Tree

This article describes how the FlexJS Tree component was created.

A tree is a form of a list, which is the basis for the Tree component - the FlexJS List component. The key difference is that a tree has a more complex data structure. Rather than have just an array of items to display, a tree has a nested structure with some items having children.

As shown below, the Tree is a derivative of the List, including specialized Tree beads which are themselves, derivations of List beads.

HierarchicalData

The Tree component uses the HierarchicalData class (or some derivative) for its dataProvider. The HiearchicalData class is a nesting of Objects, each of which has uniform content in that there is a element of the object that identifies the children. For example:

{title: "My Music", children:[...]}

where the array of children are similar objects, called "nodes". A node will either be a leaf (have no children) or branch (have a children item).

Another key piece of the Tree component is the itemRenderer which, unlike itemRenderers for the List, needs to know about the placement of the data, which it is about to display, within the HierarchicalData. This allows the Tree's itemRenderer to indent the content:

```
> My Books
V My Albums
   > The 1980s
   V The 1990s
      > Top 100
      > Blues
   > The 2000s
   > The 2010s
```

The Tree Strand

The centerpiece of the Tree component is the strand - the basic component structure of FlexJS. Since Tree is a List, Tree is a subclass of List with a couple of differences:

- Tree expects the dataProvider to be of class HierarchicalData
- Tree overrides the setter and getter for the dataProvider property and creates a FlattenedList (more on this below) which is the true dataProvider for the underlying List

In addition, the Tree comes with a style definition that replaces a couple of the beads used by List:

- The controller is a TreeSingleSelectionMouseController
- The dataProvider item renderer factory is DataItemRendererFactoryForHierarchicalData
- The itemRenderer is TreeItemRenderer

The remaining beads, such as the ListView bead and even the model (ArraySelectionModel) remain the same.

The FlattenedList

A List works well with an array of data, but a Tree works with structured data. To bridge the two, the Tree, internally, uses a org.apache.flex.collections.FlattenedList, which is a specialized version of ArrayList that uses a HierarchicalData object to glean from it the open nodes. For example, when the Tree first displays the collection, the root node is opened automatically and contains two items: "My Books" and "My Albums". When the "My Albums" node is selected, the FlattenedList is told to open that node. To do this, the FlattenedList gets the children of the node from the HierarchicalData object and adds those nodes to its own source array. While the HierarchicalData may hold hundreds of items, the FlattenedList just holds those items which are open (or leaf nodes when they become exposed by opening their parent nodes). When "My Albums" is selected, the two items in the FlattenedList become six items: the two top level nodes and the four children of "My Albums".

The List structure then works with what it knows - a one-dimensional array of data. Since the FlattenedList does not copy the original data in the HierarchicalData structure, the FlattenedList is reasonably economical.

The TreeSingleSelectionMouseController

The job of a controller is to handle events on behalf of the view and work with the data model. The TreeSingleSelectionMouseController does this by extending ListSingleSelectionMouseController and overriding the selection handler function. When a node has been selected, this controller intercepts that event and determines, based on the data held by the itemRenderer selected, which node to open or close. Since the model holds the FlattenedList, the controller just changes its state and then notifies the List parts that the dataProvider has been changed (due to the addition of open nodes or the removal of closed nodes in the FlattenedList).

The TreeItemRenderer

The TreeItemRenderer extends StringItemRenderer and changes the setter function for the data. The data being displayed is a node in the HierarchicalData and is also in the FlattenedList. But this information is not enough to determine where in the structure this data lies. For example, if the data is "Top 100", it lies on the third level.

Along with the actual data for the row being displayed, the List components supply an additional piece of information called, "listData". Normally this is empty, but for the Tree, listData is an instance of TreeItemData. This class contains the depth of the data within the tree, whether or not the node is currently open (showing its children), and whether or not the node even has any children. The TreeItemRenderer uses this additional information is provide a visual cue (open and close arrows) as well as indentation for the text of the node.

The DataItemRendererFactoryForHierarchicalData

The final piece, DataItemRendererForHierarchicalData, generates the tree itemRenderers based on the nodes in the FlattenedList (dataProvider in the Tree's model). The creation of the itemRenderers is similar to a normal list, except for the creation of the listData - TreeListData instances. As the itemRenderers are created, the factory looks at each node and determines its depth, whether or not is has children, and if it does have children, as they also open. When this completes, the other parts of the List structure can present the itemRenderers and they can format themselves accordingly.