data index if looping is occurring because there are more cell slots when looping. (See *Looping* section)
- **EndDataIndex**: The last data index being displayed. The cell view index may be different than the data index if looping is occurring because there are more cell slots when looping. (See *Looping* section)
- **NumberOfCells**: This is the number of data cells as provided by the controller delegate.
- **ScrollRect**: This is a convenience property that links to the the Unity ScrollRect component in case you need direct access.
- **ScrollRectSize**: This is the height (for vertical scrollers) or width (for horizontal scrollers) of the ScrollRect's visible area.
- **ToggleLoop**: This just sets the looping to the opposite of what it was.
- **GetScrollPositionForCellViewIndex**: This returns the scroll position of a given cell view index. The cell view index is used instead of the data index in case looping is on and you need access to a cell outside of the middle section. (See *Looping* section)

- **GetScrollPositionForDataIndex**: This returns the scroll position of a given data index. (See *Looping* section)
- **GetCellViewIndexAtPosition**: This returns the cell view index at the position specified. If you want the dataIndex, just use the formula: dataIndex = cellViewIndex % NumberOfCells.
- **RefreshActiveCellViews**: This calls RefreshCellView on each active cell that is visible in the scroller. You will need to override the RefreshCellView method in your cell views to do the actual update of the UI.

Cell View Methods

- **RefreshCellView:** This method will be called whenever **RefreshActiveCellViews** method is called on the scroller. You will likely need to store the cell's data when you first set it. That way whenever the RefreshCellView method is called it can reference this stored data to update the UI. See demo 7 for an example of this.