Grid to Order

By David Wallwork

One of the most anticipated enhancements in BBj 6.0 is the addition of sorting capability to BBjStandardGrid. Within this capability are two very versatile features; the program can initiate an actual sort, or the developer can configure the grid to be user sortable. This article provides specific instructions for using these new features.

Use the following call to perform a sort on a grid's column programmatically:

```
grid!.sortByColumn(int column, int sortOrder)
```

Where `sortOrder` may be

1	indicating ascending sort order
0	indicating non-sorted
-1	indicating descending sort order

The program can also configure the grid to be user sortable by calling:

```
    grid!.setColumnUserSortable(int column, boolean sortable)
or  grid!.setAllColumnsUserSortable(boolean sortable)
```

If a grid column is user sortable, then the grid sorts when the user clicks in the column header of the desired column. Subsequent clicks in the column header toggles between sorting the grid on that column in descending and ascending order.

The program can query the grid to discover whether a specified column is user sortable by calling:

```
grid!.isColumnUserSortable(int column)
```

One can make the grid revert to its original row order by calling:

```
grid!.unsort()
```

Developers can also sort a grid by multiple columns. The classic example is a grid in which one column contains last names and another column contains first names. Sorting the grid by first name, followed by a sort on last name, results in a telephone book appearance that groups last names together with first names sorted within that group. It is also common to sort the grid by one column and then sort the original (non-sorted) data by a different column. The program can control how the grid behaves by calling:

```
    grid!.setSortByMultipleColumns(1)  to obtain 'telephone book' behavior
or  grid!.setSortByMultipleColumns(0)  to obtain simple sort of original data
```

The program can discover the current sorting method by calling:

```
grid!.getSortByMultipleColumns()
```

All methods mentioned so far are quite clear and execute just as you would expect. However, there are three additional and less obvious methods that complete the sorting API and require a more detailed explanation. These methods are:

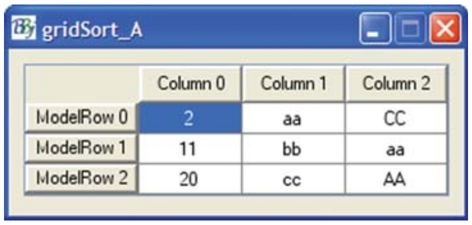
```
grid!.resort();
grid!.getRowModelIndexFromViewIndex(int p_index)
grid!.getRowViewIndexFromModelIndex(int p_index)
```

After sorting the grid, the programmer may modify the content of a cell, or if the cell is editable, the user may modify the cell, but such a change will not modify the row order of the grid. The grid remembers all sorting commands that it received whether the commands originated from the program or from a user's click in the row header. The program can cause the grid to re-sort itself using the current data by calling:

```
grid!.resort()
```

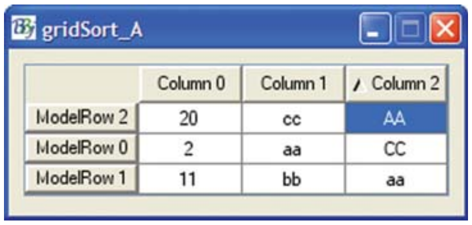
When sorting the grid, the location of a given cell may change, but the coordinates that reference that cell do not change. For example, run **`gridSort_1.bbj`** and notice the unsorted grid in **Figure 1**. The program addresses the cell containing **`AA`** with the coordinates (2,2). If the user clicks in the header of Column 2 the grid appears as shown in **Figure 2**, but the program still addresses the cell containing **`AA`** with the coordinates (2,2), even though it appears to the user as coordinates (0,2).

continued...



	Column 0	Column 1	Column 2
ModelRow 0	2	aa	CC
ModelRow 1	11	bb	aa
ModelRow 2	20	cc	AA

Figure 1. Unsorted grid



	Column 0	Column 1	Column 2
ModelRow 2	20	cc	AA
ModelRow 0	2	aa	CC
ModelRow 1	11	bb	aa

Figure 2. Sort on Column 2

David Wallwork
Senior Software Engineer

The cell actually has two coordinate positions; model coordinates (those seen by the program) and view coordinates (those seen by the user). The view coordinates are not normally of interest to the program, but if the program does need to translate between these two coordinate systems, it can use the following methods:

```
grid!.getRowModelIndexFromViewIndex(int p_index)
grid!.getRowViewIndexFromModelIndex(int p_index)
```

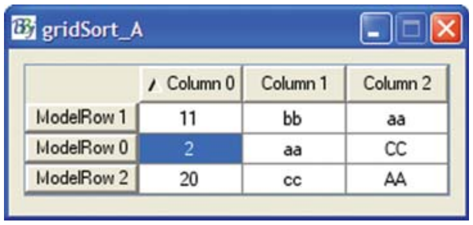
Using the sorted grid in **Figure 2**,

```
grid!.getRowViewIndexFromModelIndex(2)
```
returns the value 0

```
grid!.getRowModelIndexFromViewIndex(0)
```
returns the value 2

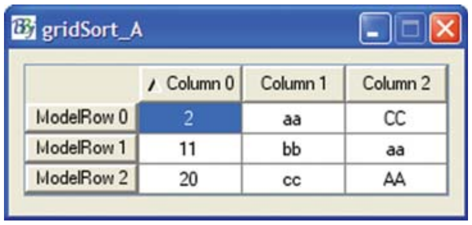
Numeric vs. Lexical Sorting

If you sort the grid in Figure 2 on Column 0, the resulting grid appears in **Figure 3**.



	Column 0	Column 1	Column 2
ModelRow 1	11	bb	aa
ModelRow 0	2	aa	CC
ModelRow 2	20	cc	AA

Figure 3. Text sort on Column 0



	Column 0	Column 1	Column 2
ModelRow 0	2	aa	CC
ModelRow 1	11	bb	aa
ModelRow 2	20	cc	AA

Figure 4. Numeric sort on Column 0

Notice that the cell containing `2` appears between the cell containing `11` and the cell containing `20`. While this looks incorrect, this sort actually ordered the values correctly. The cells in Column 0 are `GRID_STYLE_INPUTE` type, which results in a lexical sort. To sort numerically change the cell type in Column 0 to `GRID_STYLE_INPUTN` as shown below:

```
grid!.setColumnStyle(0, grid!.GRID_STYLE_INPUTN)
grid!.resort()
```

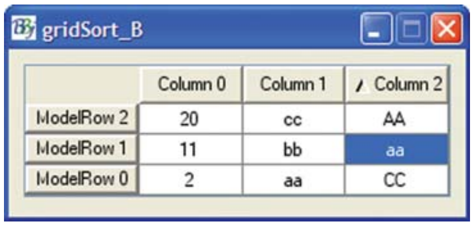
Now the column appears in **Figure 4** in a numeric sort order and the cell containing a `2` lies before the cell containing `11`.

If all the cells in a column are numeric, the grid will sort the column in numeric order. If all the cells are non-numeric, the grid will sort the column in lexical order. But if the column contains a mixture of styles, the grid does not know how to sort the column. A call to `sortByColumn()` for a column that contains mixed styles will throw an !ERR=17. Furthermore, clicking on a column's header with mixed styles will not affect the row order.

Custom Sort Orders

All this is very useful, but it may not be what you need in a particular situation. For example, refer back to **Figure 2** and see that the cell containing `aa` follows the cell containing `CC`. Why? Because 'a' follows 'Z' in the ASCII ordering. When requiring `aa` to appear before `CC` simply write a program that sorts the grid in this manner.

Figure 5 shows the output of the sample program `gridSort_2.bbj` which uses a custom sort order. The values that determine how to sort the grid are stored in a 'hidden' column. Column 2 defaults to non-user sortable, and when the user clicks in Column 2's header, the program sorts by the hidden column in order to provide the sort order. This technique requires some extra work but it allows developers to define any sort order the program requires.



	Column 0	Column 1	Column 2
ModelRow 2	20	cc	AA
ModelRow 1	11	bb	aa
ModelRow 0	2	aa	CC

Figure 5. Custom sort

Performance

All sorting runs on the client side, regardless of which new methods the developer used. This means that there is no need to send the data to the server for sorting, and then send it back to the client for displaying. The result is a very fast and efficient sort implementation.

Summary

With these new enhancements to BBjStandardGrid, BASIS provides significant new language functionality that will boost developer productivity and improve the performance of the resultant application.

Find the sample programs referenced in this article online at www.basis.com/advantage/mag-v9n2/grid.zip.