DBGrid Multi-Selection

by Dave Bolt

The DBGrid control in 32-bit Delphi has an Options property, which contains a list of features which may be enabled or disabled. Amongst these is the dgMultiSelect option. The help file says that it can be used to select multiple non-contiguous rows using Ctrl+Click or Shift+Arrow keys, and that the behaviour is similar to a multi-select listbox. I haven't found anything in the help about how this capability is used. On looking into the DBGrids unit I found the SelectedRows property, which is of type TBookmarkList. This also does not appear in any of my Delphi help files.

In this article I will outline the properties and methods of TBookmarkList and demonstrate how these can be used to work with the TDBGrid multi-selection capability through two projects: one to demonstrate the basic functionality and the second to demonstrate a simple strategy for saving and restoring selection lists.

The TBookmarkList Class

The TBookmarkList class is defined in the DBGrids unit. It has the accessible properties and methods shown in Listing 1. I am not including anything in the private section of the class definition since the only ways to make use of or change these are through the protected and public methods and properties, or by changing the DBGrids unit itself.

Protected Methods

Note that none of the protected methods is declared as virtual. I cannot see any reason you would wish to override them anyway, since they are really quite fundamental operations with little room for creativity concerning their behaviour.

CurrentRow returns a bookmark for the current row in the dataset attached to the grid. This is not necessarily a bookmark in the TBookmarkList's list (see the discussion on CurrentRowSelected).

```
protected
  function CurrentRow: TBookmarkStr
  function Compare(const Item1, Item2: TBookmarkStr): Integer;
  procedure LinkActive(Value: Boolean);
public
  constructor Create(AGrid: TCustomDBGrid);
  destructor Destroy; override;
  procedure Clear;
  procedure Delete;
  function Find(const Item: TBookmarkStr; var Index: Integer): Boolean;
  function IndexOf(const Item: TBookmarkStr): Integer;
  function Refresh: Boolean;
  property Count: Integer
  property CurrentRowSelected: Boolean
  property Items[Index: Integer]: TBookmarkStr
```

➤ *Listing 1*

```
function TBookmarkList.IndexOf(const Item: TBookmarkStr): Integer;
begin
  if not Find(Item, Result) then
    Result := -1;
end;
```

➤ *Listing 2*

Compare, quite obviously, is used to test equivalence of two bookmarks. It is called by other, public, methods like Find.

LinkActive is used to check if there is an active dataset attached. If not then any bookmarks or operations on bookmarks are considered invalid. It is assumed that if a dataset is disconnected from the grid then the bookmarks become invalid. If you disconnect a grid from a dataset, then reconnect to the same dataset, TBookmarkList assumes it is not the same dataset. It has no simple way of knowing that you did actually reconnect to the same dataset without, for instance, first closing and re-opening. I show an application specific work around for this in the second project.

Public Methods

The Create method takes a TDBGrid as its parameter. The TBookmarkList class is strongly tied to TDBGrid: handing it a TDataSource could have made it much more generally useful. Delphi assumes that a TBookmarkList will be contained by a particular TDBGrid component.

The Clear and Delete methods appear at first sight to do the same thing, *but they don't!* Clear clears all bookmarks from the list, then invalidates the associated grid which causes it to redraw, removing highlighting from any previously selected rows. Delete deletes all bookmarks from the list, and the associated records from the dataset.

The Find and IndexOf methods are also closely linked. IndexOf takes one parameter, a bookmark, and returns an index whereby a value of -1 indicates the bookmark was not found in the list. Find takes two parameters, a bookmark and an index variable. The index parameter is used to return the bookmark's index in the list. The function returns True if the bookmark was found in the list, or False otherwise. IndexOf calls Find, adding an extra call level, but saving you writing the extra logic: see Listing 2.

If you are using the Find function, the calling application must check the return value before attempting to use the index value. IndexOf would be preferred when the index is required, Find would be preferable when you just need to see if the bookmark is already in

the list, although it does require a variable for the second parameter.

The `Refresh` method is used to validate the bookmarks currently in the list. It works through all the bookmarks checking if they are valid and deleting any which are not. On completion it returns `True` if orphaned bookmarks were found. If there were orphans, the grid is invalidated before returning.

Properties

The `Count` property is quite straightforward, it returns the number of bookmarks currently in the list. In normal use the value will be zero or one. When you click on a row it will be selected, so `Count` will change to one, or increment if multi-selecting.

The `Items` property is equally obvious, it returns the bookmark at the given index (remember it's a zero based list). `Count` and `Items` are both read-only properties.

The `CurrentRowSelected` property is read/write. It can be used to check if the row at the current cursor position is already in the list or not (it uses the `Find` method). By setting `CurrentRowSelected` to `True`, you cause the current row to be added to the bookmark list if not already there.

```
if DBGrid1.SelectedRows.Count>0 then
begin
  with DBGrid1.DataSource.DataSet do
  for i:=0 to DBGrid1.SelectedRows.Count-1 do
  begin
    GotoBookmark(pointer(DBGrid1.SelectedRows.Items[i]));
    AddToList;
  end;
end;
```

➤ *Listing 3: Short form of btnCopySelectClick in BMListU1.Pas*

Using The BookmarkList

So, we have the tools, let's make some use of them. I start out with a simple example using a Paradox table. The example is on the disk, but there are a few points to note.

Firstly, I recommend that if you wish to use `MultiSelect` you set the `dgRowSelect` option to `False`. If you don't do this it may be difficult to tell if the current line is selected or not. Try it with the example. You will of course also have to set `dgMultiSelect` to `True`. Remember that the selection behaviour is 'like', but *not* 'the same as' the extended selection behaviour in a listbox component. Clicking on a cell in the grid selects the row. Clicking on the same row again selects it again, but it is already in the list. Holding down the `Ctrl` key and clicking on a cell will toggle the selection of the row. Once a selection has been made, the dataset must not be changed until you have finished with the selection, or it will be invalidated. Applying a range, for instance, can cause a selection to become invalid.

So to the example. See *Other Database Types* below for notes on Interbase. Select which table you wish to open, then open it. Try out the selection facilities in the grid

using the mouse. Use the `Toggle Selection` button to toggle selection of the current record. When you have some rows selected, click the `Copy Selections` button and information from the selected records will be copied to the list. When you have some records selected press the `Delete Selections` button and they will be gone from the table. I included the tables in a separate ZIP file so you can restore them.

Of course, you can do what you want with selected items. I originally sent information from the selected records to print. The workings are simple. A loop works through the items in the `SelectedRows` property of the grid. At the same time it moves the cursor in the dataset to the matching record: see Listing 3.

Other Database Types

I have included an Interbase version of the Paradox table for the first example so you can see that the technique works in general. Remember that Interbase needs to be set up and the Interbase server needs to be loaded before you can use the Interbase data.

The Original Problem

My original problem included a single grid being used to display data from several tables by changing table attachments. Each of the tables remains active, so strictly the list of bookmarks should remain valid, but as soon as a table is disconnected from a datasource the grid's bookmark list becomes invalid. It would be nice to keep this list with the table and restore it when the table is reconnected. Noticing that the `TBookmarkList` implementation requires a data aware grid, and that it refers back to the dataset connected to that grid, it is obvious that we cannot just create an instance of a `TBookmarkList`. On the other hand we can create a list quite readily, and we have access to the list of bookmarks through the `Items` and `Count` properties. It is a simple matter to iterate through the existing bookmarks copying them to another list before swapping datasets. The trick would seem to be setting the list back again when the original dataset is restored.

I have provided this functionality in the second project on the disk, BMListP2. There are two `TTable` components, linked to simple Paradox tables. The form contains a single `TDBGrid` and a button to toggle between the two tables. I have also provided selection buttons as for the previous example. At the top of the form there are a group of status indicators. This particular example uses the simple expedient of moving to the relevant bookmark in the saved list and then selecting the row in the grid in order to restore a bookmark list to the grid.

Points To Note

Compare the `btnCopySelectClick` and `RestoreSelectedRows` procedures in the first and second projects respectively. In the first example project I placed a bookmark on the current cursor position in the data before iterating through the list of bookmarks, then repositioned to that record. In the second project, when a dataset is reconnected to the grid I do not bother with these extra steps. The different strategies are used purely to illustrate the different behaviours. I would normally expect to use the extra steps.

When a selection is changed in the grid there does not seem to be any event to indicate it. This is a slight problem if you want to do anything in response to changes in selection, eg updating the labels I supply in the second project. This is one of the places where it would be nice to be able to override an appropriate method of the `TBookmarkList` class, which is unfortunately private. As it is, I take a simple brute force approach, check and update on a timer event.

Notes

The ZIP file on the disk for this article contains two projects and another ZIP file containing the data for the first project. The data for the first project consists of a Paradox table and an Interbase table based on the ANIMALS.DBF table from the DEMOS directory, but without the blob fields.

The second project uses the COUNTRY.DB and BIOLIFE.DB tables in the DELPHI DEMOS directories. The DBDEMOS alias is expected. This data is not included in the ZIP files, but is part of the standard Delphi installation.

Dave Bolt is a self-employed Analyst/Programmer, currently working on document storage and retrieval software for Varcarme Ltd in Sheffield, UK.