



TtaDBMRO Component

[Properties](#) [Methods](#) [Events](#) [Tasks](#) [See also](#)

The TtaDBMRO component is a descendant of TDBGrid (or TwwDBGrid) that provides Multiple Record Object capability by accessing data in a database table or query and displaying it in format defined by a TPanel. This format is duplicated vertically so that the contents of the TPanel appear to be stacked one above the other.

TtaDBMRO inherits most of the properties of TDBGrid without modification expect for [DefaultDrawing](#), [Options](#), and [TabStop](#).

[RecordPanel](#) defines the layout of the record displayed. For the [selected record](#) or row, all editing occurs in the RecordPanel.

TtaDBMRO supports all standard Borland field data aware controls plus drawing tools such as TBevel, TLabel, and TGroupBox. Registered users have access to controls from [InfoPower](#), [TurboPower](#), and Out & About (see [Supported Controls](#)). Several hooks are provided for [supporting new controls](#).

Titles may be displayed by using the [TitlePanel](#) property. By using the [AutoTitleHeight](#) property, the application can control whether to force the height of titles to be the same as the height of the RecordPanel.

The ClientHeight of the TtaDBMRO can be set to an exact multiple of the RecordPanel.Height by using the [AutoHeight](#) property. The [AutoWidth](#) property is a closely related attribute.

There are several options available to differentiate the appearance of the selected record or row from the [nonselected record](#) or row. The [UseColor](#) and [UseFont](#) properties simply use the Color and Font properties of the TtaDBMRO to draw the text fields when displaying nonselected records. The [BackgroundColor](#) property changes the panel color of the nonselected records.

[Navigation](#) within the object requires that some code be added to the parent form and, optionally, to the first and last field in the RecordPanel.

For step by step instructions, see [How to Create a MRO](#).

If you are using the Trial Run version of TtaDBMRO, the component will only run while Delphi is running. To register your copy of TtaDBMRO, see [How to Order](#).

Installation

To install TtaDBMRO into the component palette:

1. Create a new directory (e.g. d:\TADB MRO)
2. UnZip the files in the new directory
3. (Trial run users only) If using Delphi 1.0, rename TADB MRO.D16 to TADB MRO.DCU. If using Delphi 2.0, rename TADB MRO.D32 to TADB MRO.DCU.
4. (Optional - registered users only) Using a text editor, modify TADB MRO.INC per the instructions contained within TADB MRO.INC. This should only be done by those wishing to use InfoPower's TwwDBGrid as the ancestor to TtaDBMRO (otherwise the ancestor is TDBGrid) or users wishing to enable Orpheus and/or Out & About support.
5. (Recommended) Backup COMPLIB.DCL
6. Start Delphi, select Options|Install Components (Delphi 1.0) or Components|Install (Delphi 2.0).
7. Click the Add button, then the Browse button and locate MROREG.PAS in your new directory
8. Select it
9. Press OK in the Install Components dialog and wait for the Library to rebuild
10. (Trial run users) Programs can now use TtaDBMRO while Delphi is running

TtaDBMRO is now available in the Data Controls palette.

To install the on-line help:

1. TADB MRO.HLP and TADB MRO.KWF should be in the same directory as TADB MRO.DCU
2. If Delphi is running, shut it down
3. (Recommended) Backup \DELPHI\BIN\DELPHI.HDX
4. Run \DELPHI\HELP\HELPINST
5. File|Open \DELPHI\BIN\DELPHI.HDX
6. If any existing KWF files are "not found", then add the appropriate search paths by selecting Options|Search Path
7. Select Keywords|Add File menu choice and select d:\TADB MRO\TADB MRO.KWF
8. File|Save
9. Exit the program
10. Check the WINHELP.INI file in the Windows directory and be sure that this entry is included:
taDBMRO.hlp=<fullpath> where <fullpath> indicates the location of the help file

The taDBMRO help files are now installed.

Selected Record

This is the current or focused record in the MRO. Only one record at a time can be the selected record.



Nonselected Record

All other visible records other than the selected record.

White Space

The difference between the height and width of the RecordPanel and the ClientHeight and ClientWidth of the MRO sometimes produces a gap between the bottom and/or right edges of the RecordPanel and the bottom and/or right edge of the MRO. Although not necessarily white, this gap is called white space. See figure in selected record. To manage white space, see AutoHeight and AutoWidth.

Demo Programs

All the demos assume that an alias DBDEMOS exists and that BIOLIFE.DB, CUSTOMER.DB, EMPLOYEE.DB, ORDERS.DB, and related files are contained within this alias. If this is not the case, this alias and/or table will have to be re-installed.

All projects/units ending in a numeral are designed to work with TtaDBMRO descending from TDBGrid (e.g. MROPROJ1.DPR). Those ending in a letter are designed to work with TtaDBMRO descending from TwwDBGrid (e.g. MROPROJA.DPR). Since this documentation will only refer to demo projects ending with a numeral, registered users using TwwDBGrid version should simply substitute the equivalent demo ending with a letter.

MROPROJ1.DPR demonstrates all the major features of TtaDBMRO. Click on the check boxes to toggle the various features.

MROPROJ2.DPR is an example of how to support a new control.

How to Create a MRO

1. Create a new project.
2. Add a TTable and TDataSource to the form and connect them as usual to a table of your choice.
3. Select a TPanel component and add it to the form.
4. Within Panel1, place all the elements of the record you wish displayed using any standard Borland field data aware control, TLabel, TBevel, TPanel, and/or TGroupBox.
5. Select the TtaDBMRO component from the Data Control palette and add it to the form.
6. Set the Width of taDBMRO1 to the same width of Panel1 (can be approximate).
7. Set the Height of taDBMRO1 to a multiple of the height of Panel1 (can be approximate). So if the height of the panel is 80, set the height of the taDBMRO to 240.
8. Set taDBMRO1.DataSource to DataSource1.
9. Set taDBMRO1.RecordPanel to Panel1.
10. Set Form1.KeyPreview to True.
11. Add the following OnKeyDown event to Form1, then run the project:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;  
                                Shift: TShiftState);  
begin  
    taDBMRO1.FormKeyDown(ActiveControl, Key, Shift)  
end;
```

How TtaDBMRO Works

TtaDBMRO descends from TDBGrid (or TwwDBGrid) and therefore relies on TDBGrid to manage the connection to the data aware controls. In terms of drawing the screen, TtaDBMRO ensures that the DataSource has one visible field, but instead of drawing just the text of this single field, it expands the size of the cell to fit the RecordPanel and then positions the RecordPanel over the cell. This is how the selected record or row is 'drawn'.

Nonselected records or rows are drawn differently. Instead of duplicating the RecordPanel (and the controls within the RecordPanel) for the remaining visible rows in the 'grid', TtaDBMRO creates a bitmap image of them. In this way, no additional Windows resources are consumed. However, in order to create this image, TtaDBMRO must know how to draw the image of each control within the RecordPanel. For a list of supported controls, see Supported Controls. Also see Controlling EXE File Size.

TtaDBMRO takes steps to assure that there is only one visible field in the DataSource. Nevertheless, the application should not change the Visible, DisplayWidth, or DisplayLabel property of any of the fields in the DataSource at run time.

In order for keystrokes such as Up, Down, PgUp, PgDn, Ctrl+Home, and Ctrl+End to be translated into the appropriate navigation sequences, some code has to be added to the form and, optionally, to the first and last field or control in the RecordPanel. See Navigation for a complete discussion on how this is implemented.

Supported Controls

TtaDBMRO supports all Borland field data aware controls and their descendants: TDBEdit, TDBText, TDBComboBox, TDBListBox, TDBLookupList, TDBLookupListBox, TDBLookupCombo, TDBLookupComboBox, TDBCheckBox, TDBRadioGroup, TDBMemo, and TDBImage. In addition, the following drawing tools are supported: TBevel, TLabel, TGroupBox, and TPanel. Registered users have access to TwwDBLookupCombo, TwwDBLookupComboDlg, TwwDBComboBox (InfoPower), TDBComboBoxPlus, TDBLookupComboPlus (Out & About), OvcDBSimpleField, OvcDBPictureField, and OvcDBNumericField (TurboPower's Orpheus) controls. To use controls that are not descendants of these controls, see [Supporting New Controls](#).

Trial run versions of many of these tools are available from:

| | CompuServe Library | Internet |
|-------------|---|--|
| TurboPower | pcvenb, Lib 6, orphtr.exe | http://tpower.com |
| InfoPower | Delphi, Lib 22, infotrl.exe winapi, Lib 5, infotrl.exe | http://woll2woll.com |
| Out & About | Delphi, Lib 6 & 22, dbplus1.zip & dbplus2.zip | http://www.computer-shopper.com/ magazine/delphi.htm |

TtaDBMRO also supports controls that are TDBCtrlGrid compatible. This support is limited to Delphi 2.0 and the UseColor and UseFont properties will have no affect on these controls.

BLOB Controls

Support for BLOB controls (TDBMemo and TDBImage) was added in version 1.10. There is one limitation to using BLOB controls: when inserting or appending a new record, all BLOB fields in the MRO become blank. After the record is posted, the BLOB fields display normally. This behavior is probably why Borland chose not to support these field types in TDBCtrGrid.

The following work around is recommended: place this code in the Table1 AfterInsert event:

```
procedure TForm1.Table1AfterInsert(DataSet: TDataSet);  
begin  
    DataSet.Post;  
    DataSet.Edit  
end;
```

Be aware that this work around has two side effects: first, since the record is posted, the record moves to its index position in the table, so you may want to set a temporary index value, if possible, to keep the record in the same relative position (usually at the end of the table).

Second, if the user moves off the record without entering any data, the record is not automatically canceled, so you will need to delete it. This can be overcome by adding this OnBeforePost event to Table1:

```
procedure TForm1.Table1BeforePost(DataSet: TDataSet);  
var i : Integer;  
begin  
    with DataSet do  
        if State = dsEdit then  
            begin  
                {* This example checks each field, which may not always *}  
                {* be appropriate *}  
                for i := 0 to FieldCount - 1 do  
                    if not Fields[i].IsNull then exit;  
                Delete;  
                Abort  
            end  
end;
```

Unfortunately, this event is only called if the record has been modified, so if the user accidentally inserts a new record and immediately moves off the record, then this code is not executed. Therefore, some code has to be added to Table1AfterInsert after DataSet.Edit to modify a field and then clear the field. This will force the Table1.Modified property to be True, and thus ensure that the above event is called. See example in MROPROJ1.

Navigation

TtaDBMRO provides two styles of keyboard navigation between the selected record and adjacent nonselected records. These styles can be used individually or in combination.

Conventional Grid Navigation

The first style is conventional grid navigation where Up, Down, PgUp, PgDn, Ctrl+Home, and Ctrl+End have their usual meaning. Since the RecordPanel always has the keyboard focus, a means has to be provided to intercept these keystrokes and translate them into appropriate navigation sequences. To do this, set the form's KeyPreview property to True and add the following code:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;  
                                Shift: TShiftState);  
begin  
    taDBMRO1.FormKeyDown(ActiveControl, Key, Shift)  
end;
```

If ActiveControl is part of the RecordPanel and Key/Shift are either Up, Down, PgUp, PgDn, Ctrl+Home, or Ctrl+End, then the routine will perform the navigation and set Key to zero. If the user is on the last record, dgEditing is enabled, and the table is not ReadOnly, the down arrow key will Append a new record.

If there are multiple TtaDBMROs in the form, then FormKeyDown will have to be called for each TtaDBMRO.

This form of navigation is best used when none of the controls in the RecordPanel use the Up or Down keys. Controls that use these keys, which include TDBComboBox, TDBListBox, TDBLookupList, TDBLookupCombo, and TDBRadioGroup, will not receive the Up and Down keystrokes if FormKeyDown is used, so you may want to screen for these controls before calling FormKeyDown. Alternatively, you may call FormKeyDownExt which automatically screens for these controls.

Tabbing Between Records

The second style is where tabbing from the last control in the RecordPanel advances to the next record and gives the first control in the RecordPanel the focus (and visa versa). In order to implement this behavior, four routines have to be added to the form.

In the first and the last controls in the RecordPanel, place the following code in the fields' OnExit routine:

```
taDBMRO1.FieldOnExit(ActiveControl, TestControl, GotoControl, GotoNext)
```

where: ActiveControl is the ActiveControl; TestControl is the next (or prior) control in natural tab order; GotoControl is the control to go to instead of the TestControl; and GotoNext is a Boolean which indicates the direction to move the RecordPanel.

If taDBMRO1 and the RecordPanel (and those controls contained within the RecordPanel) are the only controls in the form, then TestControl and GotoControl are the same.

If your form contains TGroupBoxes or TPanels (other than the RecordPanel or TitlePanel) , then TestControl *may be a TGroupBox or TPanel even though these objects aren't normally associated with getting the focus.*

The following code is from MROUNIT1.PAS (part of MROPROJ1.DPR):

```
procedure TForm1.DBEdit1Exit(Sender: TObject);  
begin  
    taDBMRO1.FieldOnExit(ActiveControl, UseFontCheckBox, DBEdit11, FALSE)  
end;
```

When DBEdit1 has the focus, a backtab would normally move the focus to UseFontCheckBox. Instead, this routine will move the focus to DBEdit11 (the last control in the RecordPanel) and move to the previous record (GotoNext is False).

```
procedure TForm1.DBEdit11Exit(Sender: TObject);  
begin  
    taDBMRO1.FieldOnExit(ActiveControl, IndicatorCheckBox, DBEdit1, TRUE)  
end;
```

When DBEdit11 has the focus, a tab would normally move the focus to IndicatorCheckBox. Instead, this routine will move the focus to DBEdit1 (the first control in the RecordPanel) and move to the next record (GotoNext is True).

Next, two routines have to be modified in the form. This code goes in the form's declaration:

```
private  
    procedure WMParentNotify(var Msg : TWMParentNotify);  
        message WM_PARENTNOTIFY;  
protected  
    procedure ActiveChanged; override;
```

And this code implements the routines:

```
procedure TForm1.WMParentNotify(var Msg : TWMParentNotify);  
begin  
    taDBMRO1.MonitorFocus(Msg.Event);  
    inherited  
end;  
  
procedure TForm1.ActiveChanged;  
begin  
    taDBMRO1.MonitorFocus(WM_SETFOCUS);  
    inherited ActiveChanged
```

end;

MonitorFocus simply looks for WM_LBUTTONDOWN events and ignores the subsequent focus change. Why? Because the focus changes due to mouse clicks cannot be confused for Tab or BackTab, and this code keeps track of which is which.

Cursor Navigation

Clicking on a nonselected record automatically gives that record the focus. Clicking on a control in a nonselected record makes that record the selected record and gives the control the focus. However, this does not move the caret into an edit field (in the case of a TDBEdit, for example) or, if a TDBLookupCombo's drop down arrow was clicked, does it drop down the list. A few more things are needed to do this.

In order to move the caret into the field, drop down a list, or click on a scroll bar, TtaDBMRO uses the original click to move to the selected record, find the control (if any), and saves the mouse coordinates. Since it is in the middle for a WM_LMOUSEDOWN and WM_LMOUSEUP sequence, it cannot initiate another mouse press. It has to wait until the original mouse sequence has completed.

The second 'mouse press' is initiated in the Application.OnIdle event. To do this, when mroMouseClicked is True in Flags, taDBMRO temporarily inserts itself into the OnIdle event loop, processes the second 'mouse press', and removes itself from the OnIdle event loop (restoring the previous OnIdle event, if any). There is nothing further the application needs to do.

If you want to explicitly control the behavior of the OnIdle event and if mroMouseClicked is False in Flags, then following code needs to be added to Form1's OnCreate event:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  Application.OnIdle := taDBMRO1.ApplicationIdle  
end;
```

If there is more than one TtaDBMRO in the application, then each instance of TtaDBMRO would have to be called.

Changing the Background Color for Nonselected Records

Using a different background color is one way to distinguish the nonselected records from the selected one. To use a different background color, simply set the BackgroundColor to the appropriate color.

If further customization is needed, use the OnDrawBackground event. The following example only changes the background color (a result more easily achieved as described above) and does not draw any of the elements of a TPanel (i.e. BevelInner, BevelOuter):

```
procedure TForm1.taDBMRO1DrawBackground(Sender: TObject;  
    Control: TControl; const CellRect: TRect);  
begin  
    with taDBMRO1.Canvas do  
        begin  
            Brush.Color := clGreen;  
            FillRect(CellRect)  
        end  
end;
```

Controlling the Colors and Fonts of Nonselected Records

By default, UseColor and UseFont are both True which forces the use of the Color and Font attributes when drawing nonselected records. Thus, the easiest way to display the nonselected records with a different field color and/or font is to change the Color and Font properties. These properties will be applied to each field in the nonselected records.

To use the same color(s) and font(s) in the nonselected record as the selected record, simply set UseColor and UseFont both to False. When UseColor is False, the Color of the control in the RecordPanel is used when drawing each field in the nonselected record. Similarly, if UseFont is False, then the Font from the control in the RecordPanel is used.

Controlling White Space

The difference between the height and width of the RecordPanel and the ClientHeight and ClientWidth of the MRO sometimes produces a gap between the bottom and /or right edges of the RecordPanel and the bottom and/or right edge of the MRO. Although not necessarily white, this gap is called white space.

There are three ways to control white space: First, simply leave the AutoHeight and AutoWidth properties set to their default settings of True. When set, these properties will automatically manage the size to the TtaDBMRO so that there will be no white space.

Second, you may adjust the size of the TtaDBMRO manually to eliminate white space. Note, however, that if you allow the user to toggle certain TtaDBMRO attributes such as dgTitles, dgRowLines, and dgIndicator, that these actions may introduce white space into the object (AutoHeight and AutoWidth can manage the toggling of these attributes).

Third, the color of the TtaDBMRO can be chosen to disguise the white space. If the color of the RecordPanel is clBtnFace, and the background of the nonselected records is clBtnFace (same as the RecordPanel which is the default), then simply set the color of the TtaDBMRO to clBtnFace.

Supporting New Controls

Before trying to support a new control, be sure to first check to see whether the control in question is a descendant of a supported control. If so, then nothing further needs to be done. If it is not a descendant of a supported control, then custom support will have to be provided.

The following example shows how to support TDBComboBox by using the OnDrawControl event. This control already has native TtaDBMRO support, but is used in the example below and in MROPROJ2.DPR because to use a non-standard control would assume that the non-standard control is installed on your computer.

```
procedure TForm1.taDBMRO1DrawControl(Sender: TObject;
    Control: TControl; const CellRect: TRect);
var S : string;
    Rect : TRect;
    DrawFont : TFont;
    Offset : Integer;
begin
  if Control is TDBComboBox then
    with TDBComboBox(Control) do
      begin
        if not Visible then exit;
        S := '';
        if DataSource <> nil then
          try
            S := DataSource.DataSet.FieldByName(DataField).DisplayText;
          except
            on EDatabaseError do
              else raise
            end;
          Rect := CalcRect(Control, CellRect);
          {$IFDEF WIN32}
          if NewStyleControls then InflateRect(Rect, -1, -1);
          {$ENDIF}
          DrawFont := taDBMRO1.FetchFont(Font);
          Offset := taDBMRO1.GetFontOffset(DrawFont);
          taDBMRO1.DrawString(taLeftJustify, taDBMRO1.FetchColor(Color),
              Enabled, DrawFont, Offset, Offset, Rect, S);
          taDBMRO1.DrawBorder(bsSingle, Ct13D, True, Rect)
        end
      else taDBMRO1.DrawControl(Control, CellRect)
    end;
end;
```

The parameters passed to the procedure are: Sender, the TtaDBMRO; Control, the control within the RecordPanel being drawn; and CellRect, the dimensions of the cell being drawn (this is the area occupied by the RecordPanel, not the Control within the RecordPanel).

The procedure is responsible for drawing for each control within the RecordPanel. So after the procedure has added support for the new control it should call DrawControl which will handle the drawing of all supported controls.

Controls added this way will not be visible in design mode.

If the new control itself contains controls (e.g. a TPanel may contain more than one TDBEdit), then the routine must call DrawControl for each child control. See the code used in DrawPanel.

See MROPROJ2.DPR. Note that the control will be visible in design mode because the control being supported in this example has native TtaDBMRO support.

InfoPower Support

By default, TtaDBMRO is a descendant of TDBGrid and installs itself in the 'Data Controls' palette. To make TtaDBMRO a descendant of TwwDBGrid, simply open the file TADBMO.INC and make the changes to the conditional compiler directives as described in TADBMO.INC. If you have already installed TtaDBMRO into the Delphi library, you will need to remove it (Options|Install Components, select Mroreg, and then press Remove). Now simply install (or reinstall) it, following the steps in described in the Installation section. TtaDBMRO will now be part of the InfoPower palette.

When InfoPower support is enabled, the DataSource and Options properties become their InfoPower equivalents, TwwDataSource and TwwDBGridOptions. Support will also be enabled for TwwDBLookupCombo and TwwDBLookupComboDlg (since TwwDBComboBox descends from TDBComboBox, this control is already supported).

Orpheus Support

To enable support for OvcDBSimpleField, OvcDBPictureField, and OvcDBNumericField controls, simply enable the appropriate UseOvcDBXxxxField compiler directive(s) in TADBMMRO.INC (instructions on how to do this are contained in the file TADBMMRO.INC).

By default, taDBMRO supports version 2.0x of Orpheus. If you wish to use version 1.0x, see the discussion below.

Be aware there can be key conflicts between FormKeyDown and OvcController1. If you find that the Up and Down keys not only move to the Prior/Next record but also change the focused field, then you will need to clear the command assignments for the Up and Down keys in the EntryCommand property of OvcController1.

Orpheus v1.0x Issues

In order to enable support for Orpheus version 1.0x, the compiler directive UseOrpheus2 must be disabled in TADBMMRO.INC.

There are four things to be aware of when using Orpheus 1.0x controls: First, the application cannot use the Tag field of any Orpheus control in the RecordPanel. TtaDBMRO uses the Tag field internally. Any attempt by the application to use the Tag field will result in a General Protection Fault.

Second, in design mode, the nonselected records will be displayed without using the PictureMask for formatting.

These restrictions are entirely due to the fact that the current version (1.02 as of this writing) of Orpheus provides no direct way to apply a PictureMask to a string (future versions may support this). Therefore, in order to duplicate the formatting behavior of an OvcDBXxxxField, TtaDBMRO has to create a non-visible OvcXxxxField for each OvcDBXxxxField (the Tag field points to the OvcXxxxField). Creating the OvcXxxxFields while in design mode, while possible, was not deemed worth the added complexity and was potentially confusing (the OvcXxxxFields are visible in design mode).

Third, Orpheus version 1.01 and earlier has a bug in OvcDBSimpleField that prevents the PictureMask from being applied to the field. So if the PictureMask is '!' (forces upper case), the contents of the field are 'Smith', an OvcDBSimpleField will display 'Smith' while the corresponding field in the nonselected record will display 'SMITH'. Orpheus version 1.02 fixes this problem.

Out & About Support

To enable support for Out & About's TDBComboBoxPlus and TDBLookupComboPlus components, follow the instructions in the file TADBMO.INC. If you are using TDBLookupComboPlus and TDBLookupList in the same application, be sure that the unit DBLookup appears after DBLUP2 in the USES clause.

TtaDBMRO 2.0 or later requires version 4.0 or later of TDBLookupComboPlus and version 2.0 or later of TDBComboBoxPlus.

ApplicationIdle procedure

Declaration

```
procedure ApplicationIdle(Sender: TObject; var Done: Boolean);
```

This procedure is automatically inserted into the Application.OnIdle event loop when mroMouseClicked is True in Flags. If mroMouseClicked is False, then place this procedure in the OnIdle event of Application in order to fully implement mouse support within nonselected records.

See also

Navigation

Example 1 -- One form, one TtaDBMRO, mroMouseClicked is False:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  Application.OnIdle := taDBMRO1.ApplicationIdle  
end;
```

Example 2 -- One form, two TtaDBMROs, mroMouseClicked is False in both TtaMROs:

```
private  
  { Private declarations }  
  procedure DoIdle(Sender: TObject; var Done : Boolean);  
  
procedure TForm1.DoIdle(Sender: TObject; var Done: Boolean);  
begin  
  taDBMRO1.ApplicationIdle(Sender, Done);  
  taDBMRO2.ApplicationIdle(Sender, Done)  
end;  
  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  Application.OnIdle := Form1.DoIdle  
end;
```

AutoHeight property

Declaration

property AutoHeight: Boolean;

Use this property to automatically adjust the ClientHeight of the TtaDBMRO to be a multiple of the RecordPanel's height. If False, there can be white space at the bottom edge of the TtaDBMRO. This property is ignored if Align is alClient, alLeft, or alRight.

The default value of AutoHeight is True.

See also

AutoWidth property

AutoTitleHeight property

Declaration

property AutoTitleHeight: Boolean;

When True, this property forces TitlePanel.Height to be equal to RecordPanel.Height. When False, TtaDBMRO uses the TitlePanel.Height without adjustment. Set to False if you wish to have a TitlePanel that is shorter (or taller) than that of the RecordPanel.

If set at run time, be sure to set AutoTitleHeight before setting the TitlePanel property.

If the application allows the user to enable and disable the title, and AutoTitleHeight is False, and TitlePanel.Height <> RecordPanel.Height, then the application will have to adjust the Height of the TtaDBMRO so that it does not shrink each time the title is enabled.

The default value of AutoTitleHeight is True.

AutoWidth property

Declaration

property AutoWidth: Boolean;

The Width of the RecordPanel is automatically adjusted to fit inside the ClientWidth of the TtaDBMRO. However, TDBGrid rounds the drawing area of a cell (and hence the RecordPanel) to multiples of 8 pixels (this is somewhat of an oversimplification). This means there can be a gap of up to 7 pixels on the right edge of the TtaDBMRO.

Use this property to automatically adjust the ClientWidth of the TtaDBMRO to eliminate this gap. If False, there can be white space at the right edge of the TtaDBMRO. This property is ignored if Align is alClient, alTop, or alBottom.

The default value of AutoWidth is True.

See also

AutoHeight property

BackgroundColor property

Declaration

property BackgroundColor: TColor;

This property controls the background color for the nonselected records.

The default value of BackgroundColor is clBtnFace.

See also

[Controlling the Background Color of Nonselected Records](#)

CalcRect function

Declaration

```
function CalcRect(Control: TControl; const CellRect: TRect): TRect;
```

A function usually called within a drawing routine to calculate the coordinates of the Control within the CellRect.

See also

[Supporting New Controls](#)

DefaultDrawing property

Declaration

property DefaultDrawing: Boolean;

This property is set to False by TtaDBMRO and should remain False in order for the object to behave properly.

The default value of DefaultDrawing is False.

Dither95 constant

This variable became private in v1.11.

DrawBorder procedure

Declaration

```
procedure DrawBorder(Style: BorderStyle; Ctl3D,TwoTone: Boolean;  
                    const CellRect: TRect); virtual;
```

A virtual procedure used within TtaDBMRO to draw the border of the Control in the nonselected records.

See also

Supporting New Controls,

DrawString procedure, OnDrawControl event

DrawControl procedure

Declaration

```
procedure DrawControl(Control: TControl; const CellRect: TRect); virtual;
```

Control is the TControl being drawn. CellRect is the area occupied by the [RecordPanel](#), not the Control within the RecordPanel. See [Supported Controls](#) for a list of valid controls.

See also

[Supporting New Controls](#), [DrawBorder](#) procedure, [OnDrawControl](#) event

DrawString procedure

Declaration

```
procedure DrawString(Alignment: TAlignment ; Color: TColor;  
    Enabled: Boolean; Font: TFont; OffsetX,OffsetY: Integer;  
    Rect: TRect; const Text: string); virtual;
```

A virtual procedure used within TtaDBMRO to draw the text of a control in the nonselected records.

See also

Supporting New Controls, DrawBorder procedure, GetFontOffset function, FetchColor function, FetchFont procedure, OnDrawControl event

GetFontOffset procedure

Declaration

procedure GetFontOffset(AFont: TFont): Integer;

Returns the Offset parameter used by DrawString to position text within the drawing region.

See also

[Supporting New Controls](#), [FetchFont](#) function

FetchColor function

Declaration

```
function FetchColor(Value: TColor): TColor;
```

If UseColor is False, returns Value, else returns TtaDBMRO.Color.

See also

FetchFont function, Supporting New Controls

FetchFont function

Declaration

```
function FetchFont(Value: TFont): TFont;
```

If UseFont is False, returns Value, else returns TtaDBMRO.Font.

See also

FetchColor function, Supporting New Controls

FieldOnExit procedure

Declaration

```
procedure FieldOnExit (ActiveControl, TestControl, GotoControl:  
                        TWinControl; GotoNext: Boolean);
```

This routine, when used in conjunction with [MonitorFocus](#), allows the application to implement tab oriented navigation between the last field in the [RecordPanel](#) and the first field in the next record (and visa versa). ActiveControl is the ActiveControl. TestControl is the next (prior) control in tab order. When ActiveControl is equal to TestControl, the routine forces the focus to GotoControl. GotoNext indicates the direction to move.

See also

[Navigation](#), [FormKeyDown](#) procedure, [MonitorFocus](#) procedure

Flags property

Declaration

property Flags: TMROFlags;

These are the possible values that can be included in the Flags set for the taDBMRO control:

| Value | Meaning |
|-----------------|---|
| mroAutoCursor | When True, as the user moves the cursor over a field in a nonselected record, the cursor will assume the shape corresponding to that field. When False, the cursor will not change shape when moving over fields in <u>nonselected records</u> . |
| mroMouseClicked | When True, taDBMRO automatically inserts itself into the Application.OnIdle event loop resulting in mouse clicks moving the caret into the field or dropping down lists when pressed over a button. |
| mroBtnWidth | When True and running under Win95, all drop down buttons (except in TDBComboBoxes) in the nonselected record will be 16 pixels wide. When False and running under Win95, the drop down buttons will have a width equal to GetSystemMetrics(SM_CXVSCROLL). |

The default value of Flags is [mroAutoCursor, mroMouseClicked, mroBtnWidth].

The purpose of mroBtnWidth is to compensate for the fact that most controls containing buttons do not use the SM_CXVSCROLL parameter in sizing the width of their buttons under Win95 (TDBComboBoxes do size their buttons correctly). The only reason to change mroBtnWidth to False would be if a vendor released a new version of their control that did use SM_CXVSCROLL to size their button under Win95.

FormKeyDown procedure

Declaration

```
procedure FormKeyDown(Sender: TObject; var Key: Word;  
                    Shift: TShiftState);
```

This routine should be placed in the form's OnKeyDown event in order to implement record navigation for the following keys: Up, Down, PgUp, PgDn, Ctrl+Home, and Ctrl+End. Be sure to set the form's KeyPreview property to True.

Orpheus users should be aware that some of these key assignments may conflict with OvcController1 key assignments. For more details, see [Orpheus Support](#).

See also

[FieldOnExit](#) procedure, [FormKeyDownExt](#) procedure, [MonitorFocus](#) procedure, [Navigation](#)

Example

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;  
                    Shift: TShiftState);
```

begin

```
    taDBMRO1.FormKeyDown(ActiveControl, Key, Shift)
```

end;

FormKeyDownExt procedure

Declaration

```
procedure FormKeyDownExt(Sender: TObject; var Key: Word;  
                        Shift: TShiftState);
```

Same as [FormKeyDown](#), but checks to see whether Sender is a control that normally processes navigation keys (Up, Down, PgUp, PgDn, Ctrl+Home, and Ctrl+End). If the control does not normally process these navigation keys, then FormKeyDown is called. If the control does normally process navigation keys (e.g. TDBLookupCombo), then the routine does nothing.

See also

[FieldOnExit](#) procedure, [MonitorFocus](#) procedure, [Navigation](#)

Example

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;  
                        Shift: TShiftState);
```

begin

```
    taDBMRO1.FormKeyDownExt(ActiveControl, Key, Shift)
```

end;

MonitorFocus procedure

Declaration

```
procedure MonitorFocus (WinMsg: Word);
```

This routine, when used in conjunction with [FieldOnExit](#), allows the application to implement tab oriented navigation between the last field in the [RecordPanel](#) and the first field in the next record (and visa versa). MonitorFocus must be called from the form's WMParentNotify and ActiveChanged procedures where WinMsg is Msg.Event (in WMParentNotify) and WM_SETFOCUS (in ActiveChanged).

See also

[Navigation](#)

Example

```
procedure TForm1.WMParentNotify (var Msg : TWMParentNotify);  
begin  
    taDBMRO1.MonitorFocus (Msg.Event);  
    inherited  
end;
```


OnDrawBackground event

Declaration

property OnDrawBackground: TMRODrawEvent;

Use this property to override the default drawing of the background region of the nonselected records.

See also

Changing the Background Color for Nonselected Records

Example

```
procedure TForm1.taDBMRO1DrawBackground(Sender: TObject;  
    Control: TControl; const CellRect: TRect);  
begin  
    with taDBMRO1.Canvas do  
        begin  
            Brush.Color := clGreen;  
            FillRect(CellRect)  
        end  
end;
```

OnDrawControl event

Declaration

property OnDrawControl: TMRODrawEvent;

Use this property to override the default drawing of controls within nonselected records. If set, this event is called rather than DrawControl.

See also

Supporting New Controls, DrawString procedure, DrawBorder procedure

OnPrepareLookup event

Declaration

property OnPrepareLookup: TMROPrepareEvent;

Use this property to override the default lookup behavior for TDBLookupCombo, TDBLookupComboPlus, and TwwDBLookupCombo. When TtaDBMRO performs a lookup, it only searches on one field. Since some of these controls provides the ability to lookup multiple fields, you can use this event to implement more sophisticated lookups.

Options property

Declaration

property Options: TDBGridOptions;

Same as TDBGrid.Options except that dgColLines and dgColumnResize are forced to be False and the default values are slightly different. When dgRowLines is enabled and AutoHeight is True or when dgIndicator is enabled and AutoWidth is True, the dimension(s) of TtaDBMRO will grow in order maintain the same number of visible rows.

The default value of Options is [dgEditing,dgConfirmDelete,dgCancelOnExit]

RecordPanel property

Declaration

property RecordPanel: TPanel;

This TPanel contains all the data aware controls that will be displayed in the TtaDBMRO. RecordPanel may contain any Borland, InfoPower, Orpheus, and/or Out & About Productions field data aware control. Hooks are provided to allow developers to support other controls.

See also

[Supported Controls](#), [Supporting New Controls](#), and [TitlePanel](#) property

TabStop property

Declaration

property TabStop: Boolean;

Same as TDBGrid.TabStop but with a default value of False. Generally, TtaDBMRO should not have the TabStop property set to True. Instead, tabbing should move into (or from) a control in the [RecordPanel](#).

The default value for TabStop is False.

See also

[Navigation](#)

TitlePanel property

Declaration

property TitlePanel: TPanel;

An optional property that when set, and when dgTitles is set in Options, will display the TitlePanel on the top edge of the TtaDBMRO. Fill the TitlePanel with TLabels (or any other object) to serve as titles for the individual fields in the RecordPanel. If the application allows the dgTitles property to be toggled, TtaDBMRO will toggle the TitlePanel.Visible property.

See also

AutoTitleHeight property

TMRODrawEvent type

Declaration

```
TMRODrawEvent = procedure (Sender: TObject; Control: TControl;  
                             const CellRect: TRect);
```

Event type for [OnDrawBackground](#) and [OnDrawControl](#).

See also

[Controlling the Colors and Fonts of Nonselected Records](#)

TMROFlags type

Declaration

```
TMROFlag = (mroAutoCursor, mroMouseClicked, mroBtnWidth);  
TMROFlags = set of TMROFlag;
```

The TMROFlags type is a set that defines the possible values of the Flags property.

TMROPrepareEvent type

Declaration

```
TMROPrepareEvent = procedure (Sender: TObject; Control: TControl;  
                                var S : string);
```

Event type for OnPrepareLookup. The event should return a string S to be displayed inside the control.

UseColor property

Declaration

property UseColor: Boolean;

When True, the Color property is used when drawing each field within the nonselected records. When False, the colors of the fields in the nonselected records will match those of the corresponding fields in the [RecordPanel](#).

The default value of UseColor is True.

See also

[UseFont](#) property, [Controlling the Colors and Fonts of Nonselected Records](#)

UseFont property

Declaration

property UseFont: Boolean;

When True, the Font property is used when drawing each field within the nonselected records. When False, the fonts of the fields in the nonselected records will match those of the corresponding fields in the RecordPanel.

The default value of UseFont is True.

See also

UseColor property, Controlling the Colors and Fonts of Nonselected Records

WinStyle variable

This variable was removed in version 2.0 and replaced by NewStyleControls.

Controlling EXE File Size (TADB.MRO.INC)

TADB.MRO.INC serves a dual purpose: the primary function of the file is to enable or disable support for third party tools, including whether TtaDB.MRO descends from a TDBGrid or InfoPower's TwwDBGrid.

The second function is closely related to the first, but with a subtle distinction: TADB.MRO.INC provides options for disabling standard controls (as well as third party controls).

Why would one want to disable support for a standard control? To reduce the size of the EXE file. When TtaDB.MRO draws nonselected records, it tests each control in the RecordPanel to see whether it is a supported control. This testing process requires that code from the supported control be included in the EXE file even if that type of control is not in the RecordPanel. The only way to eliminate the unnecessary code is via the compiler directives contained in TADB.MRO.INC.

To illustrate the impact this can have, compiling MROPROJ2 under Delphi 1.0 with support for all standard controls plus the three Orpheus and two Out & About controls results in an EXE file size of 845KB. Disabling support for all controls other than TDBEdit by changing the compiler directives in TADB.MRO.INC results in an EXE file size of 504KB, or 341KB difference.

The following strategy is recommended: enable all the compiler directives in TADB.MRO.INC that you are likely to use and compile COMPLIB.DCL and your applications during the development cycle using these settings. When your application is nearing the end of the development cycle, then disable support for controls not needed by the application.

Limitations

TtaDBMRO is a descendant of TDBGrid and therefore relies on TDBGrid to provide data awareness and to manage some aspects of painting the screen. Not surprisingly, there are a few conflicts between how a TDBGrid paints the screen and how a MRO should paint the screen. Since these conflicts cannot be resolved without making changes in TDBGrid itself, and since most of the conflicts can be avoided, these limitations will simply be spelled out and work arounds noted.

There are three main limitations: First, records managed by a TtaDBMRO must be vertically stacked and never side by side. This limitation may be removed in subsequent versions.

Second, TtaDBMRO is prone to produce white space on the bottom and right edges (see definition of white space below). This can be overcome by: adjusting the height and width of the TtaDBMRO manually; using the AutoHeight and AutoWidth properties to manage TtaDBMRO's dimensions; or by the appropriate selection of colors.

And third, keyboard navigation within the MRO is not automatic. Extra code has to be added to the form, and optionally to two controls, in order to implement keyboard navigation.

Troubleshooting

Changes to the RecordPanel in design mode not reflected in nonselected records

The nonselected records in the MRO just needs to be redrawn. Click on the scroll bar and the appearance of the nonselected records will be updated.

Control in RecordPanel is not duplicated in nonselected records

The control may not be supported by TtaDBMRO. See section [Supporting New Controls](#). Also check TADBMO.INC to see if support for the control may have been disabled.

Control duplicated in nonselected records in design mode, but not while executing, or visa versa

COMPLIB.DCL and the EXE file were compiled with different versions of TADBMO.INC. Make sure both COMPLIB.DCL and the EXE file were compiled with the same version.

Can't navigate to the next/prior displayed record via the keyboard

Make sure that Form1.KeyPreview is True and that taDBMRO1.[FormKeyDown](#) is in the OnKeyDown event of Form1. See [Navigation](#).

Tabbing between records isn't working

If you have followed the instructions described in [Tabbing Between Records](#) and you are still having a problem, the most likely cause is that some other control is temporarily getting the focus before TestControl does in the [FieldOnExit](#) routine. In most cases, this turns out to be a TGroupBox or TPanel (other than the [RecordPanel](#) or [TitlePanel](#)). Try replacing the current TestControl with the TGroupBox or TPanel.

Clicking on a region of the selected record that is not occupied by a control leaves no control with the focus

You can move the focus to a specific control by adding an OnClick event to the RecordPanel:

```
procedure TForm1.Panel1Click(Sender: TObject);  
begin  
    DBEdit1.SetFocus  
end;
```

taDBMRO does not install properly

The most common installation problem is one in which by adding 'd:\tadbmro' to the search path (see Options|Install Components|Search Path) causes the maximum length of the search path to be exceeded. The only solution is to shorten the length of the path names or combining multiple directories into a single directory.

Up and Down keys not working for TDBLookupCombo, TDBLookupList, ...

Check to see whether taDBMRO1.[FormKeyDown](#) is being called in Form1.FormKeyDown. If so, call taDBMRO1.[FormKeyDownExt](#).

Can TDBLookupComboPlus coexist with keyboard navigation?

Yes, if it is okay to drop down the lookup list via Alt+Down rather than Down. The trick here is to use taDBMRO1.[FormKeyDown](#) under all conditions except when the drop down list is visible:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;  
                               Shift: TShiftState);  
begin  
    if (not (ActiveControl is TDBLookupComboPlus)) or  
        (not (ActiveControl as TDBLookupComboPlus).ListVisible) then  
        taDBMRO1.FormKeyDown(ActiveControl, Key, Shift)  
end;
```


Horizontal scroll bar appears on the bottom of the TtaDBMRO

Check to make sure that only one field in the DataSource has the Visible property set to True and set the DisplayWidth of the visible field to a small value (e.g. 1) and DisplayLabel of the visible field is short or blank. Your program should not alter the Visible property of any of the fields or change the DisplayWidth or DisplayLabel of the visible field. If you did not set a field visible, TtaDBMRO will select one for you (usually the first field). Also check that the Height and Width of the TtaDBMRO is greater than RecordPanel.Height and RecordPanel.Width.

If this occurs when InfoPower support is enabled, the taDBMRO is on a tabbed notebook or page, and the DataSource.DataSet.Active property is being toggled, add the following code after the Active property has been set to True:

```
taDBMRO1.DataSource := wwDataSource1;  
taDBMRO1.DataSource.DataSet.First;
```

Sluggish TtaDBMRO redraw

Check to make sure that only one field in the DataSource has the Visible property set to True and that DefaultDrawing has not been set to True.

Excessive screen flicker

To minimize flicker, the Color and BackgroundColor of the taDBMRO should be the same as RecordPanel.Color. If RecordPanel.Color is different from BackgroundColor, then set the Color of the taDBMRO to the same color as BackgroundColor.

Clicking on a field in a nonselected record does not move the caret into the field

You need to add either set mroMouseClicked to True in the Flags property or set Application.OnIdle to taDBMRO1.ApplicationIdle. See Navigation.

Demo program will not compile

Check to see if TtaDBMRO has been installed properly. If not, rebuild your library. Each demo has two versions: one for TtaDBMRO descending from TDBGrid and one for TtaDBMRO descending from TwwDBGrid. Make sure you are using the correct one. See instructions under Demo Programs.

Demo program doesn't duplicate fields in nonselected record

Check to see if any of the compiler directives in TADBMMRO.INC have been disabled for standard Borland controls. If so, enable them. On the '3d Party' page of MROPROJ1, only those controls that are both installed and enabled will appear.

Application GPF while using Orpheus 1.0x data aware controls

Check the Tag field of all Orpheus data aware controls in the RecordPanel and make sure it is zero. Setting the Tag field to a non zero value in design mode will cause a GPF in complib.dcl, while doing so at run time will cause a GPF in the application.

Error 15: File not found (xxxxx.DCU)

Check to make sure the compiler directives in TADBMMRO.INC are set correctly. Enabling a compiler option means that the related component(s) must already be installed. If they are not, the installation or rebuild of TtaDBMRO will fail and generate the above error message. The offending compiler directive can usually be found in the Uses clause near 'xxxxx'.

How to Order

To receive a registered version of TtaDBMRO, which includes all source code, support for TwwDBGrid and TDBLookupComboPlus, technical support, along with free updates of version 2.x, just send \$25 U.S. with the order form that appears at the end of this document. Or you may email your name, address, MasterCard or Visa number, and expiration date to Tamarack Associates at 72365.46@compuserve.com. Sales tax will be added to California orders. Delivery is free via CompuServe or Internet, \$5 in North America (Canada, Mexico, & U.S.), \$10 outside of North America. Please specify 3.5" or 5.25" diskettes.

TtaDBMRO is also available through CompuServe SWREG for \$29.95 (CompuServe charges Tamarack Associates a 15% handling fee). The SWREG registration ID is 8213.

Please read the Purchase Agreement before registering.

Version History

The latest version of TtaDBMRO can always be found on CompuServe in the Delphi and BDelphi forums, Lib 22 (3d Party Products), in MRO.ZIP.

- 05/06/96 Version 2.00 Delphi 2.0 support added
 - TDBLookupComboBox support added
 - TDBLookupListBox support added
 - DrawText procedure renamed DrawString
 - TwoTone parameter added to DrawBorder procedure
 - Ctrl+Del deletes the current record
 - WinStyle variable replaced with NewStyleControls

- 02/23/96 Version 1.12 taDBMRO1.BorderStyle = bsNone problem fixed
 - RecordPanel.OnClick event properly recognized
 - FormKeyDown now works with non-TTable DataSets
 - Improved error handling
 - Compatible with TDBLookupComboPlus 4.0
 - Compatible with TDBComboBoxPlus 2.0

- 01/03/96 Version 1.11 Support added for TwwDBLookupComboDlg
 - Support for DataType <> ftMemo added for TDBMemo controls
 - System color changes handled properly
 - Dither95 constant is no longer public
 - Win95 scroll bar colors always drawn properly
 - TDBLookupComboPlus works properly when LookupIndex set
 - Improved appearance of disabled controls in obscure color combinations
 - Flags property added
 - Width of drop down buttons under Win95 corrected

- 12/04/95 Version 1.10 TDBCheckBox, TDBRadioGroup, TDBListBox, TDBLookupList, TDBMemo, and TDBImage support added
 - Full cursor support for nonselected records added
 - WinStyle variable and TWinStyle type added
 - Dither95 typed constant added
 - FormKeyDown now uses Append rather than Insert
 - FormKeyDownExt procedure added
 - Ctl3D parameter added to DrawBorder procedure
 - Enabled parameter added to DrawText procedure
 - DrawText Offset parameter changed to OffsetX & OffsetY
 - OnPrepareLookup event & TMROPrepareEvent type added
 - Ctl3D & Enabled properties correctly handled
 - TDBLookupCombo now handles DisplayField
 - Underscores for accelerators now drawn (e.g. &File is File)
 - TPanel.BorderStyle = bsSingle now drawn correctly
 - Design mode GPF when Record/TitlePanel deleted fixed
 - Field's Visible property no longer has to be set in design mode
 - Additional compiler directives added to TADBMMRO.INC
 - Losing character when inserting very first record fixed
 - Moving off inserted records that are unmodified handled better
 - BackgroundColor, broken in 1.01, fixed
 - README.1xx file added to distribution list

- 11/12/95 Version 1.01 TGroupBox, Orpheus, and TDBComboBoxPlus support added
 - Setting RecordPanel to NIL no longer causes GPF
 - Flicker after moving out of an edited field eliminated
 - Clicking problem on TDBComboBox fixed

TPanel (other than RecordPanel) now respects UseFont & UseColor
STATES.DB and STATES.PX no longer needed for MROPROJ2
Minor adjustments to height calculations
Minor adjustment to positioning of right justified text
TADBMRO.INT added to trial run version

10/31/95 Version 1.00 First release

Purchase Agreement

Terms of License Agreement

The TtaDBMRO programs and documentation are the property of Tamarack Associates and are protected by United States Copyright Law, Title 17 U.S. Code, are licensed for use by one person only on as many computers as that person uses.

Where a group of programmers are working together on a project that makes use of TtaDBMRO, we expect that a copy of the software and documentation will be purchased for each member of the group. Contact Tamarack Associates for volume discounts.

You may duplicate the TtaDBMRO programs and documentation files for backup use only.

You may distribute without further licenses or run time fees applications that make use of TtaDBMRO. You may not distribute or duplicate any documentation, source code, or DCU files other than described above.

Limited Warranty

TAMARACK ASSOCIATES MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT WILL TAMARACK ASSOCIATES BE LIABLE TO YOU OR ANY THIRD PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE THE PROGRAM OR MANUAL.

By using this product, you agree to this. If you do not agree, immediately return this product for refund.

Development Environment

TtaDBMRO was developed with Delphi 1.02 and 2.0 running under WFWG 3.11, Win95, and NT 3.51 with 16MB of RAM using Paradox tables. Orpheus 2.0, InfoPower 1.2, DBPlus1 2.1, DBPlus2 4.1.

Trademarks

Borland and Paradox are trademarks of Borland International.

Orpheus is a trademark of TurboPower Software.

InfoPower is a trademark of Woll2Woll Software.

Technical Support

Questions, bug reports and suggestions may be directed to:

Tamarack Associates
CompuServe 72365,46
Internet 72365.46@compuserve.com
(415) 322-2827 (Voice & Fax)

Please clearly state what compiler options have been set in TADBMRO.INC.

Files

Trial run version includes:

| | |
|--------------|--|
| MROREG.PAS | Source code for registering taDBMRO |
| MROPROJ1.DPR | Project file for main demonstration program |
| MROUNIT1.DFM | Form file for main demonstration program |
| MROUNIT1.PAS | Source code for main demonstration program |
| MROPROJ2.DPR | Project file for second demo |
| MROUNIT2.DFM | Form file for second demo |
| MROUNIT2.PAS | Source code for second demo |
| README.TXT | Brief installation instructions |
| README.2xx | Brief description of version changes |
| TADBMR0.D16 | Delphi 16 bit DCU file (Trial Run only) |
| TADBMR0.D32 | Delphi 32 bit DCU file (Trial Run only) |
| TADBMR0.HLP | Help file |
| TADBMR0.INC | Contains conditional compiler directives |
| TADBMR0.INT | TADBMR0.PAS interface section (Trial Run only) |
| TADBMR0.KWF | Help keyword file |
| TADBMR0.RES | Resource file |
| TADBMR0.WRI | This file |

Registered version includes these additional files:

| | |
|--------------|---|
| TADBMR0.PAS | Source code |
| MROPROJA.DPR | Same as MROPROJ1, but with TwwDBGrid ancestor |
| MROUNITA.DFM | Same as MROUNIT1, but with TwwDBGrid ancestor |
| MROUNITA.PAS | Same as MROUNIT1, but with TwwDBGrid ancestor |
| MROPROJB.DPR | Same as MROPROJ2, but with TwwDBGrid ancestor |
| MROUNITB.DFM | Same as MROUNIT2, but with TwwDBGrid ancestor |
| MROUNITB.PAS | Same as MROUNIT2, but with TwwDBGrid ancestor |

Order Form

TtaDBMRO 2.x

Tamarack Associates

868 Lincoln Avenue

Palo Alto, CA 94301 USA

415-322-2827 (Voice & Fax*)

72365.46@compuserve.com

Name _____
Company _____
Address _____
City _____
State _____
Country _____
Zip/Postal Code _____
Email _____
Phone _____

Credit Card MasterCard Visa

Card Number _____

Expiration Date _____

Number of copies _____ 3.5" _____ 5.25" _____

Price per copy \$25 U.S.

Subtotal _____

Sales tax _____ (California residents only)

Shipping & handling _____ (see below)

Total _____

Shipping & handling: CIS/Internet - none; North America - \$5; outside North America - \$10.

TtaDBMRO may also be registered through CompuServe SWREG 8213 for \$29.95.

Please read [Purchase Agreement](#) before ordering.

*The fax machine can take as long as 45 seconds to answer. Set your fax accordingly.

Properties

AutoHeight

AutoTitleHeight

AutoWidth

BackgroundColor

DefaultDrawing

Flags

Options

RecordPanel

TabStop

TitlePanel

UseColor

UseFont

Methods

ApplicationIdle

CalcRect

DrawBorder

DrawControl

DrawString

GetFontOffset

FetchColor

FetchFont

FieldOnExit

FormKeyDown

FormKeyDownExt

MonitorFocus

Events

OnDrawBackground

OnDrawControl

OnPrepareLookup

Tasks

[Changing the Background Color for Nonselected Records](#)

[Controlling the Colors and Fonts of Nonselected Records](#)

[Controlling White Space](#)

[How to Create a MRO](#)

[InfoPower Support](#)

[Installation](#)

[Navigation](#)

[Orpheus Support](#)

[Out & About Support](#)

[Supported Controls](#)

[Supporting New Controls](#)

See Also

[BLOB Controls](#)

[Controlling EXE File Size](#)

[Demo Programs](#)

[Files](#)

[How taDBMRO Works](#)

[How to Order](#)

[Purchase Agreement](#)

[Troubleshooting](#)

[Version History](#)

