TDBLookUpComboPlus Component

Properties Methods Events Tasks

Unit

DBLUP2

Description
The TDBLookupComboPlus adds to the functionality of the TDBLookupCombo. The user should be aquatinted with the original control since the help for the new component focuses only on the changes. TDBLookupComboPlus adds the following capabilities;

- The ability to sort the dropdown list.
- The ability to display the list either left or right justified and above or below the edit box.
- The ability to incrementally search the drop down list as the user types.
- The ability to add new records to the lookup table on the fly.

A TDBLookupComboPlus is a cousin to the original Delphi control TDBLookupCombo. Like the original control this component is a data-aware combo box that "looks up" a value in a lookup table. The user is now assisted in this lookup with the ability to sort and search the lookup table. Additionally, the ability to add new records to the lookup table at the time of data entry into the main form has been added via a new event OnNewLookupRec.

TDBLookupComboPlus adds to the functionality of the original component by allowing the user to sort the lookup table on a different key then the actual lookup. That is that for the period of time that the list is dropped down the lookup table is in LookupIndex order. In addition to sorting the Drop Down list this component supports incremental searches of the list. As the user types the highlight in the list moves to the closest match in the list. The major advantage of this approach is that the number of keystrokes required to fill the field with the correct data is substantially reduced and accuracy is improved. This incremental search capability is implemented through the Style property. Two new styles are added to the existing csDropDown and csDropDownList. These are csIncSearch which is delivers a read-only incremental search and csIncSrchEdit which is an incremental search with the flexibility to enter data that is not in the lookup list.

The DropDownCount and DropDownWidth properties determine how long and how wide the drop-down list of the combo box is. In addition to these features which were in the original control, the TDBLookupComboPlus control also allows the user to determine the direction of the list when it Drops Down . Use the DropDownTop and DropDownAlign properties to control the list placement.

Properties

The properties listed here are either in addition to or modified from those in the TDBLookupCombo Component.

▶ Run-time only
🔑 Key properties

- 🔑 AutoDropDown
- 🔑 DropDownAlign
- 🔑 DropDownTop
- ▶ IgnorLUIndexErr
- ▶ ListVisible
- 🔑 LookupIndex
- 🔑 Style
- 🔑 ShowSpeedButton

DropDownAlign Property

Example

Applies To - TDBlookupComboPlus at both run time and design time.

Declaration

property DrowDownAlign : TLeftRight;

Note that TLeftRight is defined as
Type
    TLeftRight = (Left, Right);

Description
The DropDownAlign controls how the drop down list justifies it self in relation to the edit box part of the control. There are only two possible values for this control, **Right** and **Left**. Choose **Right** to right allign the drop down list and **Left** to left align the list. This property only appiles when the DropDownWidth property has been assigned a value.

Note that the original TDBLookUpCombo allowed the list to display off of the visible screen when there was not enough room. This problem can now be controled with this property. If fact if you try to drop the list to the left and there is no room it will automatically switch to the right. Same if you try to drop it to the right and there is not enough room it will switch to the left. If there is not enough room in either direction it will right justify and go off the screen.

Example

This example increases the width of the dropdown and then left justifies it.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   DBLookUpComboPlus1.DropDownWidth := 250;
   DBLookUpComboPlus1.DropDownAlign := Left;
end;
```

DropDownTop Property

Example

Applies to

TDBlookupComboPlus for both design time and run time.

Declaration

```
property DropDownTop: TBelowAbove;

Note that TBelowAbove is defined as
Type
    TBelowAbove= (Below, Above);
```

Description

The original TDBLookupCombo defaults to drop down below the edit box portion for the control. In the original control it will only rise up in the case where there is not enough room on the screen below the control. Use the DropDownTop property to over ride this default and force the list to rise up.

The default selection is **Below** which causes the list to appear beneath the edit box portion of the control. Select **Above** to force the list to rise up above the edit box.

Note that if this property is set to **Above** and there is not enough room to display the whole list it will automatically switch back to the drop down state.

Example

This example causes the list box portion of DBLookupComboPlus1 to rise-up above the edit box portion of the control:

```
procedure TForm1.DBLookupComboPlus1Click(Sender: TObject);
begin
   DBLookupComboPlus1.DropDownTop := Above;
end;
```

IgnorLUIndexErr Property

Example

Applies to

TDBlookupComboPlus for run time only.

Declaration

property IgnorLUIndexErr: Boolean;

Description

The IgnorLUIndexErr property is seldom used and as such it has not been published. This property exists for the rare case when two TDBLookUpComboPlus controls are accessing the same field data and a lookupIndex property is assigned. Under these circumstances the LUIndex error message is reported when a new selection in the dropdown list is selected. The reason is that the control that has the list dropped has temporarily changed the tables index to LookupIndex but the other control tries to lookup its value using the lookup index. Normally this is not a problem. Setting the value of IgnorLUIndexErr to **True** simply ignores the error message.

Example

This example turns off the reporting of the LUIndexError:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   IgnorLUIndexErr := True;
end;
```

ListVisible

Applies to

TDBlookupComboPlus for run time only and readonly.

Declaration

property ListVisible: Boolean;

Description

Use to determine if the list is visible. Returns true if the list is visible, otherwise it returns false.

LookupIndex Property

Example

Applies to

TDBLookupComboPlus for both design time and run time.

Declaration

property LookupIndex: String;

Description

The LookupIndex property identifies the index used for sorting the data in the drop down list.   If no value is assigned to LookupIndex, the table's primary index will be used to order the records.

For dBASE tables, the index must reside in the table's master index file. Non-maintained indices are not supported.

If no value is assigned to LookupIndex then the component behaves in the same as the original TDBLookupCombo.

Example

This example changes the sort order of the drop down to be by company name.:

```
procedure TForm1.DBLookupComboPlus1Click(Sender: TObject);
begin
   DBLookupComboPlus1.LookupIndex := byCompanyName;
end;
```

Style Property

Example

Applies to

TDBLookupComboPlus component both design time and run time.

Declaration

property Style: TDBLookupComboPlusStyle;

Where TDBLookupComboPlusStyle is defined as

Type

TDBLookupComboPlusStyle = (csDropDown, csDropDownList, csIncSearch, csIncSrchEdit);

Description

The Style property determines how a database lookup combo Plus box displays its items. These are the possible values:

csDropDown - Creates a drop-down list with an edit box in which the user can enter text. This method supports the OnNewLookupRec event. See csIncSrchEdit below.

csDropDownList - Creates a drop-down list with no attached edit box, so the user can't edit an item or type in a new item.

csIncSearch - Creates a drop-down list with no attached edit box, so the user can't edit an item or type in a new item. This style supports a readonly incremental search.

csIncSrchEdit - Creates a drop-down list with an edit box in which the user can enter text. As the user enters text the drop down list is incrementally searched. This style allows the user to enter items which are not in the drop down list so it is somewhat simialr to csDropDown except that it supports the incremental search. Further, this style not only supports entry of new items into the main DataSource it also supports a mechanism for adding records to the Lookup table. See the OnNewLookupRec event for more details.

The default value is csDropDown.

Comments

As in the original TDBLookupCombo, if the value of the LookupDisplay property differs from the value of the LookupField property then database lookup combo box will function as if its Style is csDropDownList, regardless of the value of the Style property.

Another behavior of the original control was that if the DataSource was not assigned and the Style was set to csDropDownList then the control would function as if it were a csDropDown. This last behavior has been modified. Now if no DataSource is defined a csDropDownList remains a csDropDownList.

Also in the original TDBLookupCombo when the style was set to csDropDownList the <home> and <end> keys did not navigate to the beginning and end of the list. This has been corrected in TDBLookupComboPlus. In the TDBLookupComboPlus if the Style property is set to either csDropDownList or csIncSearch (the readonly styles) the <home> and <end> keys navigate the list. If the Style is set to either csDropDown or csIncSrchEdit (the read/write styles) then the <home> and <end> keys navigate the edit box.
All four of the styles support a sorted list by assigning a value to LookupIndex.

Example

This example changes the style to csIncSearch:

```
procedure TForm1.DBLookupComboPlus1Click(Sender: TObject);
begin
   DBLookupComboPlus1.Style := csIncSearch;
end;
```

AutoDropDown Property

Example

Applies to

TDBLookupComboPlus for both design time and run time.

Declaration

property AutoDropDown: Boolean;

Description

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.

Example

This example causes the DBLookupComboPlus to drop down when the user starts to type in the edit field.

```
procedure TForm1.DBLookupComboPlus1Click(Sender: TObject);
begin
   DBLookupComboPlus1.AutoDropDown := True;
end;
```

ShowSpeedButton Property

Example

Applies to
TDBLookupComboPlus for both design time and run time.

Declaration
property ShowSpeedButton: Boolean;

Description
Hide the drop down speed button by setting this property to FALSE. Show the drop down speed button by selecting TRUE.

Example

This example hides the drop down speed button.

```
procedure TForm1.DBLookupComboPlus1Click(Sender: TObject);
begin
   DBLookupComboPlus1.ShowSpeedButton := False;
end;
```

Events

There is only new event defined for TDBLookupComboPlus that was not in the original TDBLookupCombo Component.

▶ Run-time only
🔑 Key properties

- 🔑 OnNewLookupRec
- 🔑 OnPrepareList
- 🔑 OnGridSelect
- 🔑 OnAfterSearch
- 🔑 OnBeforeSearch

OnNewLookupRec Event

Example

Applies to
TDBLookupComboPlus component

Declaration
property OnNewLookupRec: TNewLookUpRecEvent;

Where TNewLookUpRecEvent is defined as
```
Type
   TNewLookUpRecEvent = procedure(Sender: TObject; var Cancelled: Boolean) of object;
```

Description
This event applies to the two editable (read/write) styles of the TDBLookupComboPlus controls; csDropDown and csIncSrchEdit. Use this event to add new records to the lookup table. This event has no effect in the non-editable styles; csDropDownList and csIncSearch. See the Styles Property.

In this event handler a new record can be inserted into the Lookup table. This can be done by simply creating a new record and inserting it or by displaying a dialog box for the user to enter the new record. Refer to the example to see how this is done.

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. Refer to the code segments in the Example to see two specific examples on implementing this event.

Example

This first 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 user pressed 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;
```

OnPrepareList Event

Example

Applies to

TDBLookupComboPlus component

Declaration

property OnPrepareList :   TNotifyEvent;

Description

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 executes just before the sort order specified in the LookupIndex property is applied to the lookup table.

Example

This example uses the OnPrepareList event to fill a temporary lookup table with records.

```
procedure TForm1.DBLookupComboPlus1PrepareList(Sender: TObject);
begin
   Query1.Close;
   case RadioGroup1.ItemIndex of
      0 : Query1.Params[0].AsString := Something;
      1 : Query1.Params[0].AsString := Something else;
    end;
   Query1.Open;
   { Get the temp table ready for the new data}
   Table1.EmptyTable;
   { Move the data from the query result to the Temp Table}
   BatchMove1.Execute;
end;
```

OnGridSelect Event

Example

Applies to

TDBLookupComboPlus component

Declaration

property OnGridSelect :   TNotifyEvent;

Description

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.

Example

This example updates the data displayed in the EditDept and EditState fields in Tform1 when ever the user selects a row in the drop down list.

```
procedure TForm1.DBLookupComboPlus1GridSelect(Sender: TObject);
begin
    EditDept.Text := LookupDBTableFieldDept.Value;
    EditState.Text := LookupDBTableFiedlState.Value;
end;
```

Methods

There are no new methods defined for TDBLookupComboPlus. Please refer to the help for TDBLookupCombo for information on all available methods.

Using the TDBLookupComboPlus Component

TDBLookUpCombo Plus Reference TDBLookupCombo

Purpose
The DBLookupComboPlus component extends the idea of the original TDBLookupCombo component and can be used as a direct replacement. This component will do everything the original component could do plus more. The user should be familiar with the capabilities of the original TDBLookupCombo since this documentation only discusses the additional features of the new control. For more information about the DBLookupCombo, see Using the TDBLookupCombo Component. In addition to the capabilities offered by TDBLookupCombo, TDBLookupCombo plus offers the following features.

- The ability to sort the dropdown list.
- The ability to display the list either left or right justified and above or below the edit box.
- The ability to incrementally search the drop down list as the user types.
- The ability to add new records to the lookup table on the fly.

Users can update a field in the current record of a dataset by choosing a value from the drop-down list. Delphi populates the DBLookupComboPlus list values dynamically from a second dataset, known as the lookup dataset, at run time.

Connecting the DBLookupComboPlus
To specify a data source for the lookup dataset, set the LookupSource property.

The LookupSource must be a different data source than the DataSource. If you want to display values from a column in the same table as the first dataset, place a second data source and dataset component on the form and point them at the same data as the first data source and dataset.

To specify a field in the LookupSource dataset that Delphi links to the primary dataset, set the LookupField property.

The LookupField property column must contain the same values as the DataField property column, although the column names can differ.

Controlling the Display
To specify the columns that DBLookupComboPlus displays, set the LookupDisplay property.

If you do not set LookupDisplay, TDBLookupCombo displays the values found in the LookupField column. Use LookupDisplay to display a column other than the LookupField column, or to display multiple columns in the drop-down list. Use semicolons to separate multiple column names.

To specify the appearance of multiple columns, set the Options property.

loColLines, if True, separates the columns in the lookup list with lines.

loRowLines, if True, separates the rows in the lookup list with lines.

loTitles, if True, displays column names as titles above the columns in the lookup list.

To specify how the list justifies when it drops down, set the DropDownAlign Property to either **Left** or **Right**. To specify whether the list drops down or rises up, set the DropDownTop Property to either **Below** or **Above**.

Sorting the Dropdown List

To sort the DropDown List specify and index name in the LookupIndex property. This must be an already defined maintained index on the lookup table.

Setting up the Incremental Searches

Incremental searches of the information in the dropdown list is supported by the csIncSearch and csIncSrchEdit Style property. Set the Style property to **csIncSearch** for incremental searches with a read only edit box. Select the **csIncSrchEdit** Style when you want edit box to be read/write.

Adding Records To The Lookup Table on the Fly

The csDropDown and csIncSrchEdit Styles support the ability to enter new records into the lookup table during data entry into the main form. To accomplish this you must attach code to the onNewLookupRec event. See the example for the onNewLookupRec event for directions on how to do this.

OnAfterSearch_Event

Example

Applies to

TDBLookupComboPlus component

Declaration

property OnAfterSearch :   TNotifyEvent;

Description

This event is fired just after the incremental search occures. Use this event to set the SearchValue property back to its original value. OnBerforeSearch and   OnAfterSearch must always be used as a pair.

See   Also OnBeforeSearch

Example

Where OriginalSearchValue is a   private field defined in the form containg the TDBLookupComboPlus.

```
procedure TForm1.DBLookupComboPlus1AfterSearch(Sender: TObject);
begin
   DBLookupComboPlus1.SearchValue := OriginalSearchValue;
end;
```

See   Also OnBeforeSearch

OnBeforeSearch

Example

Applies to

TDBLookupComboPlus component

Declaration

property OnBeforeSearch :   TNotifyEvent;

Description

This event is fired just before the incremental search actually occures. Use this event to modify SearchValue temporarily . It SearchValue must then be set back to its original value by the OnAfterSearch. Always use these two events as a pair.

The primary reason for makeing these two events available is to give a way to do incremental searches on numeric values. See the example for a more complete explanation.

See Also OnAfterSearch_Event

Example

The example shows a code snipit on how you would install an incremental search for a TDBLookupCombo where the display value and lookup list is of numeric values.

The problem with numeric values used in a list goes like this. Lets say you have a customer number that is always 4 digits long but it is stored as an integer. The standard behavior for this component would go like this if the user were trying to find the customer number 1234.

User types 1, the search tries to find the value of 1 not 1000 which we want.

User types 2, the search tries to find the value of 12 not 1200 which we want.

User types 3 the search tries to find the value of 123 not 1230 which we want.

Finally the user types 4 and the correct value is selected. This is not a very useful incremental search.

In order to make this work as expected we need to convince the lookupcombo that it is actually searching for 1000 when 1 is pressed and 1200 when the 2 is pressed ... etc. The way to do this is to catch the value being pass to the search just prior to the search and right padding it with zeros. Heres the code to do this for the example described above.

```
procedure TForm1.DBLookupComboPlus1BeforeSearch(Sender: TObject);
var
  tmp : String;
begin
  { Store the real search value }
  OriginalSearchValue := DBLookupComboPlus1.SearchValue;
  { Create a modified search value }
  tmp := DBLookupComboPlus1.SearchValue;
  { special case where there is no search value }
  if Tmp = '' then
    Tmp := '0'
  else
  {right pad with 0's}
    While Length(Tmp) < 4 do
      tmp := Tmp + '0';
  DBLookupComboPlus1.SearchValue := Tmp;
end;
```

The OriginalSearchValue is a private field you declare in the form that contains the lookupcombo of type string.
VERY IMPORTANT - You must set the SearchValue property back to its original value in onAfterSearch

See Also OnAfterSearch_Event