Leap From 'Cut and Paste' to 'Drag and Drop'

By Jim Douglas

One of the hallmarks of event-driven GUI applications is that the user is in control. The developer sets up an environment consisting of various windows and GUI objects and the user interacts with the environment at will, clicking a button here, typing a bit of text there, and so on. Drag and drop gives the user even more freedom to manipulate the GUI environment directly, in a natural and intuitive way, selecting data from one GUI object and dragging it to another object.

BBj® 7.0 supports two versions of drag and drop. The basic version requires no special programming; it is enabled automatically for all text-based controls. The advanced version is more flexible, but involves a bit of programming. This article will look at the basic (automatic) version first.

Automatic Drag and Drop

Most BBj 7.0 GUI controls support a basic form of drag and drop that can copy chunks of text from one BBj GUI control to another, or between a BBj GUI control and an external application, like a word processor. This version of drag and drop is similar to the standard Cut, Copy, and Paste functions supported by all text controls. The following sample contains no special drag and drop code; it demonstrates that BBj 7.0 text controls inherently support drag and drop:

```
rem ' Default Drag-and-Drop

sysgui = unt
open (sysgui)"X0"

sysgui! = bbjapi().getSysGui()

title$ = "Drag-and-Drop as Cut-and-Paste"
window! = sysgui!.addWindow(200,200,300,130,title$,$00010003$)

editbox! = window!.addEditBox(101,10,10,280,30,"")
inpute! = window!.addInputE(102,10,50,280,30,"")
reset! = window!.addButton(1,200,90,90,30,"Reset")

gosub reset

CALLBACK(ON_BUTTON_PUSH,RESET,sysgui!.getContext(),reset!.getID())
CALLBACK(ON_CLOSE,APP_CLOSE,sysgui!.getContext())

PROCESS_EVENTS

APP_CLOSE:
RELEASE

RESET:
  editbox!.setText("The quick brown fox jumps over the lazy dog.")
  inpute!.setText("")
RETURN
```

The user can move text from one control to another, equivalent to cutting text from the source control and pasting it to the target control. The CUT drag action is indicated by the special cursor shown in **Figure 1.**

continued...

Jim Douglas
Software Engineer
Contractor

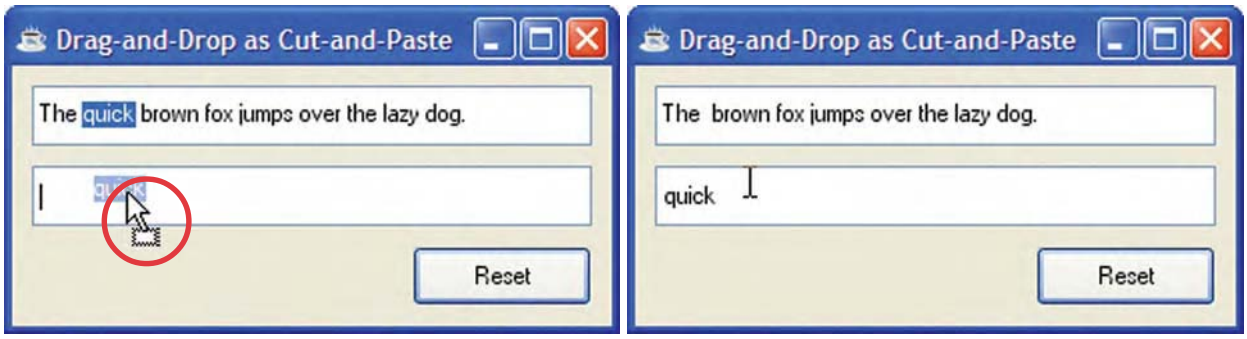

Figure 1. The cut cursor in action and the drop result

By pressing the CTRL key while dragging, the user can copy the text instead of moving it. The COPY drag action is indicated by the cursor shown in **Figure 2**.

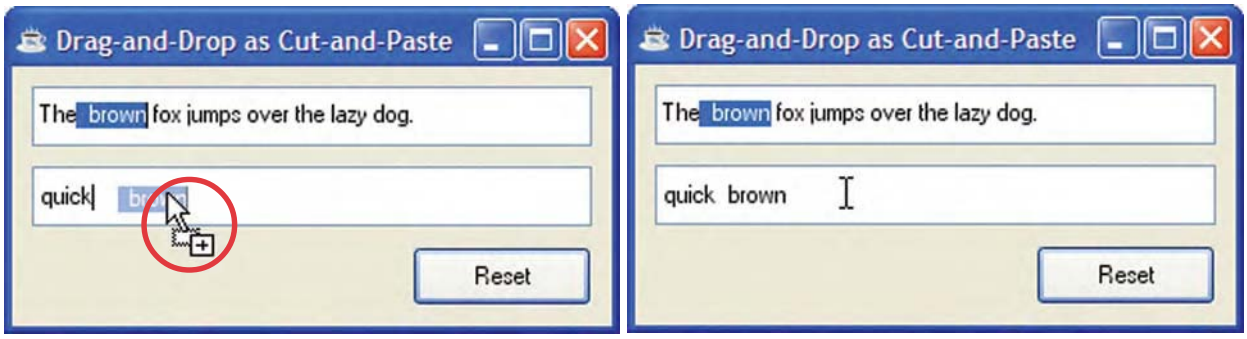

Figure 2. The copy drag cursor in action and the drop result

The Apple Human Interface Guidelines recommend using drag and drop as a supplementary user interface technique, not as the only way to accomplish a given task. In this sample, drag and drop gives the user a simple alternative to cut and paste. For a more complete sample, see **Default Drag and Drop Example** in the online documentation.

Programmable Drag and Drop

The basic version of drag and drop support handles the typical case, where a user wants to drag a chunk of text from one control to another. For more advanced or specialized applications, the developer will want to implement a more customized approach. This sample program implements a selection dialog with two lists: available options on the left and selected options on the right. The user can select one or more options from the **Available Options** list and click a button to transfer those options to the **Selected Options** list as shown in **Figure 3**.

continued...

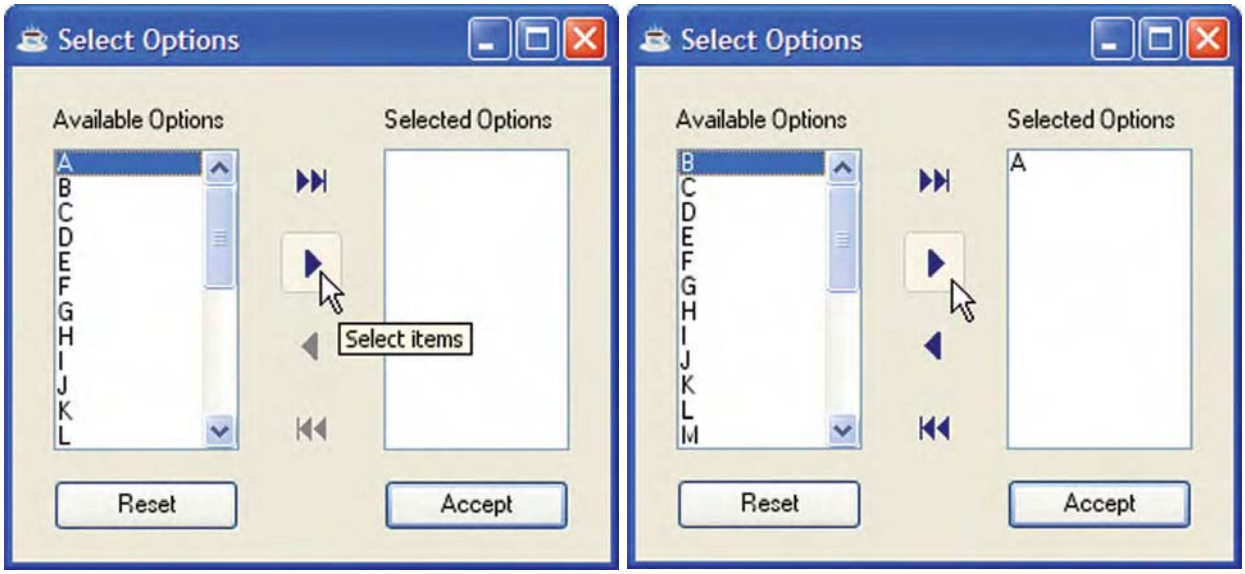

Figure 3. Selecting from a list with a click and its results

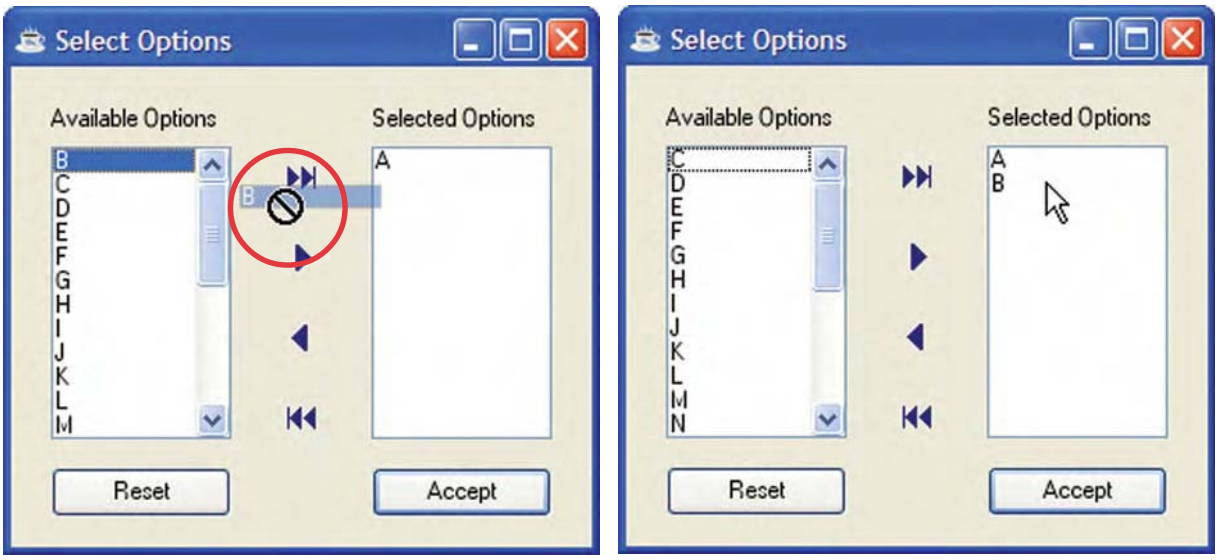

Figure 4. The move cursor indicating it is positioned over a "no drop zone" on its way to the final drop point

Drag and drop provides a more direct and intuitive way to perform the same task. The user clicks on one or more options, as in the first example, but instead of clicking a button to move the option from one list to the other, he or she simply drags and drops the chosen options to the **Selected Options** list. See **Figure 4**.

As the user drags data across a window, the drag cursor is continually updated to indicate whether data can be dropped on a particular control, and if so, how it would transfer. Notice in the first screen of **Figure 4** that as the mouse drags the data across the gray center, the cursor changes to a universal "no" symbol to indicate it is over a control that cannot accept the data. When the cursor appears as an arrow and a single rectangle as shown in the first screen of **Figure 5**, it indicates that dropping the data at that locaton will move the data rather than copy it.

Users can also use drag and drop to rearrange data within a single GUI control. For example, if the order of items in the selection list is significant, the developer can choose to implement drag and drop functionality to enable the user to reorder items within a list by dragging them. See **Figure 5**.

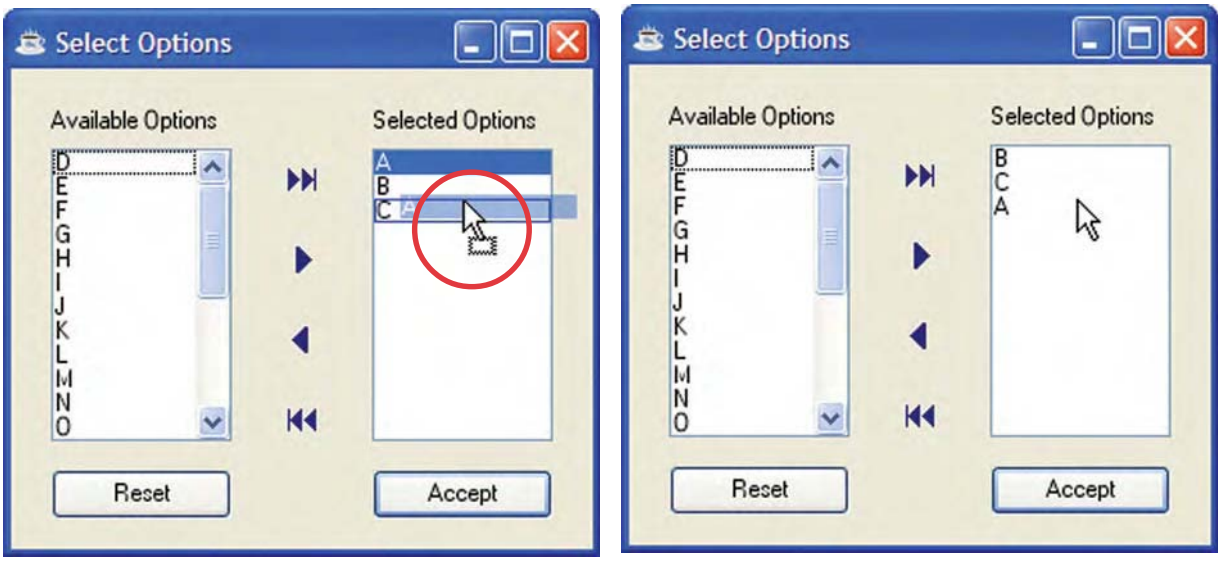

Figure 5. The move cursor in action, dragging and dropping to reorder the list

Drag and Drop Events

When a control registers for the BBjDropTargetDropEvent using the ON_DROP_TARGET_DROP callback, BBj switches from the default drag and drop behavior to the programmable behavior. In this sample program, registering these two callbacks causes BBj to fire BBjDropTargetDropEvents in response to data being dropped onto either of these controls:

```
CALLBACK(ON_DROP_TARGET_DROP,DROP_ITEMS,sysgui!.getContext(),available!.getID())
CALLBACK(ON_DROP_TARGET_DROP,DROP_ITEMS,sysgui!.getContext(),selected!.getID())
```

The corresponding event handler can be a bit overwhelming at first glance, but after breaking it down line-by-line as it appears in the next chart, the code is easily understood.

continued...

ON_DROP_TARGET_DROP EVENT HANDLER	COMMENTS
`Drop_Items:`	
`event! = sysgui!.getLastEvent()`	Get the BBjDropTargetDropEvent.
`dragSource! = event!.getDragSource()`	Get the drag source BBjControl.
`dropTarget! = event!.getControl()`	Get the drop target BBjControl.
`sel! = event!.getSelection()`	Get a BBjVector containing the selection list of items that were dragged from the drag source. The intent of this sample program is to transfer data between BBjListBox controls, so that the selection is a list of selected rows in the same format as BBjListBox::getSelectedIndices.
`location = event!.getDropLocation().get(0) + 1`	Get the drop location within the drop target. The intent of this program is to transfer data between BBjListBox controls, so we know that the drop location is a zero-based row number.
`if dragSource! = dropTarget! then`	If the user is moving items within a list, then…
`if sel!.size()<>1 then return`	For the sake of simplicity, only allow a single item to be moved at once within a list.
`sel = num(sel!.get(0))`	Get the index of the dragged row.
`if location = sel then return`	If the user dropped the row onto itself, do nothing.
`item$ = dragSource!.getItemAt(sel)`	Get the text from the dragged row.
`if location > sel then`	If the data was dropped below its original location, then…
`dropTarget!.insertItemAt(location,item$)`	…insert it at the new location
`dragSource!.removeItemAt(sel)`	…and remove it from the original location
`else`	…otherwise, the data is being dropped above its original location, so:
`dragSource!.removeItemAt(sel)`	…remove it from the original location…
`dropTarget!.insertItemAt(location,item$)`	…then insert it at the new location.
`endif`	…end of moving within a list.
`else`	…otherwise, the user is moving from one list to another:
`if sel!.size() then`	If the selection includes at least one row, then…

`for i=0 to sel!.size()-1`
`item$ = dragSource!.getItemAt(num(sel!.get(i)))`
`dropTarget!.insertItemAt(location+i,item$)`
`next i`
	Copy each of the selected rows from the drag source to the drop target…

`for i=sel!.size()-1 to 0 step -1`
`index = num(sel!.get(i))`
`dragSource!.removeItemAt(index)`
`next i`
	…and remove them from the drag source.

`endif`	End of move from one list to the other.

`canSelect = available!.getItemCount()`
`selectItems!.setEnabled(canSelect)`
`selectAll!.setEnabled(canSelect)`
	Enable or disable the selection buttons based on whether there are any items left in the “Available” list.

`canDeselect = selected!.getItemCount()`
`deselectItems!.setEnabled(canDeselect)`
`deselectAll!.setEnabled(canDeselect)`
	Enable or disable the deselection buttons based on whether there are any items currently in the “Selected” list.

`endif`	
`return`	

The Drag and Drop Events Example includes a more complex event handler for dragging and dropping between arbitrary controls.

continued…

Drag/Drop Actions

In the first sample program, dragging text from one text control to another causes it to be moved. The user can override this default behavior by pressing the CTRL key while dragging.

In the second sample program, data is always moved. It is not possible to select copy because that wouldn't make sense in this particular application.

In the selection list program, the programmer overrides the default drag and drop actions, specifying that MOVE is the only supported action for all drags and drops involving these controls:

```
available!.setDragActions(sysgui!.ACTION_MOVE)
available!.setDropActions(sysgui!.ACTION_MOVE)
selected!.setDragActions(sysgui!.ACTION_MOVE)
selected!.setDropActions(sysgui!.ACTION_MOVE)
```

A third kind of drag/drop action, the ACTION_LINK that appears as a  , is rarely used. It indicates that the application should establish a linkage between the drag source and the drop target. There is no general definition of what this means in practice; the interpretation of the link action is determined on a case-by-case basis by each application program.

Drag/Drop Data Formats

These sample programs work with plain text, but a user can transfer almost any kind of data with drag and drop. The BBjDropTargetDropEvent::getText method returns plain text and the BBjDropTargetDropEvent::getData method returns all available formats. The MIME type code specifies the data format; common values are "text/plain" and "text/html." Data dragged from a BBjListBox, BBjTree, or BBjGrid transfers as both plain text and HTML-formatted text, while data dragged from BBj text controls only transfers in plain text format. The Drag and Drop Events Example demonstrates how to process non-text data, such as dragging a color from a BBjColorChooser control.

Everything is Negotiable

Drag and drop always involves a negotiation between two controls. When the user initiates a drag, the source control packages its data in a variety of formats. At the same time, it establishes the actions (MOVE, COPY, LINK, or any combination of the three) that it is prepared to support. When the user drags the data over a potential drop target, the two controls enter into a negotiation.

- If the drag source offers a data format and action that the drop target is prepared to accept, the cursor indicates the selected action – ACTION_MOVE ACTION_COPY or ACTION_LINK.
- If the drag source does not offer a data format and action that the drop target is prepared to accept, the cursor is set to ACTION_NONE.
- If the drag source is prepared to offer the data via more than one action that is supported by the drop target, the user can choose between the available options by pressing a modifier key while dragging (SHIFT for MOVE, CTRL for COPY, or CTRL+SHIFT for LINK). The Drag and Drop Actions Example demonstrates the effects of the various drag/drop actions.

Drag/Drop Types

The developer can impose additional restrictions on the type of data that can be dragged between various controls. For example, the list controls in the selection dialog sample only accept drops from each other. This is done by specifying drag and drop type codes for each of the list controls as follows:

```
OptionListBox$ = "OptionListBox"
OptionListBox! = bbjapi().makeVector()
OptionListBox!.add(OptionListBox$)

available!.setDragType(OptionListBox$)
available!.setDropTypes(OptionListBox!)
selected!.setDragType(OptionListBox$)
selected!.setDropTypes(OptionListBox!)
```

The **Drag and Drop Types Example** in the online documentation demonstrates more complex rules for controlling drag and drop behavior between multiple controls.

Summary

This article is a brief introduction to drag and drop in BBj. Drag and drop enables the developer to enhance BBj applications with rich and flexible new features, giving the users a new way to transfer data between GUI controls. For more information and sample programs, see **BBj Drag and Drop** in the online documentation.