THE DBLOOKUPCOMBOPLUS COMPONENT

The new **DBLookUpComboPlus** gives you all the lookup power of Delphi's original DBLookUp-Combo plus much, much more;

Major Enhancements
* The ability to sort the drop down list.
* The ability to search through the dropdown list via an incremental keyboard search. (Quicken Style).
* Autofill - The text is automatically filled in as the user types characters. (Quicken Style).
* The ability to enter new records into the lookup table on the fly. (OnNewLookUpRec)
* The control can stay in a Read/Write style even when the Data source and Lookup Source are different. (this goes with the previous point)
* A new event to support the filling the lookup list with the results from a TQuery. (OnPrepareList).
* Ad new event that allows the lookup to return multiple data elements instead of just the one specified by the LookupField property. (OnGridSelect)
* The drop down list can be either left or right justified.
* The drop down list can be changed to a rise up list.
* The BoarderStyle property is available.
* The ability to hide or show the drop down speed button.

Other minor enhancements over the original control include;
- Corrects the two memory leaks found in TDBLookUpCombo.
- Automatic placement of dropdown list to fit the screen has been enhanced.
- If the old TDBLookUpCombo was placed on a form with fsStayOnTop set then the list would drop down behind the form and could not be seen. This problem has been corrected in TDBLookupComboPlus.
- The minimum height of the edit box portion of the control has been corrected to be more inline with the behavior of TEdit, TDBEdit, TCombo, etc.

Like with the original DBLookupCombo, the user can still display text but store a different ID number behind the scenes. Now all the other behaviors your users have come to expect from a truely intellegent combo box are here too. Since DBLookupCombo uses a table and not a query, the creation of the dropdown list is fast and the number of Items in the list is unlimited.

This document has four section. Additional information is available in the components on-line help file.

I. INSTALLATION

Complete installation of this component requires that you
1. Copy the distributed to your disk drive.
2. Install the component into the Delphi Component Pallet.
3. Merge the components help into the Delphi help system.
4. Set up the demo.

1. Copy the distributed to your disk drive.
   - Unzip the DBLUPLUS.ZIP archive to your hard drive.
   - Create a directory and unzip the files into the new directory.
   - Copy the DBLUPLUS.HLP file to the \Delphi\Bin directory.
   - Copy the DBLUPLUS.KWF file to \Delphi\Help.

You may keep the other files together in one directory as they are now or move just the remaining component file off to a component directory by themselves. Keep the files listed as component files together and the files listed as demo files together.

(Component Files)

DBLUP2 DCU	The Working part of the TDBLookUpComboPlus Component
DBLUPLUS PAS	This is the one you need to do the component install
DBLUPLUS DCR	Component Resource File. This contains the pallet bitmap.
PLUS WRI	This file. The documentation

({**Help related files**}

DBLUPLUS.HLP	Help file must be copied to \delphi\bin
DBLUPLUS.KWF	Help key word file. Must be copied to \delphi\help

(Show off Demo Files)

PLUSDEMO.DPR	Demo project file
PLUSDEMO.RES	Demo resource file
MAINFORM.PAS	Demo main form
MAINFORM.DFM	Demo main form
ABOUT.PAS	About form
ABOUT.DFM	About form

(Tutorial Demo #2 Files)

DEMOPROJ DPR	Demo project file
DEMOPROJ RES	Demo resource file
DEMOMAIN PAS	Demo main form
DEMOMAIN DFM	"
SUBCAT PAS	New restaurant type entry dialog
SUBCAT DFM	" "
SUBCAT DB	Demo Data file
SUBCAT PX	"
SUBCAT XG0	"
SUBCAT YG0	"
LOCATION DB	"
LOCATION PX	"
REST DB	"
REST PX	"

(Tutorial Demo #3 Files)

QRYPROJ DPR	Demo project file
QRYPROJ RES	Demo resource file
QRYMAIN PAS	Demo main form
QRYMAIN DFM	"

2. Install the component into the Delphi Component Pallet.

- Start Delphi, choose Options|Install Components and then
- Click the Browse button and locate DBLUPLUS.PAS in your new directory.
- Select it.
- Press OK in the Install Components dialog and wait for the Library to rebuild.

If you want to do something other than this simple install here's the info you need to get creative. I've listed the demo files and the component files separately in case you need to put them is separate directories.

3. Merge the components help into the Delphi help system.
OK this is the toughest part install so pay close attention.
Install the Keyword File
    a. Exit Delphi if it is running
    b. Make a backup of \delphi\bin\delphi.hdx
    c. Run the HelpInst application from \delphi\help (should be in you Delphi Group)
    d. Open \delphi\bin\delphi.hdx
    e. If the existing KWF files report "not found" then add \delphi\help to the search path by selecting Options|Search Path.
    f. Select the Keywords|Add File menu choice and select DBLUPLUS.KWF from \delphi\help directory where you copied it in step 1 above.
    g. Select File|Save
    h. Exit HelpInst
The new components help has now been merged into Delphi's help system.

4. Set up the demo.
    - The demo program expects an alias called 'beta'
    - Go to Tools|BDE Config and define an alias named 'beta'
    - Set it's directory path to the directory where you but the demo files in step 1.
    - Finally, Set the Active properties to true for each table component.

The demo should now function properly..

II. COMPONENT DESCRIPTION

The original DBLookUpCombo had two styles, csDropDown and csDropDownList. The new functionality is added through two new styles, csIncSearch and csIncSrchEdit. The relationship between the old and new styles is clearer if you think about them supporting an edit box that is either editable or non-editable. The old/new, editable/non-editable relationship looks like this.

	Editable Style	Non-Editable Style
Original Styles	**csDropDown**	**csDropDownList**
New Styles	**csIncSrchEdit**	**csIncSearch**

DBLookUpCombo adds three properties, one event and two styles to Delphi's original DBLookUpCombo. The two new styles were discussed above. Here's a brief description of the new property and new event.

Property **AutoDropDown** - Set this property to TRUE if you want the list to automatically drop down when the user starts to type in the field. This applies only to the two new field styles csIncSrchEdit and csIncSearch. Set to FALSE and the list does not drop down but the auto fill-in and incremental search still function.

Property **BoarderStyle** - This property was published and the two choices bsSingle (default) and bsNone are now available. To support the bsNone choice the control was also modified to support greater flexibility in it's height. The original TDBLookupCombo was severly limited as to it's minimum height. The only minimum height now is the height of the tallest letter with no margin above or below. It's up to the designer to select a height which looks right.

Property **DropDownAlign** - Choose "Right" to right align the drop down list and "left" to .... etc.
This property only applies when the DropDownWidth property has been assigned a value.

Property **DropDownTop** - Choose the "Above" option to force the list to rise up above the edit component of the control. The Below option is the default with the list dropping down. Note that choosing "Above" or 'Below" is really only a request to the component. If there is not enough room for the list to display as requested it will display in the way that fits. If the list can not completely display in either direction it will drop down and fall off the end of the screen.

Property **Height** - While this property was in the original TDBLookupCombo it has been modified here to support more flexibility in the minimum height. The designers of Delphi limited the minimum height because of a bug in a Windows Paint routine. This bug seems to effect different fonts differently and the minimum height requirement is too conservative for many fonts. There is still a minimum height but it is now exactly the height of the font. **BEWARE** - It is now sometimes possible, due to this paint bug, to define a height which is too small. In this case, no text will display in the edit portion of the control. If this happens just increase the height a little.

Property **LookUpIndex -** Use this index to specify the sort order of the drop down list. This is also the index used for the incremental searches. If this property is left blank then the display order of the drop down list will be the same as the index set in the lookup table component which is the index required to do the lookup. *You need to be really clear on this - there are two indexes at play here one to do the lookup and another to sort the list for display and search purposes.*

Property **ShowSpeedButton** - Hide the drop down speed button by setting this property to False.

Event **OnNewLookUpRec** - Use this event if you want to add a new record to the lookup list. This event will only be called if the style is csIncSearchEdit or csDropDown which are the two editable styles. The event is not used by csDropDownList or csIncSearch. You can either just create a new record and post it or you can bring up a dialog box where the user would enter the new record and save it. The event is defined as

```
procedure OnNewLookupRec(Sender: TObject;  var Canceled: Boolean);
```

If you decide to use this event handler you **must** follow these rules or it will fail and your application will GPF. Here are the rules:

1. You must set the canceled return var. The component defaults the value of canceled to True and cancels the edit. This is safe but annoying. You need to set this value to true for the edits to hold.
2. Insert a new record into the lookup table.
3. Update the new record with the new lookup value and any other column data.
4. Post the record.
5. Set the DBLookUpComboPlus.Value property equal to the value of the field in the new record that corresponds to the datafield. Note that this step comes after post.

The above steps must happen in stated order. Here are two code templates for OnNewLookUpRec. The first for when the code just creates the record and the second when a dialog box allows the user to create the record.

This code template is probably most useful in the case where the lookup value and display value are the same and the lookup table is a simple single field that contains the lookup/display string.

```
procedure DBLookupComboPlus1NewLookupRec(Sender: TObject;  var Canceled: Boolean);
begin
  TableLookup.Insert;
    {set the tables field values as appropriate}
  TableLookup.Post;
  DBLookupComboPlus1.Value := (value used to fill the lookup table's record);
  Canceled := False;
end;
```

The next code template is for when you want a dialog box displayed for the entry of the new lookup record. This is almost required for the situation when the lookup and display values are different.

```
procedure DBLookupComboPlus2NewLookupRec(Sender: TObject;  var Canceled: Boolean);
begin
  LookUpEntryDlg.TableLookUp.Insert;
  LookUpEntryDlg.TableLookupDisplayField.Value :=
      DBLookupComboPlus1.DisplayValue;
      {above presets sets a field in the dialog box prior to showing it}
  LookUpEntryDlg.ShowModal;                          { display the dialog box }
  if LookUpEntryDlg.ModalResult=mrOK then            { if OK then save the new vendor}
  begin
    LookUpEntryDlg.TableLookUp.Post;         { if user said ok then post}
    {*** VERY IMPORTANT*** Now Update the Combo's value property.}
    { The Combo component doesn't know anything about the table that  }
    { the LookUpDlg box uses so you must tell the combo what the        }
    { lookup value is. This will need to be done in any case where the     }
    { lookup field is different than the display field.                        }
    DBLookupComboPlus1.Value := LookUpDlg.TableLookUpValueField.asString;
  end
  else
  begin
    LookUpEntryDlg.TableLookup.Cancel;
    Canceled := True;
  end;
end;
```

Event **OnGridSelect** - This event is fired when ever an element is selected in the dropdown list. The primary use of this event is to update other fields with information from the just selected lookup record. Use this event when you want information on multiple fields in the lookup record.

Event **OnPrepareList** - Use this event when you want to do something special to the lookup table. Specifically this event is useful for preparing a temporary lookup table by filling it with the results from a query. This event is executed just before the sort order specified in the LookupIndex property is applied to the lookup table. The Demo #3, later in this document shows one way to use a TQuery and OnPrepareList to fill the lookup combo's list.

III. THE DEMO PROGRAMS

There are three demo programs. One that shows off some of the slick UI features of the control and attempts to prove that it is very much a "Quicken Style" control (Demo #1). The "Show off" demo project is called PLUSDEMO.DPR. It uses the customer.db file that came with Delphi.

The second demo (Demo #2) is less visually appealing but is more useful for the purposes of learning how to use TDBLookupComboPlus. The project file for the second demo is DEMPROJ.DPR. It is highly recommended that you study this demo.

The third project/demo (Demo #3) shows how to use a TQuery instead of a TTable to populate the dropdown list of the LookupCombo. This project is QRYPROJ.DPR.

The rest of this section describes the second and third demo projects Make sure you followed the installation instructions above to insure that the Alias is set correctly for these demos.

DEMO #1 - The Major new Features of TDBLOOKUPCOMBOPLUS

This text and the accompanying demo project/form (DEMOPROJ DPR) document the new TDBLookUpComboPlus control. The material is presented in an order that progresses from the simplest use of the component to the most complicated. In the process TDBLookUpComboPlus is compared with the original TDBLookUpCombo that came with Delphi. Much of the behavior and most of the properties in the new control are exactly the same as the old control so you can refer to Borland's documentation. In fact you should become familiar with the DBLookUpCombo since this documentation only talks about what is different between the two.

Tab Page One - csDropDownList

The csDropDownList is most limiting style in either DBLookupCombo or the new DBLookupComboPlus since it is read only. That is, when this style is set the user is limited to only the choices available in the list. This is ideal for situations where data consistency is required. Examples would be any situation where the field's value is being used to join files, or situations where limited, predefined, categorizations are required.

The New DBLookupComboPlus's version of csDropDownList adds some functionality to Borland's DBLookupCombo csDropDownList style. Specifically you can specify an index name in the LookUpIndex property and you drop down list will be sorted. I also enabled the home and end keys for when the list is dropped down. <Home> jumps to the beginning of the list and <End> jumps to, well..., the end.

Select the csDropDownList tab page in the demo and have a look around. The field labeled Cuisine gets an integer value (SubCat) from the underling main table then uses this value to lookup the text equivalent in the lookup table (SubCat) and then display the text on the screen. You will notice that when you drop down the attached list it is not sorted. Go to the new LookUpIndex property and set it's value to bySubCatName. Now when the list drops it will be in alpha order.

The other lookup field on this tab page is labeled location and displays a string that comes directly from the main table. The lookup table contains the list of valid strings. This is the case where the lookup field and display field is the same. The list in this case is always in alpha order.

The main limitation of this style is lack of any kind of search capability in the drop down list. The new csIncSearch style addresses this later.

Tab Page Two - csDropDown

The csDropDown style is the default. It along with csDropDownList were the original DBLookUpCombo's only two styles. This is the editable style.

Even though csDropDown is the editable one of the two choices it is still severely limited since as soon as you turn on a lookup where the data field and the display field are different it becomes readonly anyway. There's a very good reason for this, and that is, that since what's being stored in the main table is the data field value but the user enters in a brand new display value the application can't possibly know what to store into the table. This behavior is demonstrated in the field labeled Cuisine on the second tab page. The new csIncSrchEdit addresses this limitation. Later.

If the display field and lookup field are the same (like with the field labeled location) then you can use the new OnNewLookUpRec event to add records to the lookup table. Refer to the description of the event above. Try attaching the following code to the **OnNewLookUpRec** event handler for the ComboPlusLocation2 component on the second tab page next to the Location label. Make sure you attach this to the component on the second tab page the one titled csDropDown!

```
procedure TForm1.ComboPlusLocation2NewLookupRec(Sender: TObject;
 var Canceled: Boolean);
begin
 TableLocation.Insert;
 TableLocationLocation.Value := ComboPlusLocation2.DisplayValue;
 TableLocation.Post;
 ComboPlusLocation2.Value := TableLocationLocation.Value;
 Canceled := False;
end;
```

Now when ever you enter a value into the location field that doesn't previously exist in the drop down list, it is added to the drop down list.

Tab Page Three - csIncSearch

Incremental searches! Hummm..... There's not really much to say here. Take a look around and make sure you understand how they work and are set up. The important thing in the set up is that you must assign a lookup index if the display and data fields are not the same like with the field labeled Cuisine.

Remember that csIncSearch is like csDropDownList in that the attached edit box is not editable.

Drop down the list and press the home or end keys. Notice how it jumps to the beginning or end of the list. The original DBLookUpCombo did not do this.

Tab Page Four - csIncSrchEdit

That brings us to the last style type and the grand finale. Not only does this style sort the list, do incremental searches, allow for a read/write edit box but it also lets you add new lookup records to the lookup table even in situations where the display field and data field are different.

Give it a try. Go to the control next to the Cuisine label on the **csIncSrchEdit** tab page and type in some cuisine that's not in the list. Tex-Mex for example. Use the drop down list to find a Cuisine ID# that hasn't been used yet. Note that this dialog box uses a lookup table on it self as an aid in assigning a unique ID number. After typing in the new string press tab and a dialog box appears for entering the new lookup record. Review the code for the **OnNewLookUpRec** event attached to the component to see how it's done.

DEMO #3 - Using a TQuery instead of a TTable with TDBLookupComboPlus

The third demo project is called QRYPROJ.DPR and shows how to use a TQuery to populate the lookup list. The new event, OnPrepareList, is used to fill a temporary table with the results of a query on the VENDOR.DB table that shipped with Delphi. The query used here is very simple but complexity is not an issue. Any query may be used.

Load QRYPROJ.DPR and run it. Chick on the <All> radio button and the dropdown list will contain all the vendors in the file. Click on <Preferred> and only the best vendors are listed. That's it! You should study the source code and see how this works. The following are some things to consider as you read the source.

By way of overview, here's what going on. First the form was designed with two working controls, a TRadioGroup which is used to determine the query, and a TDBLookupComboPlus. There is also a TQuery, a TTable and a TBatchMove object on the form. When the program runs a temporary table is created, the default Query is executed and the result is batch-moved to the newly created table. When ever the list is dropped the LookupCombo looks at the status of the TRadioGroup and, if necessary, executes a new query to refill the temporary lookup table. When the form closes down the temporary table is deleted.

The style of the combo box in this demo has been set as csIncSearch. If you want to allow the user to enter new records into the lookup table using the NewLookupRec you may. Remember, though, that the new record should go into the original table that was queried, not the temporary table. Records inserted into the temporary table are just that, temporary.

In this demo an empty temporary lookup table is created when the form is created (in the source see the main forms onCreate event) and then destroyed when the form is destroyed. This is only one possible strategy. As the application developer you need to consider how you deal with temporary tables that hold query results. You should consider, where to put the temporary table, and what happens if the user runs multiple instances of the application.

If you look at the main forms onCreate method you will notice that the temporary tables DatabaseName is set to Session.PrivateDir. In most cases this is sufficient to insure that the temporary table will be created on the local work station.

The second issue of running multiple instances of the application is trickier. The problem is that one instance of the program may fill the table with one set of query results while another instance will fill the table with a completely different set of records. Some possible strategies for resolving this include
- Re-filling the temporary table from the query each time it drops.
- Creating a different temporary table for each instance of the program.
- Not allowing multiple instances of the program.

The third option was chosen for this demo. See the code in QryProg.DPR.

In setting up your own query you will also want to ensure that the query and batch-move only happen when they need to. Notice how LastRadioItemIndex property ensures that the query is only rerun if the criteria has changed.

This implementation of the TQuery > BatchMove > TTable strategy is only one of many possible. The above text will hopefully get you focused on the issues. Use this demonstration only as a starting point and be flexible.

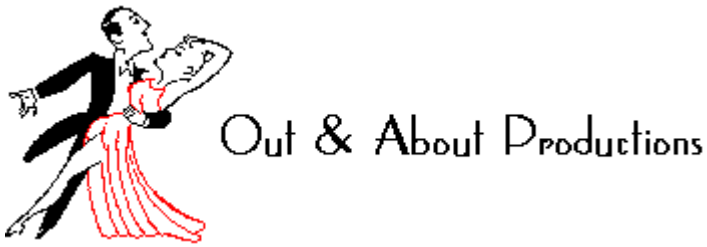
IV ORDERING INFORMATION - SHAREWARE

The shareware version of this components is fully functional inside the Delphi environment. Attempt's to run this a program that uses this component will fail as a stand-a-lone executable. If you find this control useful and would like to use it in your applications you must order the regular version.

For $20(U.S.)(Calf. residents + 7% SST), you will receive the full version plus any updates for the next year. MasterCard and Visa are excepted. The component will be sent to you by Compuserve e-mail. The source code is available for an additional fee. Please e-mail your requests. Any technical questions regarding this component should forwarded to the email address below.

The above price includes all updates and upgrades to the component for a year and technical support via Compuserve.

Alec Bergamini
CompuServe ID 75664,1224



Out & About Productions
8526 Lepus Road
San Diego, CA 92126

Acknowledgments

Special thanks to all beta testers who did lots of work just to get a free copy of this control especially;
Larry Tanner
Deven Hickingbotham
& Mike Orriss
also thanks to
Richard M. Delfs
Dave Bhatia
for their suggestions.