Handling Selections in Item Views

Concepts

The selection model used in the new item view classes offers many improvements over the selection model used in Qt 3. It provides a more general description of selections based on the facilities of the model/view architecture. Although the standard classes for manipulating selections are sufficient for the item views provided, the selection model allows you to create specialized selection models to suit the requirements for your own item models and views.

Information about the items selected in a view is stored in an instance of the QItemSelectionModel class. This maintains model indexes for items in a single model, and is independent of any views. Since there can be many views onto a model, it is possible to share selections between views, allowing applications to show multiple views in a consistent way.

Selections are made up of selection ranges. These efficiently maintain information about large selections of items by recording only the starting and ending model indexes for each range of selected items. Non-contiguous selections of items are constructed by using more than one selection range to describe the selection.

Selections are applied to a collection of model indexes held by a selection model. The most recent selection of items applied is known as the current selection. The effects of this selection can be modified even after its application through the use of certain types of selection commands. These are discussed later in this section.

When manipulating selections, it is often helpful to think of QItemSelectionModel as a record of the selection state of all the items in an item model. Once a selection model is set up, collections of items can be selected, deselected, or their selection states can be toggled without the need to know which items are already selected. The indexes of all selected items can be retrieved at any time, and other components can be informed of changes to the selection model via the signals and slots mechanism.

Using a Selection Model

The standard view classes provide default selection models that can be used in most applications. A selection model belonging to one view can be obtained using the view's selectionModel() function, and shared between many views with setSelectionModel(), so the construction of new selection models is generally not required.

A selection is created by specifying a model, and a pair of model indexes to a QItemSelection. This uses the indexes to refer to items in the given model, and interprets them as the top-left and bottom-right items in a block of selected items. To apply the selection to items in a model requires the selection to be submitted to a selection model; this can be achieved in a number of ways, each having a different effect on the selections already present in the selection model.

Selecting Items

To demonstrate some of the principal features of selections, we construct an instance of a custom table model with 32 items in total, and open a table view onto its data:

```
        TableModel model = new TableModel(8, 4, app);


        QTableView table = new QTableView();
        table.setModel(model);

        QItemSelectionModel selectionModel = table.selectionModel();
```

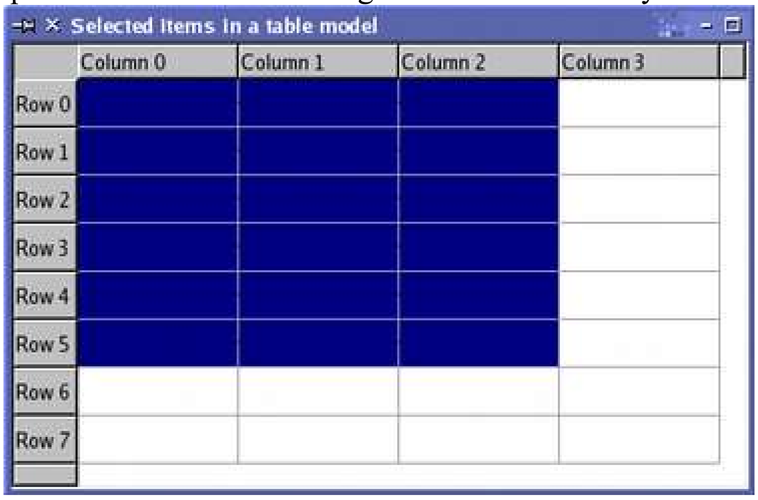
The table view's default selection model is retrieved for later use. We do not modify any items in the model, but instead select a few items that the view will display at the top-left of the table. To do this, we need to retrieve the model indexes corresponding to the top-left and bottom-right items in the region to be selected:

```
        QModelIndex topLeft;
        QModelIndex bottomRight;

        topLeft = model.index(0, 0, null);
        bottomRight = model.index(5, 2, null);
```

To select these items in the model, and see the corresponding change in the table view, we need to construct a selection object then apply it to the selection model:

```
        QItemSelection selection = new QItemSelection(topLeft, bottomRight);
        selectionModel.select(selection, QItemSelectionModel.SelectionFlag.Select);
```

The selection is applied to the selection model using a command defined by a combination of selection flags. In this case, the flags used cause the items recorded in the selection object to be included in the selection model, regardless of their previous state. The resulting selection is shown by the view.



The selection of items can be modified using various operations that are defined by the selection flags. The selection that results from these operations may have a complex structure, but will be represented efficiently by the selection model. The use of different selection flags to manipulate the selected items is described when we examine how to update a selection.

Reading the Selection State

The model indexes stored in the selection model can be read using the selectedIndexes() function. This returns an unsorted list of model indexes that we can iterate over as long as we know which model they are for:

```
        List<QModelIndex> indexes = selectionModel.selectedIndexes();

        for (QModelIndex index : indexes) {
            String text = "(" + index.row() + ", " + index.column() + ")";
            model.setData(index, text);
        }
```

The above code uses Qt's convenient foreach keyword to iterate over, and modify, the items corresponding to the indexes returned by the selection model.

The selection model emits signals to indicate changes in the selection. These notify other components about changes to both the selection as a whole and the currently focused item in the item model. We can connect the selectionChanged() signal to a slot, and examine the items in the model that are selected or deselected when the selection changes. The slot is called with two QItemSelection objects: one contains a list of indexes that correspond to newly selected items; the other contains indexes that correspond to newly deselected items.

In the following code, we provide a slot that receives the selectionChanged() signal, fills in the selected items with a string, and clears the contents of the deselected items.

```
    public void updateSelection(QItemSelection selected,
        QItemSelection deselected)
    {
        List<QModelIndex> items = selected.indexes();

        for (QModelIndex index : items) {
            String text = "(" + index.row() + ", " + index.column() +")";
            model.setData(index, text);

        }


        items = deselected.indexes();

        for (QModelIndex index : items)
            model.setData(index, "");
    }
```

We can keep track of the currently focused item by connecting the currentChanged() signal to a slot that is called with two model indexes. These correspond to the previously focused item, and the currently focused item.

In the following code, we provide a slot that receives the currentChanged() signal, and uses the information provided to update the status bar of a QMainWindow:

```
    public void changeCurrent(QModelIndex current, QModelIndex previous)
    {
        statusBar().showMessage(
            "Moved from (" + previous.row() + ", " +previous.column() + " to "
            + "(" + current.row() + ", " +current.column() +")");
    }
```

Monitoring selections made by the user is straightforward with these signals, but we can also update the selection model directly.

Updating a Selection

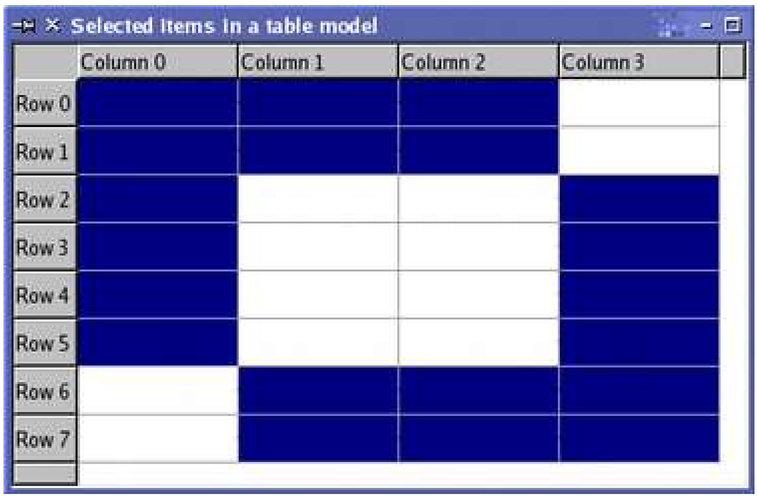
Selection commands are provided by a combination of selection flags, defined by QItemSelectionModel::SelectionFlag. Each selection flag tells the selection model how to update its internal record of selected items when either of the select() functions are called. The most commonly used flag is the

Select flag which instructs the selection model to record the specified items as being selected. The Toggle flag causes the selection model to invert the state of the specified items, selecting any deselected items given, and deselecting any currently selected items. The Deselect flag deselects all the specified items.

Individual items in the selection model are updated by creating a selection of items, and applying them to the selection model. In the following code, we apply a second selection of items to the table model shown above, using the Toggle command to invert the selection state of the items given.

```
        QItemSelection toggleSelection = new QItemSelection();

        topLeft = model.index(2, 1, null);
        bottomRight = model.index(7, 3, null);
        toggleSelection.select(topLeft, bottomRight);

        selectionModel.select(toggleSelection, QItemSelectionModel.SelectionFlag.Toggle);
```

The results of this operation are displayed in the table view, providing a convenient way of visualizing what we have achieved:

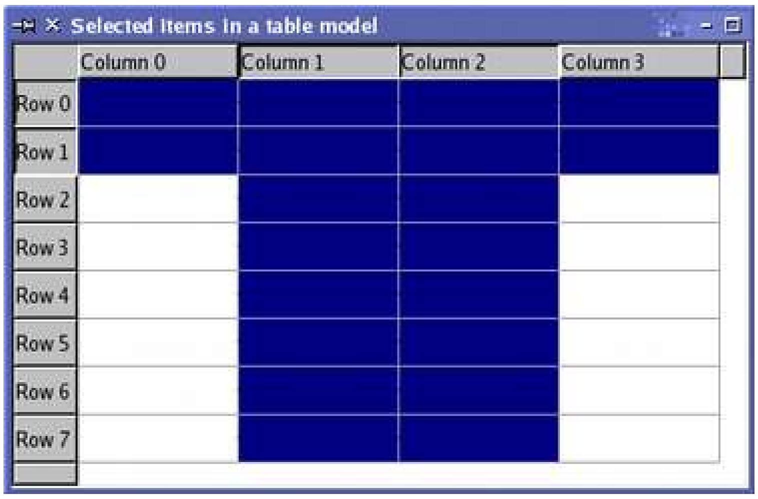


	Column 0	Column 1	Column 2	Column 3
Row 0	■	■	■	
Row 1	■	■	■	
Row 2	■			■
Row 3	■			■
Row 4	■			■
Row 5	■			■
Row 6		■	■	■
Row 7		■	■	■

By default, the selection commands only operate on the individual items specified by the model indexes. However, the flag used to describe the selection command can be combined with additional flags to change entire rows and columns. For example if you call select() with only one index, but with a command that is a combination of Select and Rows, the entire row containing the item referred to will be selected. The following code demonstrates the use of the Rows and Columns flags:

```
        QItemSelection columnSelection = new QItemSelection();

        topLeft = model.index(0, 1, null);
        bottomRight = model.index(0, 2, null);

        columnSelection.select(topLeft, bottomRight);

        selectionModel.select(columnSelection,
                              QItemSelectionModel.SelectionFlag.Select,
                              QItemSelectionModel.SelectionFlag.Columns);

        QItemSelection rowSelection = new QItemSelection();

        topLeft = model.index(0, 0, null);
        bottomRight = model.index(1, 0, null);

        rowSelection.select(topLeft, bottomRight);

        selectionModel.select(rowSelection,
            QItemSelectionModel.SelectionFlag.Select, QItemSelectionModel.SelectionFlag.Rows);
```

Although only four indexes are supplied to the selection model, the use of the Columns and Rows selection flags means that two columns and two rows are selected. The following image shows the result of these two selections:



The commands performed on the example model have all involved accumulating a selection of items in the model. It is also possible to clear the selection, or to replace the current selection with a new one.

To replace the current selection with a new selection, combine the other selection flags with the Current flag. A command using this flag instructs the selection model to replace its current collection of model indexes with those specified in a call to select(). To clear all selections before you start adding new ones, combine the other selection flags with the Clear flag. This has the effect of resetting the selection model's collection of model indexes.

Selecting All Items in a Model

To select all items in a model, it is necessary to create a selection for each level of the model that covers all items in that level. We do this by retrieving the indexes corresponding to the top-left and bottom-right items with a given parent index:

```
        QModelIndex topLeft = model.index(0, 0, parent);
        QModelIndex bottomRight = model.index(model.rowCount(parent) - 1,
            model.columnCount(parent) - 1, parent);
```

A selection is constructed with these indexes and the model. The corresponding items are then selected in the selection model:

```
        QItemSelection selection = new QItemSelection(topLeft, bottomRight);
        selectionModel.select(selection, QItemSelectionModel.SelectionFlag.Select);
```

This needs to be performed for all levels in the model. For top-level items, we would define the parent index in the usual way:

```
        QModelIndex parent = null;
```

For hierarchical models, the hasChildren() function is used to determine whether any given item is the parent of another level of items.