TreeTableView API Discussion

The TreeTableView control is a new control targeted at JavaFX 8.0. The Jira issue for this feature request is at RT-17288, but this wiki page should cover the main points in summary.

New API

The following list enumerates all new classes currently being proposed for inclusion in JavaFX 8.0 for the TreeTableView. Note that it also includes classes related to cell spanning support which is extra and may not be included.

javafx.scene.control

- CellSpan
- ResizeFeaturesBase
- SpanModel
- TableColumnBase
- TableFocusModel
- TablePositionBase
- TableSelectionModel
- TreeSortMode
- TreeTableCell
- TreeTableColumn
- TreeTablePosition
- TreeTableRow
- TreeTableView

javafx.scene.control.cell

- CheckBoxTreeTableCell
- ChoiceBoxTreeTableCell
- ComboBoxTreeTableCell
- ProgressBarTreeTableCell
- TextFieldTreeTableCell
- TreeItemPropertyValueFactory

API Discussion

The TreeTableView API is intended to be very familiar to users of the TreeView and TableView controls. The intent is to reuse as many API, or API concepts, as possible in building the API for TreeTableView. To this end the main APIs used by TreeTableView are the TreeItem class (that is, exactly the same class as used by TreeView) and TreeTableColumn (which whilst a separate class from TableColumn is analogous to it, except it deals with TreeItem<T>, rather than objects of type T. Also, TreeTableColumn extends from a new TableColumnBase class - see below for more details). The TreeTableView class itself provides much of the same API as that already in TreeView and TableView, such as containing a list of columns, a list of 'sort columns' that specifies the sort order, a single root TreeItem, etc.

Because the TreeTableView data model is tree-like (in the same way as TreeView) there is a single root TreeItem, which has children TreeItems, and this recursively defines the tree structure of the TreeTableView. The data model is also used as the view model, in that reordering the children of a TreeItem will result in them being visually reordered also.

TreeTableColumn

The TreeTableColumn class is a new class introduced along with TreeTableView. Conceptually it is identical to the TableColumn class used by TableView. In fact, there is so much common API that a new TableColumnBase class has been introduced in the current implementation to extract it out (and to make the implementation simpler - see the implementation section below). The major difference between the two classes is that TreeTableColumn only deals with TreeItem<T>, whereas TableColumn, being the more general of the two classes, can work with any type for T. The similarities with TableColumn include the following:

- TreeTableColumn has a reference back to the TreeTableView it is associated with.
- TreeTableColumn supports nested columns.
- TreeTableColumn uses the same cell value factory / cell factory concept for data retrieval and rendering.

TableColumnBase currently extracts out functionality such as the column text, graphic, width, context menu, API for getting cell data, and many parts of the sorting API, among other things.

Discussion: TreeTableColumn

TreeTableColumn uses the same cellValueFactory concept as used in TableColumn. This means that to get a value out of a TreeItem for a given column, the following code is the worst-case scenario:

```
TreeTableColumn<Person, String> firstNameCol= new TreeTableColumn<Person, String>("First Name");
firstNameCol.setCellValueFactory(new Callback<CellDataFeatures<Person, String>, ObservableValue<Person>>() {
    @Override public ObservableValue<String> call(CellDataFeatures<Person, String> p) {
        // first getValue() returns the TreeItem out of the CellDataFeatures,
        // second getValue() returns the Person instances out of the TreeItem.
        return p.getValue().getValue().firstNameProperty();
    }
});
```

This is unfortunate, and what makes matters worse is that the commonly recommended approach of using PropertyValueFactory will not work as it is currently hard-coded to work with TableColumn, not TreeTableColumn. This suggests that perhaps a new class needs to be introduced such that the code above can be avoided. For example, here is the code a developer would write for a TableColumn:

```
firstNameCol.setCellValueFactory(new PropertyValueFactory<Person,String>("firstName"));
```

Because we can't reuse PropertyValueFactory, it may be necessary to introduce a new class (let's call it "TreeItemValueFactory"), such that developers can write code along the following line of code, rather than the first sample above:

```
firstNameCol.setCellValueFactory(new TreeItemValueFactory<Person,String>("firstName"));
```

Sorting

Sorting is discussed further in the UX specification for TreeTableView, but in general it introduces new concepts that are not borrowed from TreeView (but may be backported to TreeView if the need arises). Essentially, TreeTableColumn defines sortType (ascending or descending) and comparator properties, and these specify how a single column should be sorted, should it be used to sort the rows of the TreeTableView. So far this is very reminiscent of TableView, but where we diverge is in the introduction of a sortMode enumeration in TreeTableView, which at present defines two types of sort mode: "ALL_DESCENDENTS" and "ONLY_FIRST_LEVEL". The definition of this enum gives a little more insight into how these two modes differ:

```
public static enum SortMode {
/**
* Sort all nodes in the tree, but only sort nodes belonging to the
* same parent.
*/
ALL_DESCENDANTS,

/**
* Sort first level nodes only regardless of whether the root is
* actually being shown or not. In other words, this will always sort
* the children of the root node only.
*/
ONLY_FIRST_LEVEL;
}
```

The default sort mode is ALL_DESCENDENTS, as this tends to be the more common of the two.

Discussion: Sorting

There are a few points to discuss with respect to sorting, including:

1. Are there any other critical sort modes that need to be supported in the first release of TreeTableView?
2. Presently the SortMode enumeration is defined inside the TreeTableView class as a public static enum. If this enumeration is to ever be used by TreeView should we instead extract this out into its own enumeration class?

Row & Cell Factories

As with TableView, the TreeTableView allows for customisation of entire rows, or individual cells, inside of it. To customise an entire row (which is not an overly common use case), a developer would set the rowFactory property inside TreeTableView. The rowFactory property is a Callback that must return a TreeTableRow instance. This is entirely analogous to the TableView rowFactory returning a TableRow instance.

Similarly, if a developer wishes to customise a single cell in a TreeTableRow (which is the far more common use case), they should set a custom cellFactory on the relevant TreeTableColumn. This will customise all cells in that particular column, and once again is analogous to setting the cellFactory property on a TableColumn instance (except a TreeTableCell instance must be returned in place of a TableCell instance).