UNIVERSAL MIND
enterprise RIA

Custom Components in a Enterprise RIA Environment

Douglas Knudsen

Agenda

First we will cover the basics then go over some code. Well, a bunch of code. The intent of the examples is to show custom components that the developer may actually use in a enterprise business application such as may exist on a Intranet or Customer Portal or some such animal.

- What are they?
- Why do we need them?
- Where do they live?
- How do we make use of them?
- What to think about?
- Code!

Agenda

Again we will cover the basics then go over a whole bunch of code. The intent of the examples is to display use of item renderers in a enterprise business type situation such as may exist in a Intranet application or Customer Portal.

- What are they?
- Why do we need them?
- How do they work?
- Are all renderers created equal?
- Where do they live?
- How do we make use of them?
- Code!

UNIVERSAL MIND

Custom Components

UNIVERSAL MIND

Custom Components

What are they?

UNIVERSAL MIND

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself
- Can be AS or MXML based

UNIVERSAL MIND

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself
- Can be AS or MXML based
- Can accept values (injection)

UNIVERSAL MIND

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself
- Can be AS or MXML based
- Can accept values (injection)
- Can expose instance variables

UNIVERSAL MIND

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself
- Can be AS or MXML based
- Can accept values (injection)
- Can expose instance variables
- Can broadcast events, standard or custom

UNIVERSAL MIND

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself
- Can be AS or MXML based
- Can accept values (injection)
- Can expose instance variables
- Can broadcast events, standard or custom
- Can be composite, that is contain other components

UNIVERSAL MIND

Custom Components

What are they?

- Just a class typically extending some descendent of UIComponent or UIComponent itself
- Can be AS or MXML based
- Can accept values (injection)
- Can expose instance variables
- Can broadcast events, standard or custom
- Can be composite, that is contain other components
- Guess what? The Flex SDK is really a bunch of components!

UNIVERSAL MIND

Custom Components

UNIVERSAL MIND

Custom Components

Why do we need them?

UNIVERSAL MIND

Custom Components

Why do we need them?

- Modularize code

UNIVERSAL MIND

Custom Components

Why do we need them?

- Modularize code
- Add or extend functionality not contained in the Flex SDK

UNIVERSAL MIND

Custom Components

Why do we need them?

- Modularize code
- Add or extend functionality not contained in the Flex SDK
- Encapsulate functionality

Custom Components

Why do we need them?

- Modularize code
- Add or extend functionality not contained in the Flex SDK
- Encapsulate functionality
- Satisfy some business requirement

UNIVERSAL MIND

Custom Components

Why do we need them?

- Modularize code
- Add or extend functionality not contained in the Flex SDK
- Encapsulate functionality
- Satisfy some business requirement
- UX baby!

Custom Components

Why do we need them?

- Modularize code
- Add or extend functionality not contained in the Flex SDK
- Encapsulate functionality
- Satisfy some business requirement
- UX baby!
- Amaze your mum

UNIVERSAL MIND

Custom Components

UNIVERSAL MIND

Custom Components

Where do they live?

UNIVERSAL MIND

Custom Components

Where do they live?

- Simply put we place them in the class path

UNIVERSAL MIND

Custom Components

Where do they live?

- Simply put we place them in the class path
- So, somewhere under the flex source directories set in the project settings

UNIVERSAL MIND

Custom Components

Where do they live?

- Simply put we place them in the class path
- So, somewhere under the flex source directories set in the project settings
- Can also hang out in a library, aka swc

UNIVERSAL MIND

Custom Components

Where do they live?

- Simply put we place them in the class path
- So, somewhere under the flex source directories set in the project settings
- Can also hang out in a library, aka swc
- In the case of a ItemRenderer, they can appear inline

UNIVERSAL MIND

Custom Components

Where do they live?

- Simply put we place them in the class path
- So, somewhere under the flex source directories set in the project settings
- Can also hang out in a library, aka swc
- In the case of a ItemRenderer, they can appear inline
- They can even appear in MXML code <mx:Component….

UNIVERSAL MIND

Custom Components

UNIVERSAL MIND

Custom Components

How do we make use of them?

UNIVERSAL MIND

Custom Components

How do we make use of them?

- Use them like any other class in Flex!

UNIVERSAL MIND

Custom Components

How do we make use of them?

- Use them like any other class in Flex!
- Don’t forget, those MXML tags you use are just class instances

UNIVERSAL MIND

Custom Components

How do we make use of them?

- Use them like any other class in Flex!
- Don’t forget, those MXML tags you use are just class instances
- In MXML use a MXML name space: xmlns:comps="path.to.my.components”

UNIVERSAL MIND

Custom Components

How do we make use of them?

- Use them like any other class in Flex!
- Don’t forget, those MXML tags you use are just class instances
- In MXML use a MXML name space: xmlns:comps="path.to.my.components”
- In AS use import statement: import path.to.my.components

UNIVERSAL MIND

Custom Components

UNIVERSAL MIND

Custom Components

What to Think About?

UNIVERSAL MIND

Custom Components

What to Think About?

- Encapsulate stuff that needs to be, Check your privates private!

UNIVERSAL MIND

Custom Components

What to Think About?

- Encapsulate stuff that needs to be, Check your privates private!
- Think about a buzzword: Loose Coupling, do not use notations like parent.parent.foo.goo = 42

UNIVERSAL MIND

Custom Components

What to Think About?

- Encapsulate stuff that needs to be, Check your privates private!
- Think about a buzzword: Loose Coupling, do not use notations like parent.parent.foo.goo = 42
- Use the Events Luke! If a component needs to communicate to the outside world, use events, custom or built-ins

UNIVERSAL MIND

Custom Components

What to Think About?

- Encapsulate stuff that needs to be, Check your privates private!
- Think about a buzzword: Loose Coupling, do not use notations like parent.parent.foo.goo = 42
- Use the Events Luke! If a component needs to communicate to the outside world, use events, custom or built-ins
- AS Components that you want to use via MXML require zero argument constructors

UNIVERSAL MIND

Custom Components

What to Think About?

- Encapsulate stuff that needs to be, Check your privates private!
- Think about a buzzword: Loose Coupling, do not use notations like parent.parent.foo.goo = 42
- Use the Events Luke! If a component needs to communicate to the outside world, use events, custom or built-ins
- AS Components that you want to use via MXML require zero argument constructors
- Make use of the component life cycle, more on this later time permitting

UNIVERSAL MIND

Custom Components

Show me the code!

UNIVERSAL MIND

Custom Components

No, really, show me the code!

UNIVERSAL MIND

Item Renderers

UNIVERSAL MIND

Item Renderers

What are they?

UNIVERSAL MIND

Item Renderers

What are they?

- A specific set of custom components for list based controls such as the DataGrid, List, TileList, etc…

Item Renderers

What are they?

- A specific set of custom components for list based controls such as the DataGrid, List, TileList, etc…
- Can be AS or MXML based

Item Renderers

What are they?

- A specific set of custom components for list based controls such as the DataGrid, List, TileList, etc…
- Can be AS or MXML based
- Can accept values depending on implementation

Item Renderers

What are they?

- A specific set of custom components for list based controls such as the DataGrid, List, TileList, etc…
- Can be AS or MXML based
- Can accept values depending on implementation
- Can broadcast events (What can't in Flex, eh?)

UNIVERSAL MIND

Item Renderers

What are they?

- A specific set of custom components for list based controls such as the DataGrid, List, TileList, etc…
- Can be AS or MXML based
- Can accept values depending on implementation
- Can broadcast events (What can't in Flex, eh?)
- In abstract terms, they can be thought of as a new view to your model in a list based control

UNIVERSAL MIND

Item Renderers

UNIVERSAL MIND

Item Renderers

Why do we need them?

- Extend base functionality of list controls

  eg: use CheckBox in a DataGrid column.

UNIVERSAL MIND

Item Renderers

Why do we need them?

- Extend base functionality of list controls

  eg: use CheckBox in a DataGrid column.
- Combine controls into one ‘item’

UNIVERSAL MIND

Item Renderers

Why do we need them?

- Extend base functionality of list controls

  eg: use CheckBox in a DataGrid column.
- Combine controls into one ‘item’
- Add programmatic functionality to a ‘item’
  eg: colorize text based on value

UNIVERSAL MIND

Item Renderers

Why do we need them?

- Extend base functionality of list controls

  eg: use CheckBox in a DataGrid column.
- Combine controls into one ‘item’
- Add programmatic functionality to a ‘item’
  eg: colorize text based on value
- Can be used to add editable functionality, here they are known as ItemEditors

UNIVERSAL MIND

Item Renderers

Why do we need them?

- Extend base functionality of list controls

  eg: use CheckBox in a DataGrid column.
- Combine controls into one ‘item’
- Add programmatic functionality to a ‘item’
  eg: colorize text based on value
- Can be used to add editable functionality, here they are known as ItemEditors
- Defy gravity and again most importantly, amaze your mum!

UNIVERSAL MIND

Item Renderers

How do they work?

- The list based control has to pass data in its model to the renderer
- The Flex SDK uses the *data* property of the item renderer to pass the information
- Thus *IDataRenderer, IDropInListItemRenderer,* and *IListItemRenderer* exist, they all describe how this interaction takes place
- Each control sets data in a slightly different way, but basically the same.
- The DataGrid control stands out in that the *data* element is set to the data element representing that row. So, yes, 10 columns means 10 copies of that VO. (according to the docs)
- Good to know that the control only instantiates what it needs here and re-uses the instances for data not currently displayed, thus be wary of using creationComplete type approaches, make use of set data() instead or listen for dataChange event.

Item Renderers

Are all renderers created equal?

Default: Each of the list controls has a default renderer used when no custom one is identified. What this default is depends on the control. The DataGrid for example has a default renderer extending UITextField, the DataGridItemRender class. This implements IDropInListItemRenderer too and thus has access to BaseListData which is how DataGridItemRender gets to the value injected into it via the dataField property.

Item Renderers

Are all renderers created equal?

Drop-in: simple form using a component in the Flex SDK such as Image. Any component implementing *IDropInListItemRenderer* will work here. These work very specifically in that they expect to use a specific property in the data. Therefore they can easily work on multiple columns say in a DataGrid. Consider the Image control. Called Drop-in for a reason, simply drop them in and they work. Can use these as ItemEditors too, set the editorDataField correctly! Can also create custom drop-ins.

UNIVERSAL MIND

Item Renderers

Are all renderers created equal?

Inline: More powerful than drop-in renderers allowing the developer to compose functionality as well as configure controls. Define 'inline' with the MXML for the list component. Not reusable, well not reusable outside the file where implemented. Can add a Script block too. Inline renderers support the *data* property which is bindable of course. Be wary of scope here, variables inside <mx:Component > are localized. Further, variables outside are not directly available in scope, rather you can access them via the outerDocument property.

UNIVERSAL MIND

Item Renderers

Are all renderers created equal?

Component: Use any custom component as a renderer. Allows fully customizable renderers. Reusable! Most of the requirements for inline renderers apply here, so implement *IDataRenderer* to gain access to *data* injected into the renderer.

UNIVERSAL MIND

Item Renderers

How do we use them?

Drop-in: list which component to use in the itemRenderer or itemEditor attribute

- <mx:DataGridColumn headerText="Active" itemRenderer="CheckBox" dataField="selected" />

UNIVERSAL MIND

Item Renderers

How do we use them?

Inline: Place “inline” with MXML code

```
<mx:DataGridColumn headerText=“Image Name” >
  <mx:itemRenderer>
    <mx:Component>
      <mx:VBox>
        <mx:Image source=“{data.lePath}” />
      </mx:VBox>
    </mx:Component>
</mx:DataGrid>
```

*You can add a Script block in there too! Put it inside the root, this

UNIVERSAL MIND

Item Renderers

How do we use them?

Component: Just like the drop-in type, list which component to use in the itemRenderer attribute

- <mx:DataGridColumn headerText="Active" itemRenderer="path.to.my.renderer" />

UNIVERSAL MIND

Item Renderers

Exposing properties to the control

- Use ClassFactory and AS to set renderer and properties:

```
var renderer:ClassFactory = new
  ClassFactory( GPSIconRenderer );
renderer.properties = { showField: "gps" };
delColumn.itemRenderer = renderer;
```

Show me some code!

UNIVERSAL MIND

Where to Next?

Flex Component Life Cycle

- Learn ‘what happens when’ in the life of a component
- Exert fine control over your component
- Learn which methods to override to perform certain customizations
    - Eg: `createChildren()` or `layoutChrome()`

UNIVERSAL MIND

HIGH IMPACT CONSULTING