The GUI Enhancements of BBj

By Kshanti Greene

When designing the BBj® SYSGUI, we had a few goals in mind and a few ideas on how to improve the syntax and functionality of window and control management. Most important was making the syntax clearer and simpler. We wanted to offer an alternative to SENDMSG() and CTRL functions, which are difficult to program with because of their parameter length limitation and their reliance on a number to indicate the function. If you have programmed GUI in BBx® you probably know how hard it is to remember the number for a function - unless you have a photographic memory!

Another goal was to improve the event-loop format. The event-loop paradigm used to be the standard in many languages that are used to write GUI programs. Today, however, more and more languages are using a messaging style in which components register to be alerted of events. When the event occurs, methods within the component will get called automatically.

Our third goal was to add a tree control to BBj as it is available in other modern GUI programming languages.

And finally, the grid control needed an overhaul. In Visual PRO/5®, the grid is a powerful control but requires extensive coding to maintain. In BBj, we wanted to automate much of the grid behavior and allow it to manage itself once it has been set up.

OUR SUCCESSES

We have succeeded in all these goals for the BBj SYSGUI. First of all, we have introduced the BBjObject syntax. We modeled this syntax after common Object-Oriented programming languages and used the "dot-notation" that is so intrinsic in languages like Java and C++. Currently BBjObject syntax is not available for all functions for all controls, but these will be included soon. We've improved event handling with the introduction of CALLBACKs and, yes, we also have an easy-to-use tree control.

As for the Grid control, we have made many improvements. The grid can now manage its own data, editing and other functions that used to be managed by the BBx program. Once the BBj programmer creates and sets up a grid, he can leave it alone. We have also added new SENDMSG()s to simplify grid setup and over 100 BBjGrid methods that duplicate the behavior of the grid SENDMSG()s as well as add functionality. We have also added BBjObject methods for almost all GUI controls.

CALLBACKs

The benefits of CALLBACKs are numerous. First of all, they follow the event-handling paradigm used in modern GUI programming languages such as Java and C++ for Microsoft Windows. Instead of handling an event loop, and checking each event type and control ID that comes out of the loop, you can now register a CALLBACK for a control, and a specified subroutine will automatically execute when the event occurs on that control. The programmer need only manage the events for controls that he wants to handle.

The following demonstrates the code required to run a Read Record-mode event loop:

```
REPEAT
    READ RECORD
      (SYSGUI,SIZ=EVENT,ERR=LOOP_END)EVENT$
    REM Handle Event
UNTIL EOJ
```

In CALLBACK mode, the loop is handled internally, so the only line needed is:

```
PROCESS_EVENTS
```

The following demonstrates the steps required to retrieve the code and control ID from the Read Record string.

```
ID = event.ID
code$=event.code$
If code$ = goodCode$ and ID = goodID then
       REM Handle event
Fi
```

When a BBx programmer uses Read Record mode, he has to check the type and ID for every event in order to catch those events in which he is interested. The programmer will ignore events in which he has no interest. This creates wasted program execution.

In CALLBACK mode, the CALLBACK specifies the ID and event type:

```
CALLBACK(EVENT_TYPE, HANDLER, CONTEXT, ID)
```

The program executes efficiently because it only handles events that the programmer intended to handle.

Creating a CALLBACK involves three main steps:

1. Register the CALLBACK with an event type, a subroutine to call, and a context and control ID. The following CALLBACK manages the close event on a window with context CONTEXT:

```
CALLBACK(ON_CLOSE, EXIT_PROGRAM, CONTEXT)
```

A CALLBACK for a control also needs a control ID:

```
CALLBACK(ON_BUTTON_PUSH,  BUTTON_PUSHED, CONTEXT, BUTTON_ID)
```

2. Add PROCESS_EVENTS in the correct place in the program. You should place it before the subroutines that may be called in response to the events. This is the verb that runs an internal event loop in the BBj Interpreter:

```
PROCESS_EVENTS
```

3. Create subroutines that the Interpreter will call for each CALLBACK. More than one CALLBACK can call the same subroutine.

```
EXIT_PROGRAM:
      REM Clean Up
      RELEASE
RETURN
```

Additionally, more information can be retrieved about the event using:

```
EVENT$=SysGui!.getLastEventString()
```

This call returns the same string that READ RECORD retrieves in READ RECORD-style event handling. You can also use the NOTICE() function.

An equivalent CALLBACK event type exists for each SYSGUI regular and Notify event. For example:

- Event code "B" = ON_BUTTON_PUSH
- Event code "e" = ON_EDIT_MODIFY
- Grid Notify event 22 = ON_GRID_UPDATE

TREE CONTROL

The BBj programmer can create and manage the tree control using BBjTree methods. SENDMSG()s will not work on it, although some common control mnemonics and CTRL() functions, such as 'SIZE' and CTRL(0), will work on it. Both CALLBACKs and Notify events are available to handle tree events. The following example demonstrates how to create and set up a tree control:

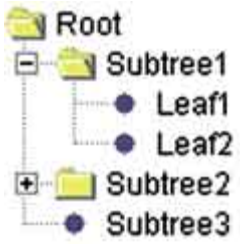
Use a BBjWindow object to create a tree. Set a root before any other nodes are added to the tree. The root is the top-level node in the tree. You must assign each node an ID:

```
myTree! = myWindow!.addTree(id, left, top, width, height)
myTree!.setNode(0,"Root")
```

Once you have set a root, you can add nodes to parent nodes. You have to add the first node to the root node. Add nodes to a parent node by specifying first the child ID and then the parent ID:

```
REM 'The following code will create the
REM 'tree displayed in Figure 1
REM 'Add the first level of nodes to Root node
myTree!.addNode(1,0,"Subtree1")
myTree!.addNode(2,0,"Subtree2")
myTree!.addNode(3,0,"Subtree3")
REM 'Add second level of nodes to tree
myTree!.addNode(4,1,"Leaf1")
myTree!.addNode(5,1,"Leaf2")
myTree!.addNode(6,2,"Leaf3")
```

Figure 1. This figure demonstrates a tree control generated using BBjTree methods.



Our online BBj documentation has more information on tree features and functionality. You can read more at: www.basis.com/onlinedocs/documentation/index.htm

GRID ENHANCEMENTS

Grid enhancements cover several areas including setup and maintenance, automation, BBjGrid methods and data-aware grid improvements.

Setup and Maintenance

We have made several important improvements to the BBj grid control that allow a BBj programmer to set up a grid and then leave it alone, unless he desires special handling.

The most important new feature is data caching. In Visual PRO/5, the grid does not cache data. Therefore the programmer has to watch for table update events and fill the cells with data when the events occur. Once a cell goes off the screen, the grid will "forget" about the data in the cell. In BBj, this is no longer the case. Once a cell has data, the data will remain in the cell until it is changed, either programmatically or by the user. This means that the program no longer needs to watch for the table update event. The programmer can also fill the grid before it is even set visible.

In case a programmer still wants to use the table update event to manage data, which may be useful for grids with a large amount of data, we have added a SENDMSG() to

make data loading more efficient. If the programmer sets SENDMSG(110); Set Update Cached Cells to FALSE, then BBj will only fire table update events for cells that have no data in them. Once the program has put data into a row, then BBj will not fire events requesting a data update for that row. This can greatly improve performance for a large grid that requires scrolling.

The BBj programmer can also handle grid appearance during grid setup, instead of using the Draw Cell SENDMSG() in response to the table update event. We added several SENDMSG()s to allow the programmer to set attributes by column. In most grids, specific properties such as style, color and alignment will be common for all cells within a column. The following SENDMSG()s allow the programmer to set these attributes initially per column, so that he does not need to set them for each individual cell:

```
SENDMSG(102): Sets default background color for a column
SENDMSG(103): Sets default text color for a column
SENDMSG(104): Sets default style for a column
SENDMSG(105): Sets default alignment for a column
```

Again, the benefit of data caching and column-attribute setup is that a programmer can set up a grid during program initialization, and he does not need to handle the table update event in order to set data and attributes in cells. This will also appear to improve performance of the grid because the BBj program can set up the grid while it is not visible.

Automation

BBj automates several functions that need to be managed by the program in Visual PRO/5. These functions include:

- Cell Editing
- Checkbox- and button-style cell handling
- Drag and drop

In Visual PRO/5, the program must call the Start Edit SENDMSG() in response to a user clicking on a cell. This is no longer required. By default, cell editing will begin on a cell when the user double-clicks on an editable cell. If the user changes the text in the cell, then the text will remain as changed when focus on the cell is lost. The grid is editable by default but can be set non-editable by grid, column or cell using SENDMSG()s 106-108. SENDMSG(111) can be used to set the number of clicks to begin editing.

In Visual PRO/5, programs also have to manage the toggling of cells that are checkbox or button style. When a user clicks on a cell, the program must respond to the event and set the style in the cell in order to toggle the state of the check or button. This is not necessary in BBj. By default, a cell that is checkbox or button style will be toggled when the user single-clicks on the cell. A BBj programmer can turn off the automatic toggling by setting the cells to be non-editable.

Drag and drop handling in the grid is also easier. Now in BBj, if you enable drag-and-drop on a grid using SENDMSG(33), then when a user clicks on a cell and drags and releases the mouse over another cell, BBj copies the contents of the initial cell into the second cell. A user can drag-and-drop between any grids within the same BBj process.

These automated features are available by default in BBj. However, the editing and toggling features are sometimes not compatible with Visual PRO/5 behavior. In order to make the BBj grid backward-compatible, we have added a SETOPTS bit. The grid backward-compatibility bit is SETOPTS bit 8(10), and it will turn off automated cell editing and toggling of checkbox- and button-style cells. The grid will behave as in Visual PRO/5. After careful consideration, we decided to make the automated behavior

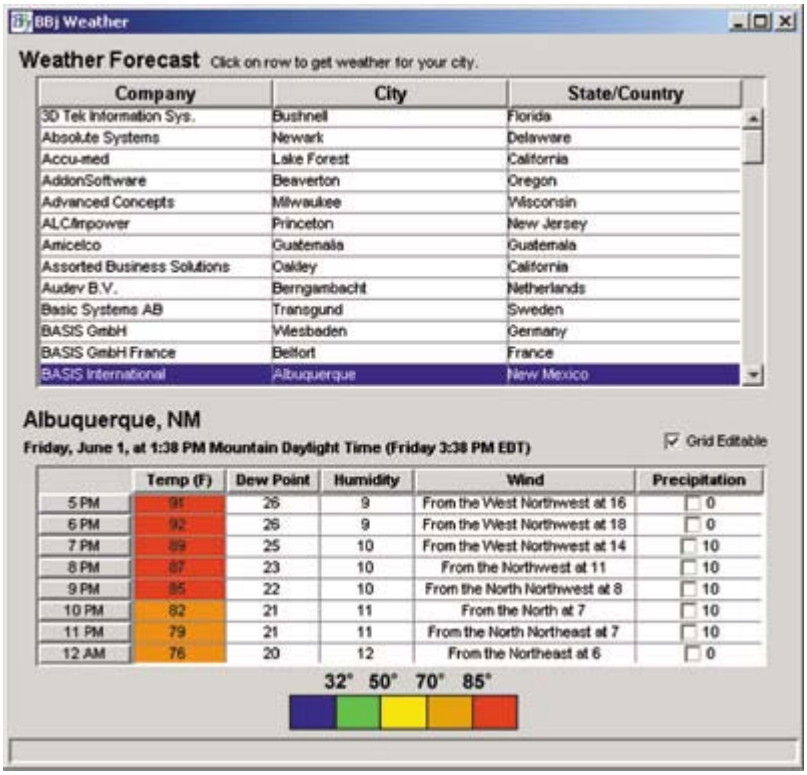
the default in the grid, instead of having to make new programs set a SETOPTS bit to turn on the BBj behavior.

BBjGrid Methods

BBj has introduced Object syntax to simplify and clarify function syntax and to make it easier to add new functionality. Object syntax has many benefits. First of all, it has no parameter length limit. A BBjObject method can have as many parameters as is needed. This means that we do not have to use templated strings in order to pack a lot of information into one function. Parameter and return types are also based on what is needed for the method. Any data type or object can be used as a parameter or return value. SENDMSG() and CTRL() functions require specific types and always return a string.

We have also designed the methods to simplify behavior. In other words, one method has one functionality, instead of cramming a lot of side effects into one SENDMSG(). For example, a program can easily set one attribute in the grid without setting up a Draw Cell SENDMSG() string that is designed to set several attributes at once. The method names describe their functionality, so they are easier to remember than the numbers Visual PRO/5 uses to distinguish SENDMSG() and CTRL() functions. Each grid SENDMSG() has an equivalent BBjGrid method or methods, and we have added functionality as well.

Figure 2. Here is a screen from the TechCon2001 weather demonstration program. The demo uses BBjGrid methods and enhanced grid behavior as well as embedded Java objects. You can find this demo program on this issue's CD in \Program Files\basis\demos\techcon.



Data-Aware Grid

We have made a few enhancements to the data-aware grid as well. First of all, SENDMSG(100); Synch Grid Data will synchronize the grid with the current view of the data. The BBj programmer no longer needs to disconnect and reconnect the data channel to the grid in order to get updated data. Another additional feature is the option to display a confirm message box for the user to confirm if he wants an update to continue. SENDMSG(112) will set this option. Lastly, the programmer does not need

to use the Start Edit SENDMSG() to begin editing on a data-aware grid. Editing will begin when a user double clicks on an editable row.

BBjObject Methods

We've added BBjObject methods for all controls except image lists. A BBj programmer can access each control using the BBjControl methods, which include general methods to set and retrieve size, location, borders and text. Each control also has individual methods to duplicate and enhance the functionality of the mnemonics, CTRL() and SENDMSG() functions. The BBjTabCtrl methods offer a vast improvement over the TABDESC$ string, which required you to set all tab information at once. Now you can add and retrieve individual tab attributes, and even BBjControls, from the tab control. The following methods allow the programmer to use the control object within the tab control, instead of referring to a control ID.

```
addTab(String title, BBjControl)
BBjControl getControlAt(Int index)
```

THE FUTURE OF BBj GUI

We've introduced many improvements to GUI programming in Business BASIC with the release of BBj 1.0. However, we will not stop here. We have many ideas and plans for even more improvements. Some of these future enhancements include:

- AppBuilder: Build resources with full access to all available control methods and add automatic code generation
- More improvements to CALLBACKs, including making additional event information available without using getLastEventString()
- Add Listbutton and other control cell styles to the grid
- More data-aware controls and more powerful data-aware grids

And of course, we are always open to suggestions for improvements...