Custom ItemRenderers

Ryan Frishberg and Joan Lafferty

MAX 360 – 11/19/08

Introduction and Resources

Grandfather to Custom ItemRenderers : Alex Harui

Additional Resources:

Alex's Blog: http://blogs.adobe.com/aharui
Peter Ent's Blog: http://weblogs.macromedia.com/pent

Agenda

- Types of ItemRenderers
- Virtualization of ItemRenderers
- Common Problems and Solutions
  - Using States, Editors, and Reusable itemRenderers.
- Building for Performance
- ItemRenderers in Flex 4 (Gumbo)

itemRenderer

- A list renders data, usually in a label
- You can use item renderers to render the data however you want
- itemRenderer is a property on List classes typed to take an IFactory.

```
<mx:List itemRenderer="MySuperSweetItemRenderer">
    <mx:dataProvider>
        ...
    </mx:dataProvider>
</mx:List>
```

Types of ItemRenderers

- Two main types:
  - Custom Item Renderers
  - Drop-in Item Renderers (IDropInListItemRenderer)
- Three ways to create them:
  - New ActionScript Class
  - New MXML Class
  - Inline

Custom ItemRenderers

- Most basic item renderer
- What developers use most of the time
- Your item renderer is specific to the data you want to display.
- Item renderer implements IDataRenderer

```
<?xml version="1.0" encoding="utf-8"?>

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Label text="{data.lastName + ', ' + data.firstName}" />

</mx:VBox>
```

Creating a custom item renderer

- Three ways to create them:
  - New ActionScript Class
  - New MXML Class
  - Inline

Drop-in ItemRenderers

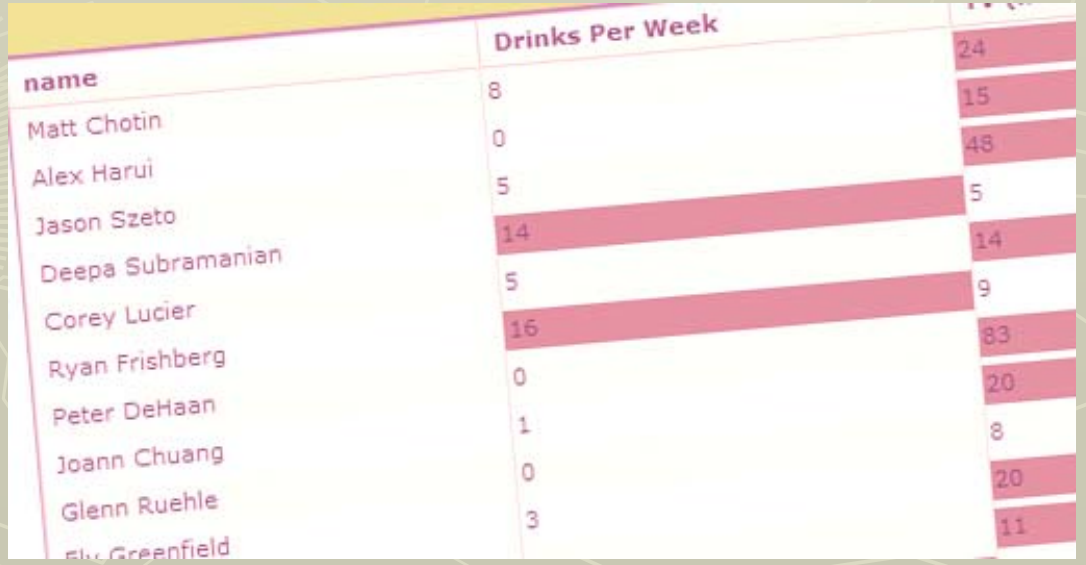
- What most Flex framework components implement
- Drop in item renderers implement IDropInListItemRenderer (a.k.a. – ISchemaLessListItemRenderer)
- Doesn't just give you the data property, but it gives you the listData
- Why do you need this? How to create an item renderer for multiple DataGrid columns?
- You can "drop in" the item render, and it'll just display the data—no matter what the data is

Reusable Renderers

- Most itemRenderers are written to associate a renderer with a column.

```
<mx:Label text="{data.firstName}" fontStyle="italic"/>
```

- To make an itemRenderer reusable, do not associate it with a column.
- Use the listData property on your itemRenderer to determine what data you are representing.



Reusable Renderers: Using listData

- To use listData, your itemRenderer must implement IDropInListItemRenderer.
- Components like CheckBox, Button, NumericStepper, Text and DateField already implement IDropInListItemRenderer.
- Containers do not implement IDropInListItemRenderer.

Implementing IDropInListItemRenderer

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.controls.listClasses.IDropInListItemRenderer“>

    <mx:Script>
        <![CDATA[

            import mx.controls.listClasses.BaseListData;

            private var _listData:BaseListData;
            public function get listData() : BaseListData
            {
                return _listData;
            }
            public function set listData( value:BaseListData ) : void
            {
                _listData = value;
            }
            ...
```

Sizing an item renderer

- The explicit width of an itemRender is always the size of its container (or the column size in a datagrid)
- The height of an itemRender is always rowHeight unless variableRowHeight = true (false by default)
- By default, the width of a List is 200 pixels and the number of rows displayed is 7.
- Reverse this for horizontal lists

Virtualization / Renderer Recycling

Renderers as Editors

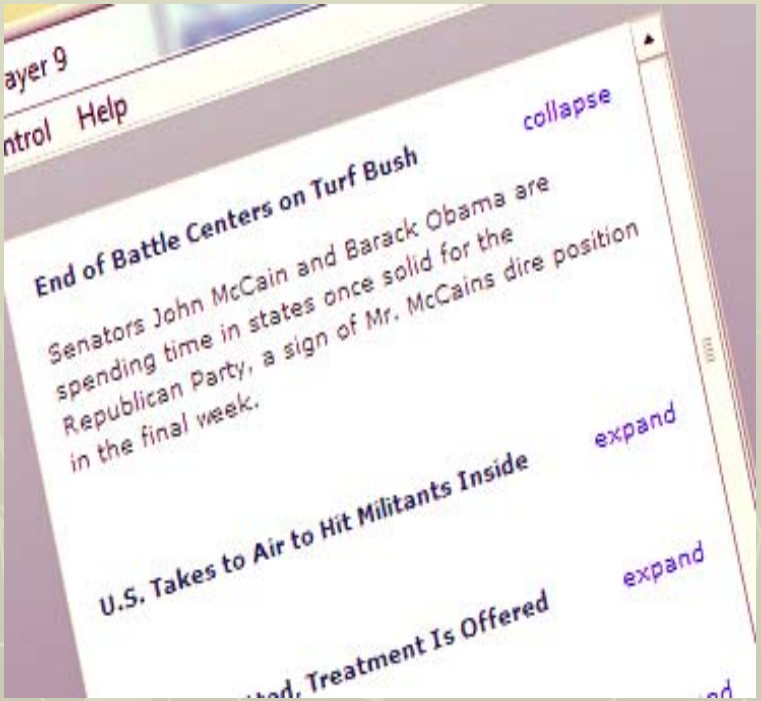
- Setting a custom itemRenderer like a CheckBox or DateField does not mean your data is saved.
- List component should be editable.
- Set rendererIsEditor=true.
- Set editorDataField if the default property of your renderer is not “text”.

Example: Slider as a Renderer and Editor

- Slider does not implement IDataRenderer or IDropInListItemRenderer.
- Make a custom Slider that implements IDataRenderer, IListItemRenderer and IDropInListItemRenderer.
- Set editable=true, itemRenderer="MySlider", rendererIsEditor="true" editorDataField="value"

```
<mx:DataGridColumn
    dataField="rating"
    headerText="Rating (1 to 10)"
    itemRenderer="MySlider"
    rendererIsEditor="true"
    editorDataField="value">
```

Using States in ItemRenders



- Remember row recycling. States are not saved in an itemRenderer resulting in unexpected behavior when you scroll.
- The State should be associated with the data of your itemRenderer.
- Reset states in your set data() methods.

Resetting the state in the set data function.

```
override public function set data( value:Object ) : void
{
    // setting the data property on the super class is required
    // for any override of the set data function.
    super.data = value;

    if(data.expanded)
        currentState="expandedState";
    else
        currentState="collapsedState";
}
```

Performance

- We are smart about creating itemRenderers, but there are still lots of item renderers on screen in a 20x20 datagrid.
- Item renderers need to be performant because there are so many of them.

Performance: Tips for Speed

- Layout's often expensive and unnecessary for item renderers
  - Instead, base your renderers off of UIComponent and layout the items explicitly yourself
- Developing your item renderers in MXML can be more expensive. For example, you probably don't need to use data-binding.
  - Instead just use ActionScript – you know when the data is changing (data setter gets called), so there's really no need for binding

ItemRenderers in Gumbo

- Still a lot of details TBD
- No virtualization support today
- No IDropInListItemRenderer support yet (no labelField, labelFunction)
- Not just itemRenderer property but itemRendererFunction as well
- DataGroups support both Data Items and Visual Elements (perhaps won't support Visual Element anymore, though)

Resources

- Ryan's Blog:

http://frishy.blogspot.com/

- Joan's Blog:

http://butterfliesandbugs.wordpress.com