Get Rid Of File Maintenance Gridlock With Grid Control--Part II

By Kurt Williams

Editor's Note: *The following is the second in a two-part series on Visual PRO/5®'s grid control feature. Data presented in this article is based on the grid sample code currently included with Visual PRO/5. The complete Datagrid coding sample can be downloaded from BASIS' Visual PRO/5 2.0 Coding Samples web page at www.basis.com/visualpro5/samples.html.*

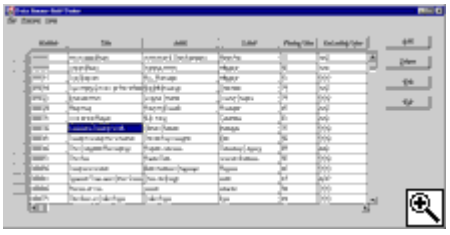
Part one of this tutorial is also available online.

Developing and maintaining file maintenance utilities can be one of the least enjoyable and most time-intensive aspects of Business Basic development. But with the release of Visual PRO/5 2.0, developers have gained the grid control, a powerful new interface tool that can generate a spreadsheet-like display of file data that is easy to move through and easy to read. In addition, a data-aware capability included within the new grid control can help developers keep on top of large amounts of information by handling all the back-end manipulation and formatting of the data in the grid.

In the first part of this series, found in the Autumn 1998 issue of *The BASIS Advantage*, step-by-step instructions covered how to set up a data-aware grid. Once a data-aware grid is created, developers can begin to incorporate the important edit, add, and delete record capabilities into the grid. This article focuses on the final steps needed to add these capabilities and create a fully functional, data-aware grid.

Placing a Cell in Edit Mode

Before editing the data in a given cell, three operations must be set up in the grid-control code: a way for the user to indicate that a cell should be edited, a method to place the cell in edit mode, and a procedure to save the changes.



Selecting a cell to edit.

In the sample program shown, two methods to select a cell are presented--double-clicking on the cell itself and clicking the edit button while the cell is highlighted. Setting up the edit-button method of cell selection is straightforward. Select Form 101 frmDataAwareGrid from the Object drop-down list. Then select Push Button 105 btnEdit from the Control drop-down list and the Button Pushed option from the Event drop-down list. In the GUIBuilder™ edit area, add a single line of code:

```
gosub edit_cell
```

This code will be executed anytime the user clicks on the Edit button.

Next, select Grid 100 grdTestGrid from the Control drop-down list and Grid Double

Clicked from the Event drop-down list. Below the remarks in the GUIBuilder edit area, add the same line of code:

```
gosub edit_cell
```

This code will be executed anytime the user double-clicks on a grid cell.

At this point, an `edit_cell` subroutine needs to be written. Start by choosing New Subroutine/Function from the Object drop-down list. Enter Edit Cell in the Name Subroutine/Function dialog. Below the header remarks in the GUIBuilder edit area, enter the following code:

```
edit_cell:
rem get the current row
  grid_id=num(fattr(datagrid_temp$,"grdTestGrid","ID"))
  row$=sendmsg(gb__sysgui,grid_id,gridGetSelectedRow,0,"")
  row = dec($00$+row$)
rem get the current col
  col$=sendmsg(gb__sysgui,grid_id,gridGetSelectedCol,0,"")
  col=dec(col$)
rem prep the edit
  editparams_desc$="mask:c(1*=0),restore:c(1*=0),"+
: "initstr:c(1*=0),key:u(2),col:u(2),row:u(4)"
  dim editparams$:editparams_desc$
  editparams.key = 0
  editparams.col = col
  editparams.row = row
rem block editing the first column which is the primary key
  if col=0 then
:   msg$="You cannot edit the primary key for this record.";
:   style = msgboxExclamation;
:   title$="No Edit";
:   trash=msgbox(msg$,style,title$)
: else
:   trash$=sendmsg(gb__sysgui,grid_id,gridStartEdit,0,editparams$)
return
```

The code below the first remark extracts the grid ID from the descriptive template obtained in the initialization code and then uses the Get Selected Row function (SENDMSG() Function 45) to retrieve the row number of the selected cell. The code below the second remark uses the Get Selected Column function (SENDMSG() Function 44) to retrieve the column number of the cell.

To use the Start Edit function (SENDMSG() Function 31), a templated string with several values must be created. The code below the third remark creates this string and sets the values for the row and column retrieved above.

Finally, the code below the fourth remark checks to see if the user wants to edit a cell in the first column, which contains the primary key. If so, the program presents a message box informing the user that the primary key cannot be edited. If the user wants to edit a cell in any other column, it issues the Start Edit function (SENDMSG() Function 31) with the templated string that was defined in the first article of this series.

To see the cell edit routine in action, select Run Program on the Program menu and click the Save button in the Save program dialog. The program will present the grid as it did before. Now double-click on a cell and a heavy outline will appear around the cell while the caret is placed at the beginning of the cell. Double-click on any value in the Recording Type column, delete the current contents, and type in a new value. Notice that the cell will only accept alphabetic characters and it converts them to uppercase because of the `AAA' mask that was previously set up for that field in the template. Once the editing of a cell's contents is complete, move the focus to another cell by clicking on it. This action instructs the grid to automatically commit the changes that were made to the first cell.

Adding a New Record

Once it is possible to edit a cell, the add functionality should be incorporated into a grid. To add a new record, a routine must be attached to the Button Pushed event for the Add button.

Begin by selecting Form 101 frmDataAwareGrid from the Object drop-down list. Then select Push Button 103 btnInsert from the Control drop-down list and Button Pushed option from the Event drop-down list. In the GUIBuilder edit area, add a single line of code:

```
gosub add_record
```

This code will be executed anytime the user clicks on the Add button.

Now an **add_record** subroutine needs to be written. Select the New Subroutine/Function from the Object drop-down list and enter **Add Record** in the Name Subroutine/Function dialog. Below the header remarks in the GUIBuilder edit area, enter the following code:

```
add_record:
rem send the add row message
  grid_id=num(fattr(datagrid_temp$,"grdTestGrid","ID"))
  trash$=sendmsg(gb__sysgui,grid_id,gridDataAwareFunctions,0,
: gridAddRow$)
  rem this flag is checked in the Grid Edit Mode start event
  rem so that the primary key value can be set by the program
  rem when the grid start edit event occurs
  add_in_progress=1
return
```

Adding a new record to a file.

This code pulls the grid control ID from the descriptive template and then uses it in the Perform Data-Aware function (SENDMSG() Function 81), a SENDMSG() function used to perform various actions on the data-aware grid. The last parameter is a constant string value set in the Define Constants routine--**gridAddRow$**. This parameter causes the function to add a new row to the grid that will ultimately hold a new record. Finally, a flag called **add_in_progress** is set. Every time a grid cell is placed in edit mode, a Grid Edit Mode event occurs. When this event occurs, and **add_in_progress** is true, the flag indicates that a new record has been added and a new key for that record needs to be generated.

To set that added record logic, select Form 101 frmDataAwareGrid from the Object drop-down list. Then choose Grid 100 grdTestGrid from the Control drop-down list and Grid Edit Mode from the Event drop-down list. In the GUIBuilder edit area, add the following code:

```
while add_in_progress
  rem this code catches the start edit on col 0 (primary key)
  rem and sets the value
  add_in_progress=0
if gb__notice.col = 0 then
: gosub create_new_key;
: trash$=sendmsg(gb__sysgui,gb__notice.id,gridSetEdit,0,newkey$);
: trash$=sendmsg(gb__sysgui,gb__notice.id,gridEndEdit,0,$$);
: desc$="mask:c(1*=0),restore:c(1*=0),initstr:c(1*=0),";
: desc$=desc$+"key:u(2),col:u(2),row:u(4)";
: dim editparams$:desc$;
: editparams.key = 0;
: editparams.col = gb__notice.col+1;
: editparams.row = gb__notice.row;
: trash$=sendmsg(gb__sysgui,gb__notice.id,gridStartEdit,0
:   editparams$)
wend
```

When the Grid Edit Mode event occurs and the **add_in_progress** flag is true, this code will execute. It immediately sets the **add_in_progress** to false and does a gosub to the

create_new_key routine. The new key value is placed in the variable **newkey$** and passed to the grid's special INPUTE control via the Set Edit Text function (SENDMSG() Function 35). This message places the data from **newkey$** into the INPUTE control. It then immediately issues an End Edit function (SENDMSG() Function 26). This ends the edit on the first column. Then an **editparams$** string is set up in the same manner it was created in the **edit_cell** routine earlier. Notice that the column value is bumped by one. A Start Edit function (SENDMSG() Function 31) is then issued with the **editparams$** that places the second column in edit mode.

The Grid Edit Mode event is a Notify event. GUIBuilder automatically retrieves the notice from the SYSGUI device and places it in a templated string called **gb__notice$**. The routine above uses that string to access the row and column information.

To complete the add record process, a **create_new_key** routine needs to be developed. Select New Subroutine/Functionfrom the Object drop-down list and enter Create New Key in the Name Subroutine/Function dialog.

Below the header remarks in the GUIBuilder edit area, enter the following code:

```
create_new_key:
rem ' assign new number
  trash$=fattr(datarec$,"CDNUMBER")
  keylen = dec(trash$(10,2))
  keymask$ = fill(keylen,"0")
  newkey$=keyl(alt_chan,err=cnk_no_keyl)
  done=0
bump_it:
  while !(done)
    newkey$=str(num(newkey$)+5:keymask$)
    done=1
    read(alt_chan,key=newkey$,dom=got_it)
    done=0
    got_it:
  wend
return

cnk_no_keyl:
  rem this handles the case of an empty file
print err
  newkey$=str(1:keymask$)
  goto bump_it
```

This code generates a new key for the file. First, it creates an encoded information string for the CDNUMBER field and then gets the length of the key from the string. Afterwards, the code builds a mask of all zeros that is the same length as the key. Using the KEYL function, the code takes the last key in the file. Notice that it uses the **alt_chan** and not the channel that was bound to the grid. Once a channel has been bound to the grid, the program should avoid any I/O to that channel.

Next, the CDNUMBER value is bumped by five and a test conducted to see if the key is in the file. If the key is not present in the file, the process is done and the variable **newkey$** contains the new key. If it is in the file, the loop is executed again; the **cnk_no_keyl** routine is there to handle the special case of an empty file.

Once the code is completed, select Run Program from the Program menu and click the Save button in the Save program dialog. The program will present the grid as it did before. Now click the Add button. A blank row will appear at the end of the grid and the new key value will be inserted into the first column. The second column will be in edit mode. Once the focus moves off the new row, the record is saved to the file.

Deleting a Record

Now that adding a record is possible, it is necessary to add the record or row deletion

capability. Begin by selecting Form 101 frmDataAwareGrid from the Object drop-down list. Then choose Push Button 104 btnDelete from the Control drop-down list and Button Pushed from the Event drop-down list. In the GUIBuilder edit area, add a single line of code:

```
gosub delete_current_row
```

This code will be executed anytime the user clicks on the Delete button.

Now add the **delete_current_row** routine. Select the New Subroutine/Function from the Object drop-down list. Enter **delete current row** in the Name Subroutine/Function dialog. Below the header remarks in the GUIBuilder edit area, enter the following code:

```
delete_current_row:
rem get the current row
  grid_id=num(fattr(datagrid_temp$,"grdTestGrid","ID"))
  row$=sendmsg(gb__sysgui,grid_id,gridGetSelectedRow,0,"")
  row = dec($00$+row$)
rem create a temporary template to hold row contents
  dim tmp_datarec$:datarec_desc$
rem get the data from the current row
  tmp_datarec$=sendmsg(gb__sysgui,grid_id,
: gridDataAwareFunctions, row,gridRetrieveRow$)
rem confirm the delete
  msg$="Are you sure you want to delete CD Number "+
: tmp_datarec.cdnumber$+"?"
  style = msgboxYesNo+msgboxInfo+msgboxSecond
  title$="Delete Confirmation"
  resp = msgbox(msg$,style,title$)
rem if response is yes then send the delete message
  if resp<>msgboxYes then
: return

rem delete the row
  trash$=sendmsg(gb__sysgui,grid_id,
: gridDataAwareFunctions,row,gridDeleteRow$)
rem disconnect the grid
  trash$=sendmsg(gb__sysgui,grid_id,gridSetDataAware,0,$$)
rem reset the file pointer
  read(data_chan,key="",err=dr_continue)
  dr_continue:
rem reconnect the grid
  trash$=sendmsg(gb__sysgui,grid_id,
: gridSetDataAware,data_chan,datarec_desc$)
return
```

The grid control ID is determined and then the Get Selected Row function (SENDMSG() Function 45) gets the row number of the cell that was selected. Next, a temporary template is dimensioned to hold the data from the grid row. Then, using the row number, a Perform Data-Aware function (SENDMSG() Function 81) is issued with the **retrieve_row** flag. This flag instructs the function to return all the data from the current row into the temporary template. Next, the program presents a message box that asks the user to confirm or deny the deletion. If the deletion is confirmed, the process of deletion proceeds; otherwise, the grid just executes a return.

The actual deletion takes place using the Perform Data-Aware function (SENDMSG() Function 81)--with the **delete_row** flag. Once a record is deleted, the grid will visually indicate this by displaying a deleted icon in the first column and filling all the data fields with asterisks. The only way to clear the empty row is to disconnect and reconnect the data channel. This is accomplished with the last three commands in the routine. A SENDMSG() Function 80 is issued with a channel number of zero. This disconnects the grid from the channel. Then, the file pointer is reset on the primary channel and rebound to the grid with another SENDMSG() Function 80.

To try the delete process, select Run Program from the Program menu and click the Save button in the Save program dialog. The program will present the grid as it did before. Click on any row and then click the Delete button. The program will display a message box asking if you want to delete that row. If you answer "Yes," the deletion will proceed. If you answer "No," the deletion will be canceled.

At this point, the developer has a fully functional, data-aware grid that can manipulate data automatically while providing the ability to edit, add, or delete individual cells in the grid.