Getting started with Lazarus:

Control Basics

Michaël Van Canneyt

Abstract

In this article about lazarus GUI programming, an overview of basic controls is presented. The basic properties common to all controls are examined, and techniques to modify them are presented as well. In particular, sizing and positioning are investigated in depth.

1 Introduction

In the previous article about Lazarus GUI programming, the basis of all GUI programs - the form - was discussed. In this article, some of the elements that can be used on the form (controls) will be examined.

Controls make up the functionality of a form. They are everything that is visible on a form: buttons, edit controls, lists, checkboxes, radio buttons, menus, treeviews and listviews, and many more things. A more complete overview of available controls will be presented below.

As explained in the previous article, there are 2 kinds of controls: Controls that have their own window handle associated with them, or the ones that do not have an own window handle associated with them. (a Window handle is a reference to an underlying GUI object). The basic difference between the two is that the former

- has a canvas (a drawing area) associated with it
- can receive focus (and hence keyboard input)
- can have other controls positioned on it.

It is represented by the TWinControl class. The latter

- has no canvas: it must draw itself on its parent (windowed) control.
- can not receive focus
- cannot contain other controls.

It is represented by the TControl class.

The controls obviously have a lot in common:

- They can draw themselves
- They have a parent control
- They have a size and a position
- They respond to mouse events (including drag and drop)
- They can have a help context associated to them, as well as tooltips.

For this reason, TWinControl descends from TControl.

In practical use, it hardly matters whether a control is a TControl or a TwinControl: Their use is transparent.

2 Basic Controls

The following is an incomplete list of basic controls which can be found on the 'Standard' tab of the component palette:

TButton is a simple button with a caption text. It can be pressed by the user when he wants to perform an

action.

TLabel simply displays a text. (the caption). The text can be single-line, or multi-line. One of the letters in the caption canfunction as a hotkey, and pressing this hotkey will set focus on an assocuated TWinControl (specified in the FocusControl). The key which should act as a hotkey should be marked with an Ampersand, and it will be displayed underlined.

TEdit Can be used to allow the user to enter a single line of text. The text can be limited in length, and can be masked with a password character (so the text is not shown, the password character is shown instead)

TMemo Is for allowing the user to enter multiple lines of text. The memo will automatically display scrollbars if need be. The text is displayed in a uniform font: no additional formatting is possible.

TToggleBox is similar to a Tbutton, but once clicked, it remains depressed, until the user clicks it again.

TCheckBox displays a box with a checkmark, accompagnied by some small text (the caption). The user can toggle the checkmark by clicking the box. One can have several checkboxes next to each other: they can be checked or unchecked at will. It is most useful to let a user enable or disable some options.

TRadioButton is similar to a TcheckBox, but differs in that if several radiobuttons appear on a control, only one of the radio buttons can be checked at a time: checking one radio button, will uncheck the others. This is useful for letting the user select one of mutually exclusive options.

TListBox displays a number of strings, and allows the user to select one or more of the strings. The strings can be drawn in a variety of styles (depending on the style of the listbox).

TComboBox allows the user to enter a text, just like a TEdit, but offers a list of possible texts. It is possible to limit the possible entries to one of the pre-defined items, or to allow the user to enter a text which is not yet in the list. The list with items is presented if the user clicks a 'dropdown' button, but can be made visible permanently.

TScrollBar is a bar with small handle (the 'thumb') which can be dragged to a certain position within the bar: this position can then be used to position other controls.

TGroupBox A groupbox can be used to group a set of controls: it draws a bevel around the controls and puts a caption in the bevel. This is a purelycosmetic control.

TRadioGroup A radiogroup is actually a combination of agroupbox and a set of radiobuttons: For each line in the Items property, a readiobutton will bedisplayed in the groupbox. The ItemIndex property determines which of the radiobuttons is checked.

TCheckGroup Similar to the radiogroup, the checkgroup displays a series of checkboxes. The Checked array determines which of the items is checked.

TPanel is similar to a label, but it draws a border around it, and can contain other controls. It can be used to group other controls, and is very handy to force certain layouts.It's also often used to descend drawing controls from, since it has a canvas to draw on.

Of all these controls, Tlabel is the only one which is not a TwinControl descendent. The controls ae demonstrated in the controlsdemo example program that comes on the CD privided with this issue. Part of it is visible in the following figure:

3 Positioning and size

Layout is an important issue when designing a GUI program. There are many things to consider when designing the layout of an application:

- Screen resolutions and font sizes may vary.
- Themes may change fonts and button sizes.
- The user may decide to resize the window (the form).
- In an internationalized application, the displayed texts may vary in size, so some controls may need to be shifted or resized to accomodate the new text's size.

Any of these necessitates resing and moving the controls on the form.

Obviously, it is possible to do all this resizing and positioning manually. This can be a daunting task, which is fortunately taken care of by the LCL, provided some properties are set correctly.

Some widget sets take care of sizing and positioning issues by using layouting objects: The GTK and Qt widget sets work like this. MS-Windows provides no layouts, but uses a fixed position and size. The Lazarus Component Library (LCL), modeled after Delphi's VCL, also uses a fixed position and size approach: By default, a control does not change position or size. But this behaviour can be changed using a variety of properties.

The main published properties that determine the position and size are the following:

Top,Left. These properties determine the position of the control. They are always relative to the parent controls client area top-left corner, and are measured in pixels.

Width,Height These properties determine the size of the control.They are in pixels. They must be positive,

and can be larger than the boundaries of the parent control allow for. What happens if such values are set, depends on the parent control.

Align This property can be used to align the control along one of the borders of the parent control. The default value alNone disables this feature.

Anchors This property can be used to align some of the borders of the control with the borders of the parent control (by default) or align it with the borders of any other control on the form. When the parent control is resized, or the linked control is moved or resized, the anchors will cause the current control to be moved or resized as well.

Autosize This property tells the control to resize itself so it's contents fit in the current size.

OnResize This event can be used to take action when the control is resized: this can be used to do manual resizingin case the other properties are not flexible enough to do the resizing. For instance in the case of a grid, this event can be used to rescale the columns so that thay always take up the full width of the grid.

BorderSpacing is the amount of space(in pixels) to leave between the control and the other controls it is anchored to.

Constraints determines the minimum and maximum sizea control may have when resizing.

The above properties are published, and can be set in the Object Inspector in the Lazarus IDE. The following public properties also can be used:

BoundsRect is a Trect record which defines the contours of the control. It can be used to set the top,left, widt,height in one call, instead of 4 consecutive calls..

ClientRect is the area available forchild controls: it does not include the border of the control. For a form with a menu, the clientrect starts just below the menu.

AnchorSides These properties determine the controls that the current control is anchored to.

4 Manual Resizing

Doing manual resizing is a tedious task, but sometimes there is no other way. Manual resizing can be demonstrated easily using a simple form containing a memo, which should always keep it's borders at the same distance of the borders of the form (i.e. It grows and shrinks with the form). The button should always stay in the lower-right corner of the form, and should keep it's size. It should look as follows:

To do this manually, it's necesary to determine the offsets for the button position relative to the lower and

right borders of the form, and the offsets of the memo right and lower sides relative to the form's lower right border. These distances are determined in the OnCreate event of the form:

```
procedure TMainForm.FormCreate(Sender: Tobject);
begin
  MHDiff:=ClientHeight-MSize.Height;
  MWDiff:=ClientWidth-MSize.Width;
  BTOff:=ClientHeight-BCLose.Top;
  BLOff:=ClientWidth-BClose.Left;
end;
```

The button is called Bclose, the memo is callse Msize. The ClientWidth and ClientHeight are the width and height of the form, available to controls within the form.. The 4 variables MHDiff,MWDiff,BTOff and BLOff can now be used in the OnResize event of the form:

```
procedure TMainForm.FormResize(Sender: Tobject);
begin
  Msize.Height:=ClientHeight-MHDiff;
  Msize.Width:=ClientWidth-MWDiff;
  Bclose.Top:=ClientHeight-BTOff;
  Bclose.Left:=ClientWidth-BLoff;
end;
```

As can be seen, it's not so hard to do this. Obviously, if there are a lot of controls on the form, the code will get more complicated and tedious to maintain. Fortunately, there are other ways of doing this.

5 Sizing using Align

The Align property can be used as well for resizing. It's limited in its scope, but can be used in many cases, and complex layouting can still be achieved with this property. The Align property can have the following values:

alNone no alignment is done. The control keeps it's position.

alTop the control is at all times glued to the top edge of the parent component: This means that it keeps its height and the width equals the width of the parent control.

alBottom the control is at all times glued to the bottom edge of the parent component. This means that it keeps its height and the width equals the width of the parent control.

alLeft the control is at all times glued to the left edge of the parent control. The width property is kept constant, and the height equals the height of the parent control.

alRight the control is at all times glued to the left edge of the parent control. The width property is kept constant, and the height equals the height of the parent control.

alClient the control tries to take as much space as available.

alCustom Unused at this moment.

Note that several controls can have e.g. their align property set to alTop. In this case the first control is glued to the edge of the parent control, the second is glued to the first control's bottom: they are stacked to the top edge of the parent control.

The alignsize project which comes on the CD accompagnying this issue, shows how to layout the form with the memo and button:

- Add a panel PButtons, set align to alBottom. Set bevels to none.
- Add on this panel, drop a second panel (PClose) set align to alRight and set bevels to none.
- On the PClose panel, drop the BClose button.
- Add a Msize memo to the form, with the align property set to alClient, and borderspacing.around set to 8.
- Resize Pclose and Bclose to the button is right-aligned with the memo.

After this is done, when resized, the form will behave identically to the form where layouting is done

manually. Layouting with this technique is fast and easy to do, but has also some drawbacks: Tpanel is a windowed control: as such it takes extra resources, and since it can receive focus, it disturbs the tab order of the other controls, so it should be used sparingly. Luckily, there are other techniques to make the form behave correct.

6 Sizing with anchors

Anchors are a powerful mechanism to resize controls automatically. On top of the anchors present in Delphi's VCL, Lazarus adds some additional mechanism to the anchors: It can anchor controls not only to their parent control, but also to neighbouring controls.

Anchoring is done through the Anchors property, which is a set of the following values:

akTop The control's top edge remains at the same distance of the parent control's top edge.

akLeft The control's left edge remains at the same distance of the parent control's left edge.

akRight The control's right edge remains at the same distance of the parent control's right edge. If needed, the control is resized horizontally to achieve this.

akBottom the control's bottom edge remains at the same distance of the parent control's bottom edge. If needed, the control is resized vertically to achieve this.

The default set is [akTop,akLeft], this simply means that the control keeps its current position and size. Setting a TEdit's anchors property to [akTop,akLeft,akRight] will make it keep it's position, but it will grow and shrink so it has always the same distance to the right edge of the form it is put on.

The example with the memo and button can be reworked using simply the anchors properties of the controls: it is sufficient to set the memo's anchors to [akTop,akLeft,akRight,akBottom] this ensures that all edges of the memo keep the same distance to the corresponding edges of the form. The button's anchors property is set to [akRight,akBottom], this makes sure it remains at the same position relative to the bottom-right corner of the form. The project 'anchorsize' demonstrates this.

But Lazarus offers more possibilities with anchors: It is possible to anchor controls to each other, specifying a distance between the controls. This is handy for more complex layouts. Imagine an edit control with a label in front of it. The label may change size as it's caption changes, for example when it is translated. In that case, the edit control should be repositioned, so it keeps the same distance to the label. This is demonstrated in the 'anchors' project: it contains a label, and 2 edit, controls, one of which is at the same top position as the label, as can be seen in the following picture:

The text in the second edit is copied in the OnChange event to the label caption. The first edit's anchors property can be edited using the property editor of the 'Anchors' property. It looks as follows:

As can be seen in the picture, the top, left and right anchors have been set. The top and right anchors have no sibling set. This means that they are anchored to the parent control, in this case the form. The left anchor has the sibling control set to the LEAnchored label. The middle speedbutton is depressed, which means that the edit control is anchored to the right side of the label. It could also have been anchored to it's left side, or to it's center.

When the dialog is closed, and the project is run, the effect of these anchor settings can be observed by typing a text in the second edit bow: the label's caption will be changed. Since the label's autosize property is set to true, the label will change it's size, and the first edit control is moved to it keeps the same distance from the label. The right edge of the edit remains at a constant distance from the form's border.

It should be obvious that this mechanism allows for some very complex layouting to be set up: most, if not all situations can be catered for with this mechanism.

Besides the 'Anchors' property, the dialog also sets the 'BorderSpacing' property. This property contains the distances relative to the other controls, plus an overall distance to be kept. Additionally, the 'Anchorsides' property is set: this property is not published, and cannot be manipulated in the Object Inspector: the anchorsides maintains the links to the neighbouring controls (the siblings, in the above figure) and the anchor 'sides' (represented by the 3 buttons).

The BorderSpacing and the AnchorSides properties are quite complex, discussing them completely falls outside the scope of this article.

7 Layouting using ChildSizing

Although the LCL uses primarily a fixed-position layouting strategy, it does offer a way to arrange controls in a grid-like fashion, keeping the controls in the cells of a grid which sizes as the control is sized. This is accomplished using the ChildSizing poperty of some controls. This property is of type TControlChildSizing, a TPersistent descendent, which has 2 main properties:

Layout. This controls how the controls are positioned in the grid. It can take the following values:

- **cclNone** means nothing is done: the controls are left on their designed location. This is the default, and amounts to no sizing at all.
- **cclLeftToRightThenTopToBottom** means that the controls are stacked in horizontal lines, starting at the left, filling the lines, and then continuing on the next line once the line is full.
- **cclTopToBottomThenLeftToRight** does the same, only the lines run from top to bottom.

ControlsPerLine is the number of controls that are stacked on a line.

The controls are stacked with space between them. The amount of space between the controls is controlled by the following 4 integer properties:

LeftRightSpacing Amount of space between the first control of a line and the left border

TopBottomSpacing Amount of space between the first control and the top border.

HorizontalSpacing Amount of horizontal space between the controls.

VerticalSpacing Amount of vertical space between the controls.

When the control is resized, it can be told to resize or reposition the controls it contains. The behaviour is controlled by the following 4 properties:

EnlargeHorizontal What to do in the horizontal direction when the control is enlarged.

EnlargeVertical What to do in the vertical direction when the control is enlarged.

ShrinkHorizontal What to do in the horizontal direction, when the controls becomes too small to contain it's contents.

ShrinkVertical What to do in the horizontal direction, when the controls becomes too small to contain it's contents.

These 4 properties are of the same type: TChildControlResizeStyle. This enumeration type has the following 4 values:

crsAnchorAligning this just keeps the position and sizes, as in Delphi.

crsScaleChilds this will scale child controls, and keeps the amount of space between the childs fixed.

crsHomogenousChildGrowth this will enlarge the child controls equally: each child gets an equal amount of pixels added to it's dimensions.

crsHomogenousSpaceGrowth this will enlarge the space between child controls equally: each spaced gets an equal amount of pixels added.

The childsizing project demonstrates these properties: it places a number of buttons on a panel (PGrid). A second panel contains a series of controls to set the ChildSizing properties of the PGrid panel. By setting the properties, the effect on the control stacking and resizing can be observed. The code of this project is straightforward: it's simply the setting of the various properties. The following figure shows what it looks like:

Note that this mechanism does not make use of any underlying widget set properties: the resizing and

repositioning is done completelt by the LCL, and hence is available on all supported platforms.

8 Conclusion

In this article, it was argued that an important aspect of GUI design is the layout of the forms: the look and feel is determined in a large part by how an application behaves if it (or parts of it) is rescaled or resized. The LCL offers many mechanisms to resize an application automatically: Even very complex layouts can be managed without requiring any code: Indeed, various solutions for a given problem can be invented. For the cases where the default mechanisms are not sufficient, the event mechanism offers an event to do the necessary resizing manually.