

Developing with FireMonkey components

Purpose: To understand how to create custom styles with FireMonkey

Start a new Blank Multi-Device Application.

Save in a separate folder `CustomStyleTest` as

`CustomStyleTestProject.dproj`

and `CustomStyleTestUnit.pas`.

Add a **TPanel** component, `Panel1`, to the form. Adjust size of form so that it is just greater than `Panel1` as shown in

Figure 42.1. Select **Windows 64-bit** and right click `Panel1`

on the form and select `Edit Custom Style` to bring

up the **Style Designer** - *Figure 42.4*. As **TPanel** inherits

from **TRectangle**, the **Style Designer** shows a **TRectangle**

control. This control has a **Fill** property. Change the

Fill|Color to **Chartreuse** and change the **Name** field to

`PanelStyleChartreuse`. The **Platform** field in the **Style**

Designer - *Figure 42.4* - shows **Windows 10 Desktop** so this

custom style will apply to **Windows 10 Desktop** applications

only. Close the **Style Designer** by clicking the cross in the

tab and click **Yes** when prompted to apply

changes.

If this is the first time that changes are

applied, Delphi adds a **TStyleBook**

component, `TStyleBook1`, to the form as

shown in *Figure 42.2*. The **TPanel** control's

colour is now **Chartreuse**.

The available styles are revealed by expanding

Styles in the **Structure** window, - *Figure 42.3*.

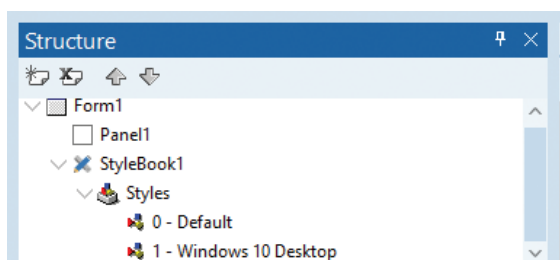


Fig. 42.3 Structure window

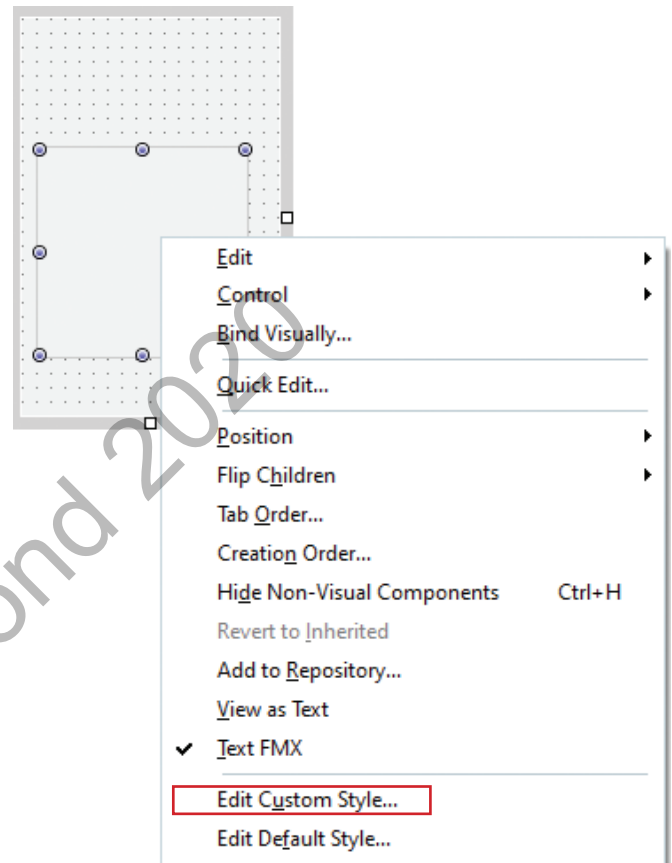


Fig. 42.1 CustomStyleTestUnit form, Panel1 right clicked

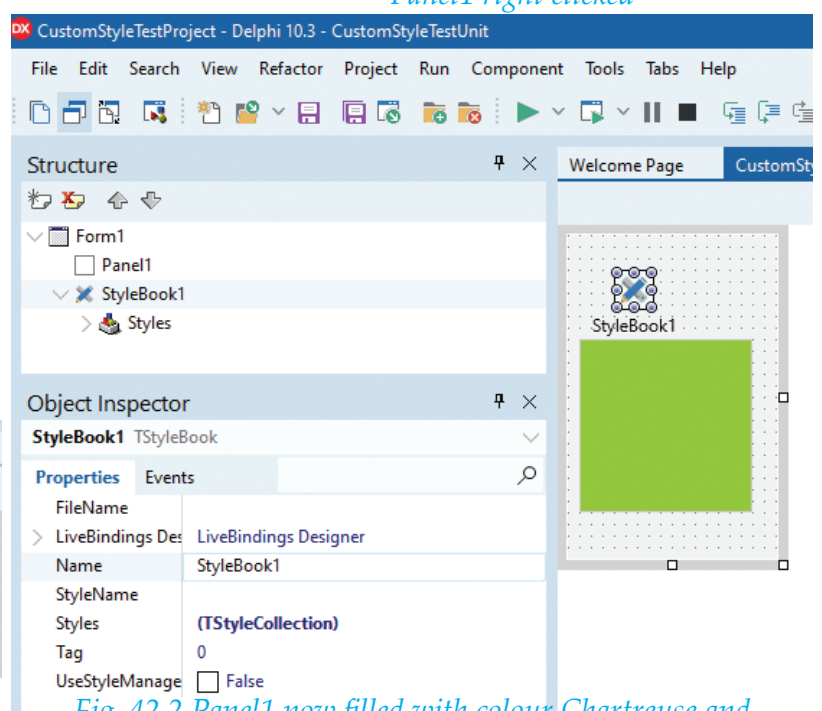


Fig. 42.2 Panel1 now filled with colour Chartreuse and StyleBook1 added

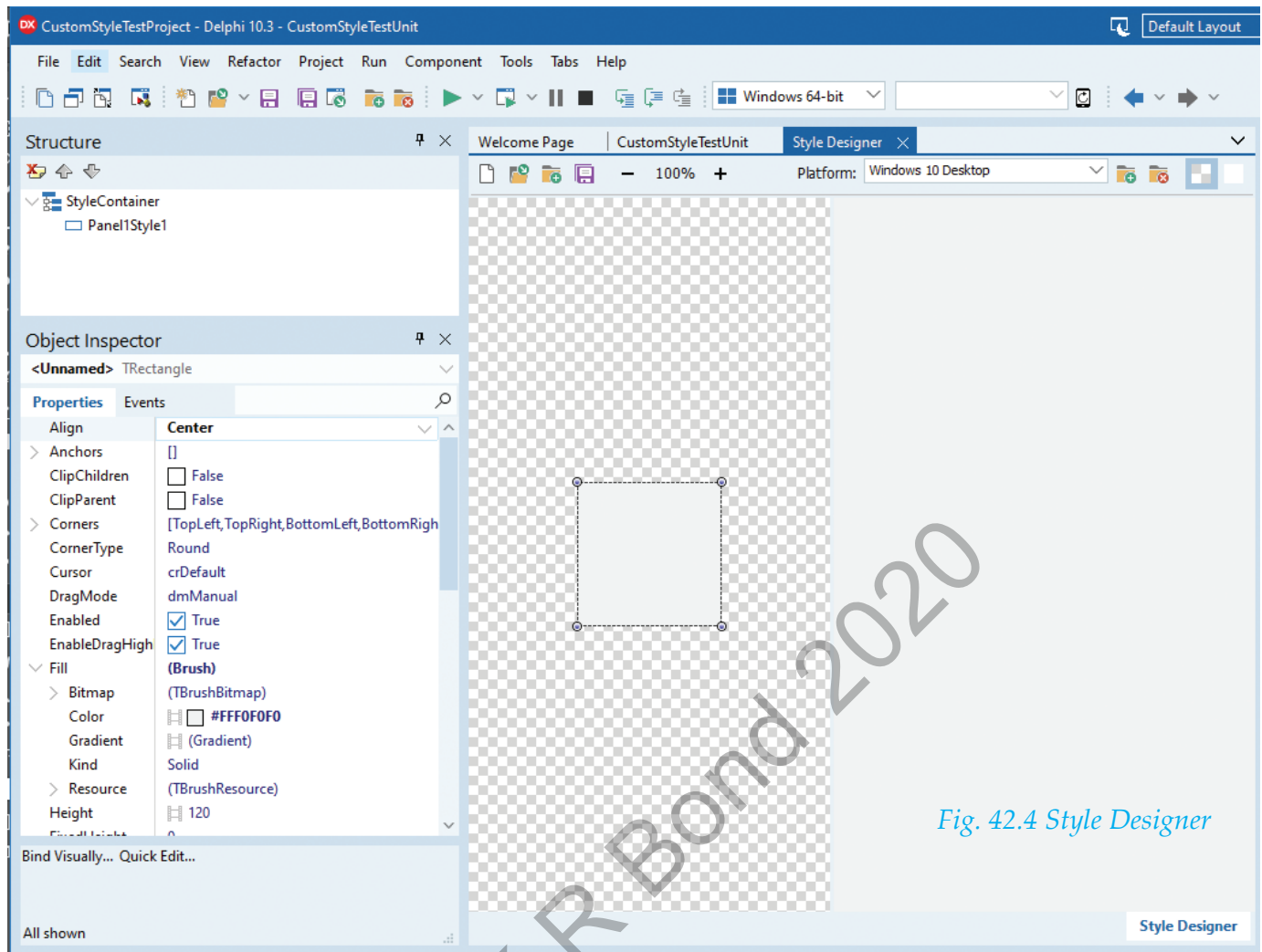


Fig. 42.4 Style Designer

Figure 42.5 shows that the **StyleName** property field has been set to `Panel1Style1` for `PanelStyleChartreuse`.

Figure 42.6 shows that a new custom style named `Panel1Style1` has been created and applied to the **StyleLookup** field of `Panel1`. The default style is `panelstyle`, the initial style for `Panel1`.

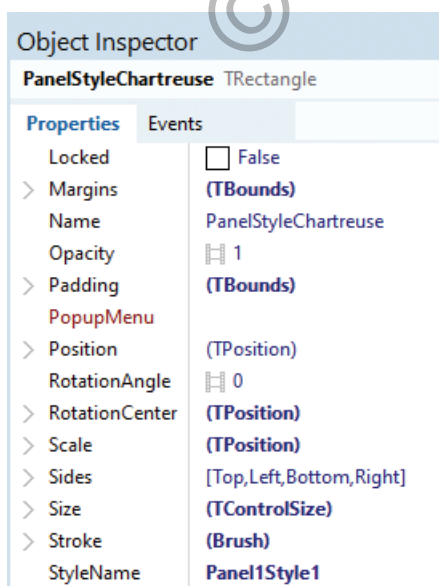


Fig. 42.5 Object Inspector reveals new style as StyleName Panel1Style1

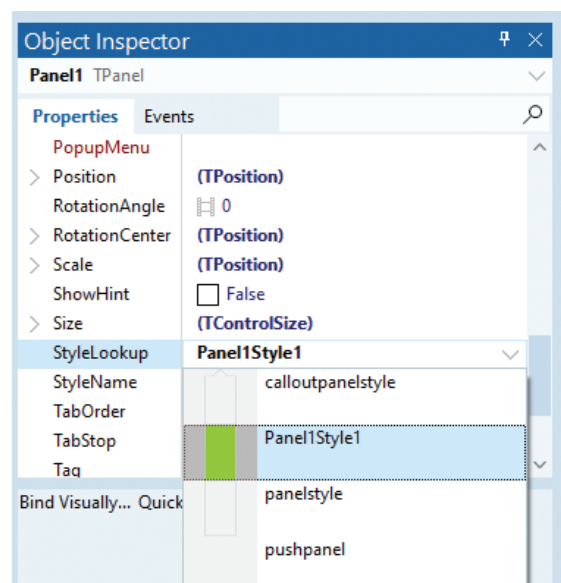


Fig. 42.6 Object Inspector reveals new style added to StyleLookup

■ Purpose: To learn how to use anchors

Anchors are used to set the way a control resizes and moves around when the form changes. Each control has 0 to 4 anchors.

The position of the control relative to one or more edge(s) of parent may be set as follows:

- Top
- Bottom
- Left
- Right
- Any combination of the above

The default is **Top**, **Left**.

Typically, the **Object Inspector** is used to set the values of these anchors at design time. *Figure 42.7* shows the **Anchors** property in the **Object Inspector**. The anchors have field names **akLeft**, **akTop**, **akRight** and **akBottom**.

The custom-styled **TPanel** control, **Panel1**, shown in *Figure 42.8* is contained within its parent, a **TForm** control, **Form1**.

Figures 42.9, 42.10, 42.11, 42.12 show the effect of the different anchors when applied singly to the **TPanel** control of *Figure 42.8* and the form, **Form1**, is enlarged at design time. Note that the width and height of **Panel1** remain unaltered. *Figure 42.13* shows the result of enlarging the form in *Figure 42.8* when both **Left** and **Top** anchors of the **TPanel** control are selected. Note that the width and height of **Panel1** again remain unaltered.

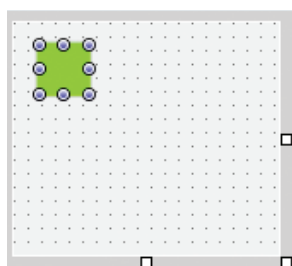


Fig. 42.13 Anchor - Left, Top, form enlarged

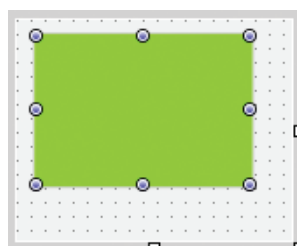


Fig. 42.14 Anchor - Left, Top, Right, Bottom form enlarged

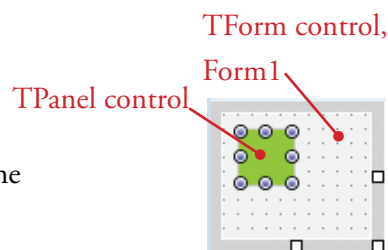


Fig. 42.8 Master view of Form1 containing custom-styled TPanel control, colour Chartreuse

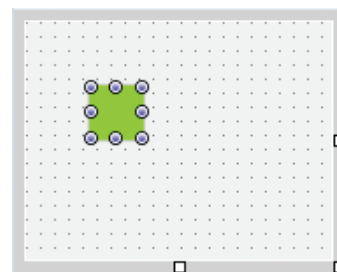


Fig. 42.9 No anchors, form enlarged

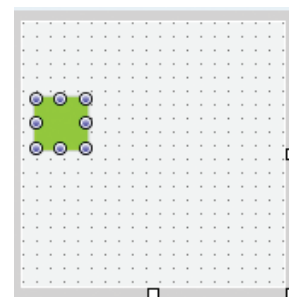


Fig. 42.10 Anchor - Left, form enlarged

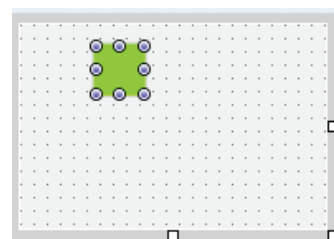


Fig. 42.11 Anchor - Top, form enlarged

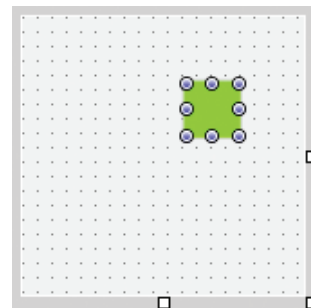


Fig. 42.12 Anchor - Right, form enlarged

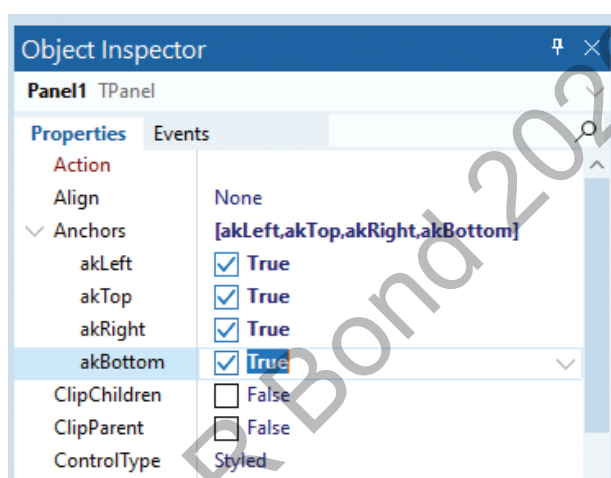


Fig. 42.7 The Anchors property exposed in the Object Inspector

Figure 42.14 shows the result of enlarging the form in *Figure 42.8* at design time when all four anchors are selected. The width and height of **Panel1** change.

■ Purpose: To learn how to use alignment

The anchors property enables a control to maintain a specified relationship with an edge of a parent as illustrated in the last section. If instead, we want the control to lie along an edge of its parent then we use the **Align** property of the control. The default is **None**.

Figure 42.15 shows the result of using the **Object Inspector** to set the **Align** property to **Bottom**, of the custom-styled **TPanel** control, **Panel1**, shown in Figure 42.8. The height

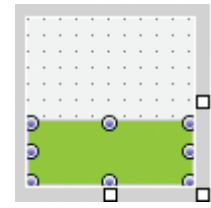


Fig. 42.15 Alignment - Bottom

value of **Panel1** in Figure 42.8 is preserved but its width and position are changed so that it lies along the bottom edge. Now, when the form is enlarged, the height of **Panel1** is preserved but its width increases so that it continues to lie along the bottom edge - Figure 42.16.

The respective effect for the other edges is achieved with alignment settings **Top**, **Left** and **Right**.

Figure 42.17 shows that when **Align** is **Top**, **Panel1**'s position changes and its width increases so that it lies along the top edge, but the height of **Panel1** remains the same.

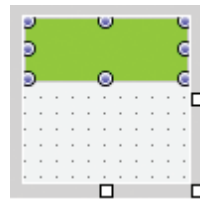


Fig. 42.17 Alignment - Top



Fig. 42.16 Alignment - Bottom, form enlarged

Figure 42.18 shows that when **Align** is **Left**, its position changes and its height increases so that it lies along the left edge, but the width of **Panel1** remains the same.

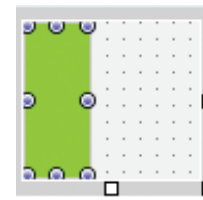


Fig. 42.18 Alignment - Left

Figure 42.19 shows the effect on **Panel1** in Figure 42.8 when **Align** for **Panel1** is set to **Center** (US spelling). The width and height of **Panel1** are unaltered.

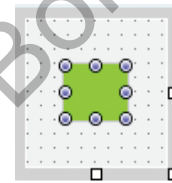


Fig. 42.19 Alignment - Center (US spelling)

Figure 42.20 shows the effect on **Panel1** in Figure 42.19 when the form is enlarged. Again, the width and height of **Panel1** are unaltered.

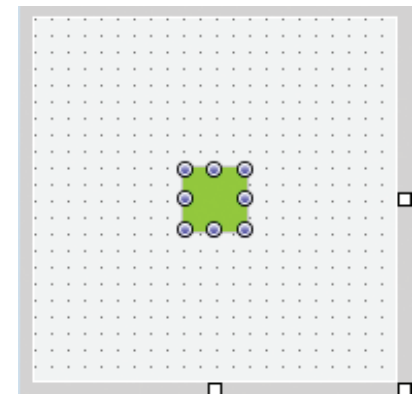


Fig. 42.20 Alignment - Centre, form enlarged

Figure 42.21 shows that if **Panel1**'s **Align** property is set to **Scale**, **Panel1** in Figure 42.8 resizes and moves to maintain the relative position and size as its parent, **Form1**, is resized.

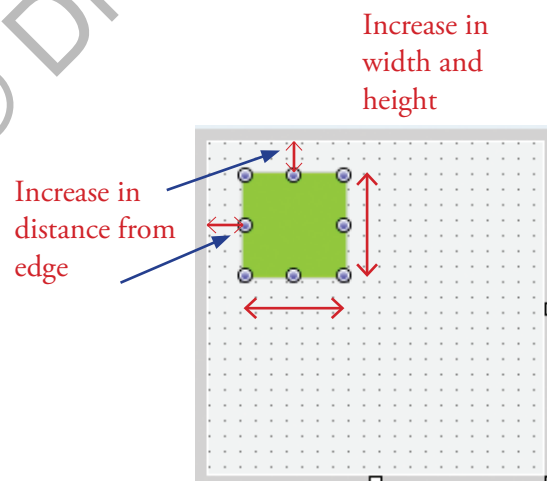


Fig. 42.21 Alignment - Scale, form enlarged

Figure 42.22 shows a form containing two custom-styled **TPanel** components, one with **Align** set to **Bottom** (red **TPanel** control) and one with **Align** set to **None** (chartreuse **TPanel** control)

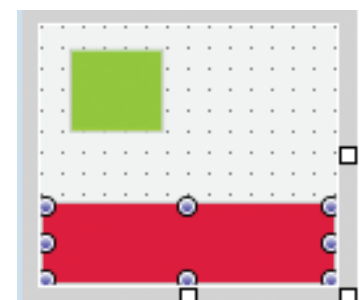


Fig. 42.22 Two custom-styled TPanel controls with align set to None and Bottom, respectively

Figure 42.23 shows the same form but with the chartreuse **TPanel** control's **Align** property set to **Client**. The **Client** setting causes the control to resize to fill the client area of its parent, the form.

As there is another bottom-pinned control already occupying part of the parent area, the chartreuse-coloured control resizes to fit the remaining parent area. *Figure 42.24* shows the client area of an example form, `Form1` for a Windows 64-bit desktop application.

Figure 42.25 shows two **TPanel** controls with their **Align** property both set to **Bottom**.

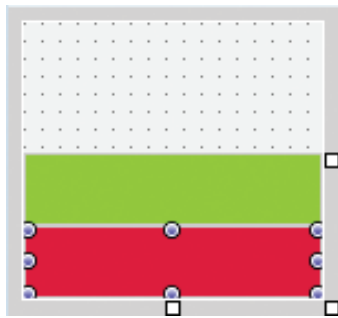


Fig. 42.25 Two custom-styled TPanel controls, Align set to Bottom for both

If the chartreuse-coloured **TPanel** control's **Align** property is now changed to **MostBottom** then this control is pinned to the very bottom of the form's client area and the red-coloured **TPanel** control moves to above it, as shown in *Figure 42.26*.

MostTop, **MostLeft** and **MostRight** are also available.

Figures 42.27 and *42.28* illustrate the use of **FitLeft**. The **TPanel** control resizes to partially fit the parent area by preserving the aspect ratio. The control moves to and pins to the left side of the parent.

FitTop, **FitBottom** and **FitRight** are also available.



Fig. 42.27 Custom-styled TPanel control, Align set to None

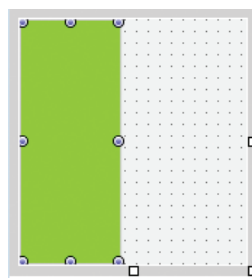


Fig. 42.28 Custom-styled TPanel control, Align set to FitLeft

Align property value **Contents** is similar to **Client** but differs in resizing to fill the entire bounds of its parent, to the extent of overlapping it and any other control present in the client area. *Figure 42.29* shows the result of setting the **Align** property of the red **TPanel** control in *Figure 42.26* to **Contents**.

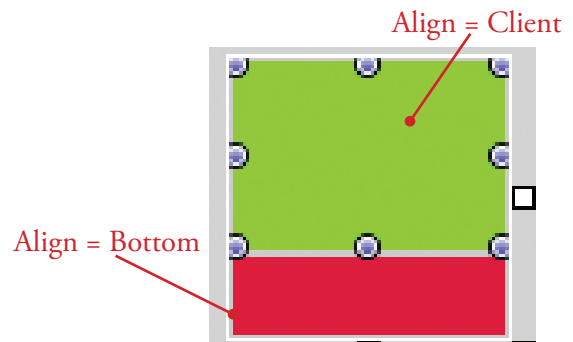


Fig. 42.23 Two custom-styled TPanel controls with align set to Client and Bottom, respectively

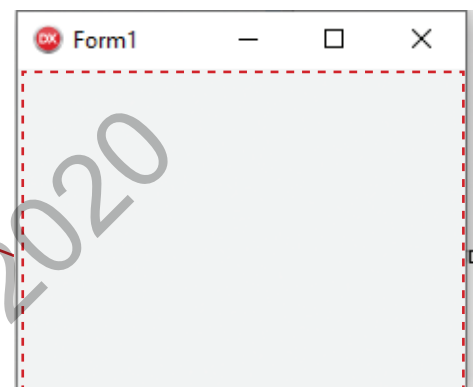


Fig. 42.24 FireMonkey Windows 64-bit desktop application

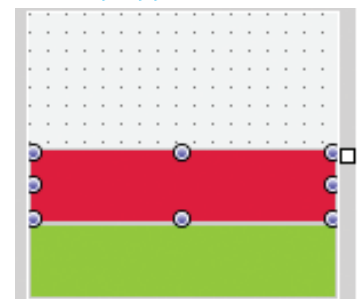


Fig. 42.26 Two custom-styled TPanel controls, red-coloured Align's property set to Bottom as before, chartreuse-coloured Align setting changed to MostBottom

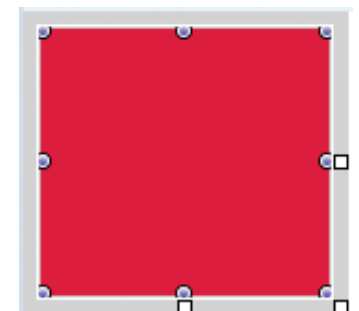


Fig. 42.29 Two custom-styled TPanel controls, chartreuse-coloured Align setting set to MostBottom, as before, and red-coloured Align's property set to Contents

Purpose: To learn how to use layouts

FireMonkey layouts are containers for other graphical objects that can be used to build visually attractive and complex user interfaces. FireMonkey layouts offer the possibility to manipulate a group of controls as a whole as well as enabling the arrangement, sizing, and scaling of their child controls. Complex user interfaces may be constructed, relatively easily, by just using properties such as **Position**, **Align**, **Margins**, and **Padding** with **Anchors** and layouts - see [Figure 42.30](#).

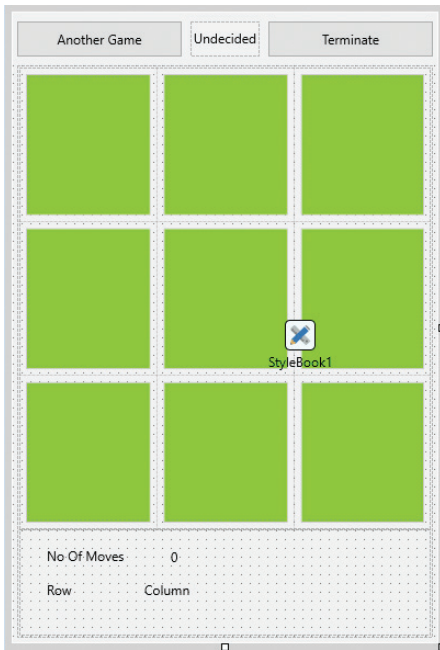


Fig. 42.30 User Interface design of a calculator application

The available layouts are located in the **Tool Palette**, under the **Layouts** category - [Figure 42.31](#).

An instance of **TLayout** is a simple container which is a parent to the other controls that it contains and which can be manipulated as a group, e.g. rotated, hidden or made visible by setting the **TLayout** instance's **Visible** property to **True** or **False**.

TLayout is visible at design time but not runtime and does not automatically set any properties of its children.

[Figure 42.32](#) shows a form at design time containing four **TLayout** controls, each containing a custom-styled **TPanel** control, colour chartreuse, and each of these, a **TText** control (**TText** is similar to **TCaption** but is less customisable than **TCaption**).

First, **Layout1** was created containing one **TPanel** and one **TText** control with its **Text** property set to 'X'. This layout was then copied three times and the **x** and **y Position** properties of each copy and the original, **Layout1**, set to achieve a grid arrangement as shown in [Figure 42.32](#). The **TStyleBook** control, **StyleBook1**, was created when the **TPanel** control, **Layout1**, was custom styled. It is only visible at design time. The align property of each **TLayout** control was set to **Scale**. The align property of each **TPanel** control was set to **Client** and each **TText** to **Center**.

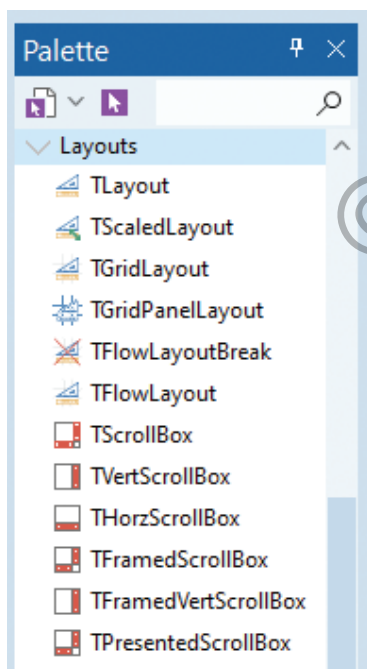


Fig. 42.31 Tool Palette, Layouts

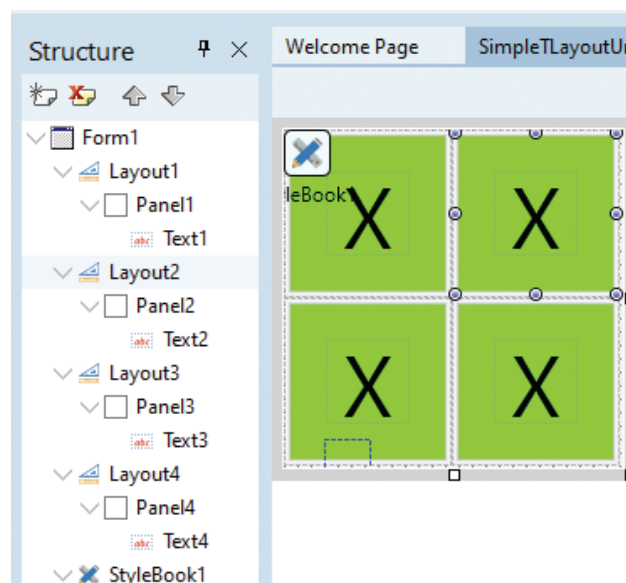


Fig. 42.32 Creating four TLayout containers each containing a TPanel and a TText control

Add a fifth **TLayout** control, `Layout5`, and arrange for it to be the parent of `Layout1` to `Layout4` inclusive, as shown in the **Structure** pane in [Figure 42.33](#). Set the `align` property of `Layout5` to `Client` so that it fills the client area of the form, `Form1`.

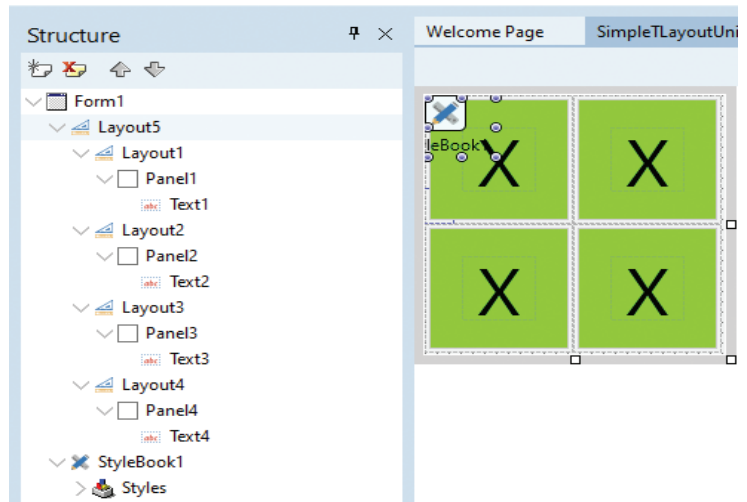
Save project as `SimpleTLayoutProject.dproj`.

[Figure 42.34](#) shows this project in execution.

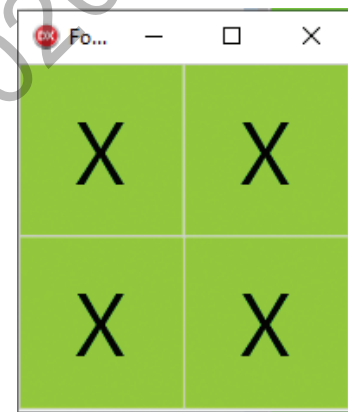
[Figure 42.35](#) shows the same project in execution but after scaling the form.

On scaling the form, the container `Layout5` expands so that it continues to fill the form's client area whilst `Layout1` to `Layout4` scale accordingly because they are contained by `Layout5` and their alignment is set to `Scale`.

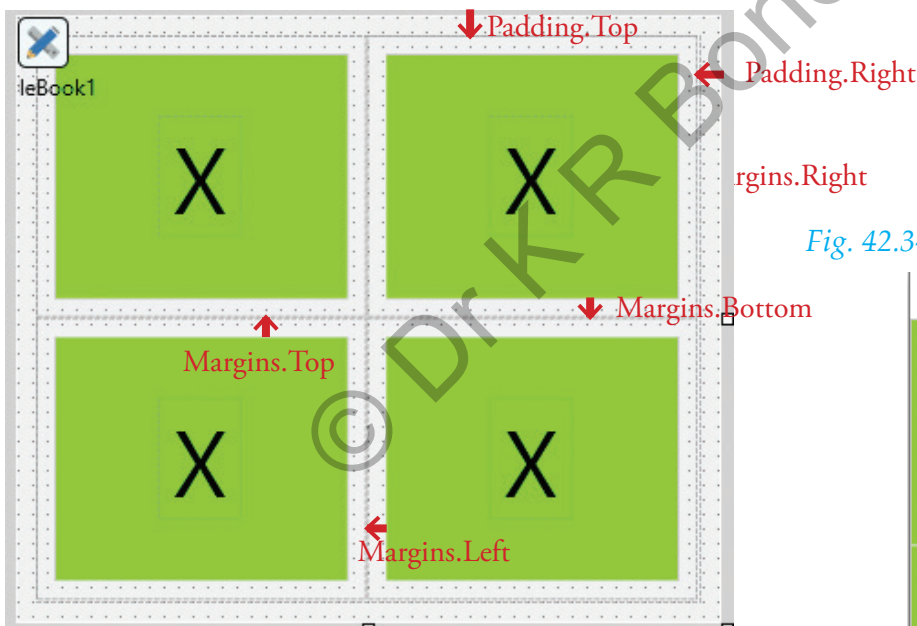
Margins and padding may be used to make clearer the boundaries between each panel and between the form and the panels as shown in [Figure 42.36](#).



[Fig. 42.33](#) Placing the four **TLayout** containers inside a fifth **TLayout** control

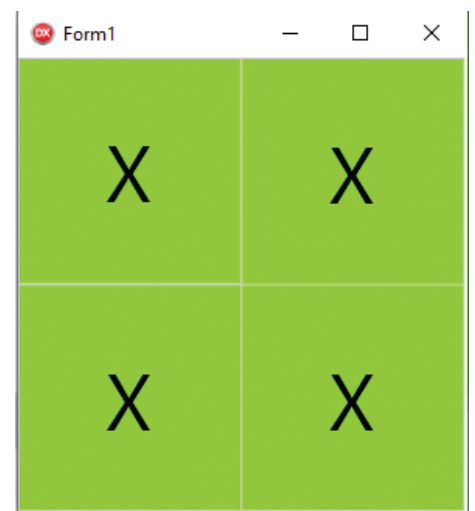
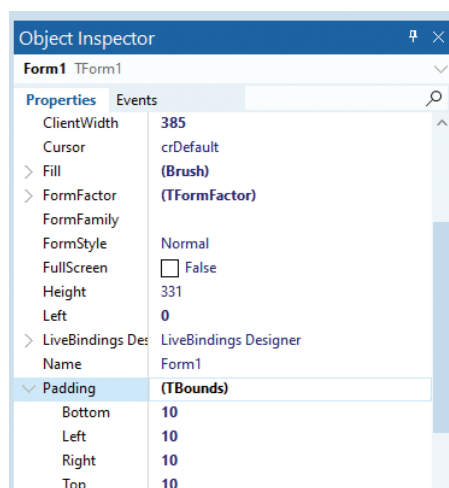


[Fig. 42.34](#) `SimpleTLayoutProject` in execution



[Fig. 42.36](#)
`SimpleTLayoutProject`
using margins and padding

Padding sets the spacing between parent and its children. [Figure 42.37](#) shows the **Bottom**, **Left**, **Right** and **Top** settings for the **Padding** property of `Form1` which are responsible for the padding shown in [Figure 42.36](#).



[Fig. 42.35](#) `SimpleTLayoutProject` in execution with form scaled at runtime

[Fig. 42.37](#) Setting **Padding** property of `Form1` to 10, **Left**, **Right**, **Top** and **Bottom**

HOW TO PROGRAM EFFECTIVELY IN DELPHI

Margins set the spacing between siblings and to the edges of their parent.

Figure 42.38 shows the Object Inspector settings for the **Margins** property of `Panel1`. Similar settings were applied to the **Margins** property of the other panel components.

Figure 42.39 shows the **SimpleTLayoutProject** in execution with margins and padding.

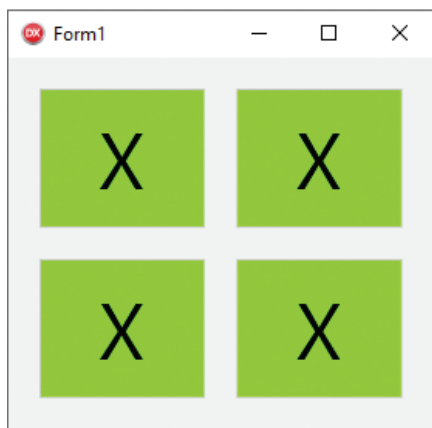


Fig. 42.39 SimpleTLayoutProject in execution showing margins and padding

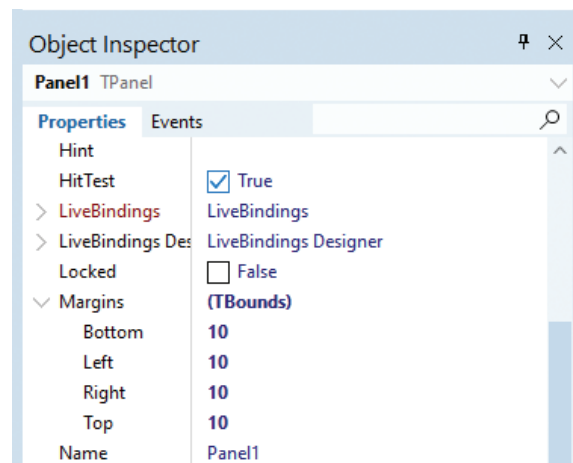


Fig. 42.38 Setting Margins property of Form1 to 10, Left, Right, Top and Bottom

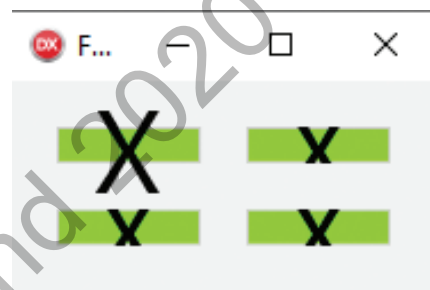


Fig. 42.40 SimpleTLayoutProject in execution and scaled down to show clipping for Panel2, Panel3 and Panel4

When the **BorderStyle** property of `Form1` is set to **Sizeable** the form is resizable. However, this creates a problem if the **ClipChildren** property of a **TPanel** component is not set to **True**. Figure 42.40 shows what happens on scaling down `Form1` when the **ClipChildren** property of `Panel1` is **False** and **True** for the other panels. Of course, it is possible to stop a form being scaled. To do this set the **BorderStyle** property of the form to **Single**. It might also be a good idea at the same time to set its **Position** property to **DesktopCenter**.

TLayout is one of several layout containers available from the **Layouts** tab of the component **Palette** - Figure 42.41.

In the next section we will use **TGridPanelLayout** to build a noughts and crosses board game.

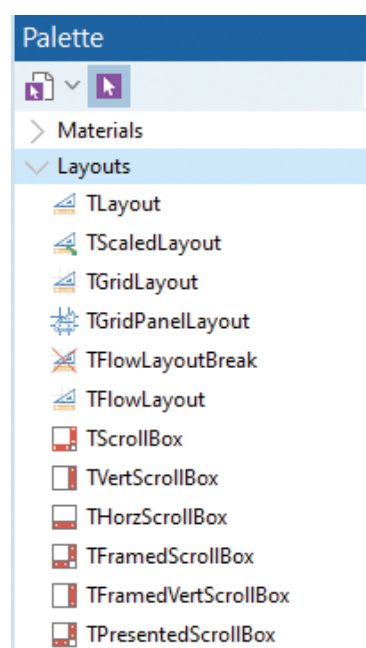



Fig. 42.41 Layouts tab of component Palette

■ Purpose: To learn how to develop a multi-device board game

We will build a single form application with FireMonkey that can be deployed to all GUI-supported platforms.

Create a Blank Multi-Device Application - **File|New|Multi-Device Application**. Save the project as **NoughtsAndCrossesProject** and its unit as **NoughtsAndCrossesUnit**.

1. Place a **TLayout** container on the form, set its **Align** property to **Client** and its **Name** property to **GameAreaLayout**. Rename **Form1**, **OsAndXsForm**. Set the form's **Width** to 387 and its **Height** to 574. Set its **Caption** to **Noughts and Crosses**.
2. Place a **TToolBar** control on the form and set its **Align** property to **MostTop**, and its **Height** property to 44.
3. Place a **TLayout** container on **GameAreaLayout**, set its **Align** property to **Bottom**, its **Height** property to 113 and its **Name** property to **BottomOfFormLayout**.
4. Place a **TGridPanelLayout** on **GameAreaLayout**, set its **Align** property to **Client** and its **Name** property to **GridPanelLayout**.
5. Click the ellipsis in the value field of property **ColumnCollection** of **GridPanelLayout** (Figure 42.43) to bring up the **ColumnCollection** editor - Figure 42.44. Click the icon  to add a new column. Adjust the **Value** field of each column until each is approximately 33.33% - Figure 42.45. You will need several attempts to achieve this.

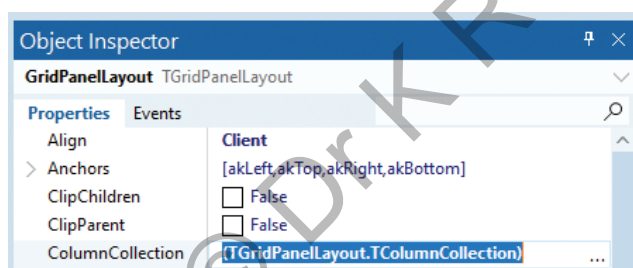



Fig. 42.43 Object Inspector showing *GridPanelLayout ColumnCollection* property

6. Click the ellipsis in the value field of property **RowCollection** of **GridPanelLayout** to bring up the **RowCollection** editor. Click the icon  to add a new row. Adjust the **Value** field of each row until each is approximately 33.33%. You will need several attempts to achieve this.
7. Select **GridPanelLayout** and add a **TPanel** component to the grid. This panel, **Panel1**, will be assigned automatically to the first grid cell.
8. Select **Style: Windows 64-bit**, **View: Master** and right click **Panel1** on the form and select **Edit Custom Style** to bring up the **Style Designer** - Figure 42.4. Change the **Fill|Color** to **Chartreuse** as described in the opening section of this chapter, set **Align** to **Client** and **Margins** all to 4. Click **File|Save All**.

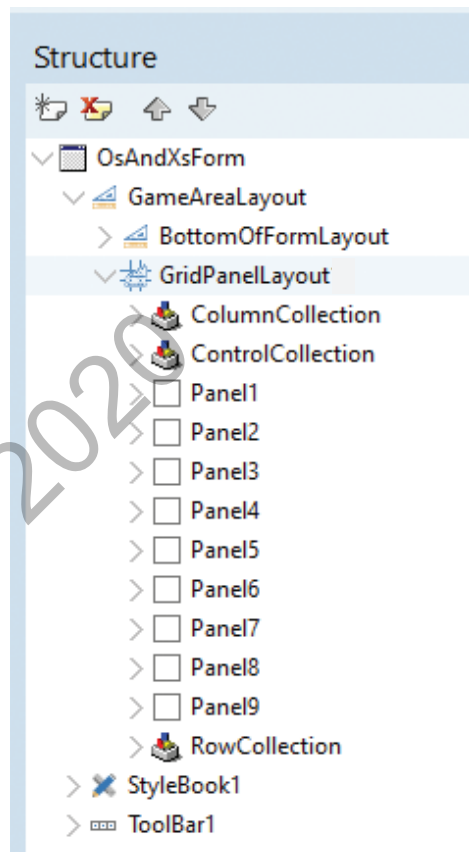


Fig. 42.42 Structure pane for *NoughtsAndCrossesProject*

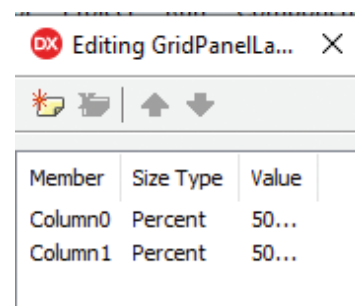


Fig. 42.44 Editing *GridPanelLayout. ColumnCollection*

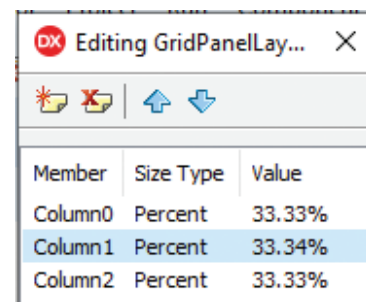


Fig. 42.45 Editing *GridPanelLayout. ColumnCollection*

9. Copy and paste `Panel1` eight times. The eight additional panels, `Panel2` to `Panel9`, will be assigned automatically row by row to the grid as shown in [Figure 42.46](#). Property `StyleLookup` should be `Panel1Style1`, the custom style created in bullet 8 which sets the panels' fill color to Chartreuse. Click **File|Save All**.
10. Set `GameAreaLayout`'s **Padding** to all 6. `OsAndXsForm` should now look as shown in [Figure 42.46](#) and the **Structure** pane as shown in [Figure 42.42](#).
11. Add a **TText** component to `Panel1`, set **Align** to **Client**, set **HitTest** to **True** then copy and paste to the other panels.
12. Change `Text1.Tag` to 11, `Text2.Tag` to 12, `Text3.Tag` to 13, `Text4.Tag` to 21, `Text5.Tag` to 22, `Text6.Tag` to 23, `Text7.Tag` to 31, `Text8.Tag` to 32, `Text9.Tag` to 33. Click **File|Save All**.
13. Place two **TButton** controls on the **TToolBar** control, `ToolBar1`. Change **Name** property of the first to `btnAnotherGame` and the second to `btnTerminate`. Set **Align** property of first button to **MostLeft** and set **Align** property of second button to **MostRight**. Set the **Width** of each to 150. Set the **Text** property of the first to `Another Game` and the second to `Terminate`. Set **Margins.Bottom**, **Left**, **Right** to 4 and **Margins.Top** to 8 for both buttons.
14. Add a **TLayout** container to `ToolBar1`. Change its **Name** to `StateOfGameLayout`. Set its **Align** property to **Client**. Set **Margins.Bottom**, **Left**, **Right** to 4 and **Margins.Top** to 8. Click **File|Save All**.
15. Place a **TLabel** component on `StateOfGameLayout`. Change its **Name** to `lblStateOfGame`, its **Align** property to **Center**, its **Text** property to `Undecided`, its **Width** to 151, its **TextSettings|HorzAlign** to **Center**.
16. Place four **TLabel** components on `BottomOfFormLayout`. Change their **Name** property to `lblColumn`, `lblRow`, `lblNextSymbol`, `lblNoOfMoves`, respectively. Set the **Text** property of `lblColumn` to `Column`, `lblRow` to `Row`, `lblNoOfMoves` to `No Of Moves`. Clear the **Text** property field of `lblNextSymbol`.
17. Set `lblNoOfMoves`'s **Position.X** to 25 and **Position.Y** to 16.
18. Set `lblRow`'s **Position.X** to 25 and **Position.Y** to 47.
19. Set `lblColumn`'s **Position.X** to 114 and **Position.Y** to 47.
20. Set `lblNextSymbol`'s **Position.X** to 208 and **Position.Y** to 16.
21. Add a **TLabel** component to `lblColumn`. Change its **Name** to `lblColumnNo`. Set **Position.X** to 64, **Position.Y** to 0, **Width** to 25. Clear the **Text** property field.
22. Add a **TLabel** component to `lblRow`. Change its **Name** to `lblRowNo`. Set **Position.X** to 39, **Position.Y** to 0, **Width** to 25. Clear the **Text** property field.
23. Add a **TLabel** component to `lblNoOfMoves`. Change its **Name** to `lblNumber`. Set **Position.X** to 113, **Position.Y** to 0, **Width** to 19. Set **Text** property field to 0. Click **File|Save All**.
24. Now **Run (F9)** the program. The result is shown in [Figure 42.47](#).

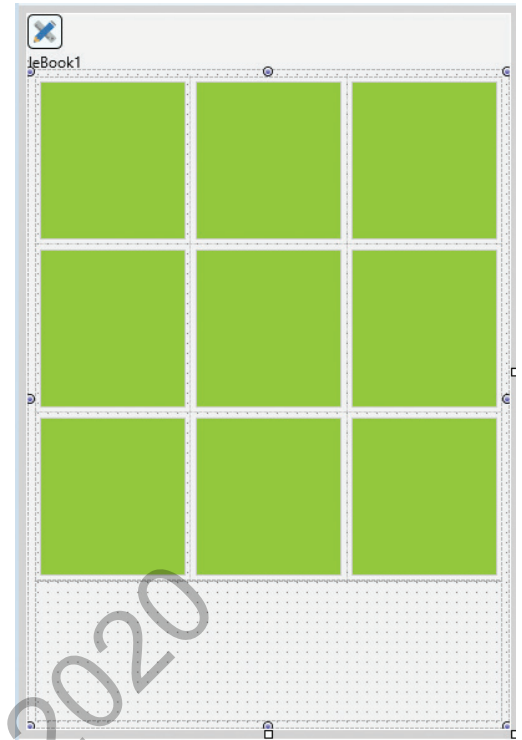


Fig. 42.46 OsAndXsForm

A two-dimensional array `GridArray` will be used to keep track of the state of the game. Its structure is as follows

```
TGridArray = Array[1..3,1..3] Of TSquareState;
```

`TSquareState` is a user-defined enumerated type defined as follows

```
TSquareState = (Empty, SquareCross,  
               SquareNought);
```

Add `TSquareState` and `TGridArray` definitions in that order to the **Type** area of the **Interface** section of the unit code.

Variable `GridArray` is declared as follows

```
GridArray : TGridArray;
```

The following nested for loop will be used to initialise `GridArray` at the beginning of each game:

```
For Row := 1 To 3  
Do  
  For Column := 1 To 3  
  Do GridArray[Row, Column] := Empty;
```

Two other user-defined enumerated types need to be added to the **Type** area of the **Interface** section.

```
TSymbol = (Nought, Cross);  
TGameState = (GameUndecided, Draw,  
             NoughtWin, CrossWin);
```

Select the `OsAndXsForm` in the Object Inspector and its **Events** tab. Double click in the **OnCreate** empty field to create an event handler `FormCreate` - *Figure 42.48*.

```
Procedure TOSAndXsForm.FormCreate(Sender: TObject);  
Var  
  Row, Column : Integer;  
Begin  
  NoOfMoves := 0;  
  Player1Symbol := Cross;  
  Player2Symbol := Nought;  
  lblNextSymbol.Text := 'X starts';  
  If (NoOfGamesPlayed Mod 2) = 0  
  Then InitialiseCurrentSymbol(Cross)  
  Else InitialiseCurrentSymbol(Nought);  
  For Row := 1 To 3  
  Do  
    For Column := 1 To 3  
    Do GridArray[Row, Column] := Empty;  
End;
```

Table 42.1 OnCreate event handler, FormCreate

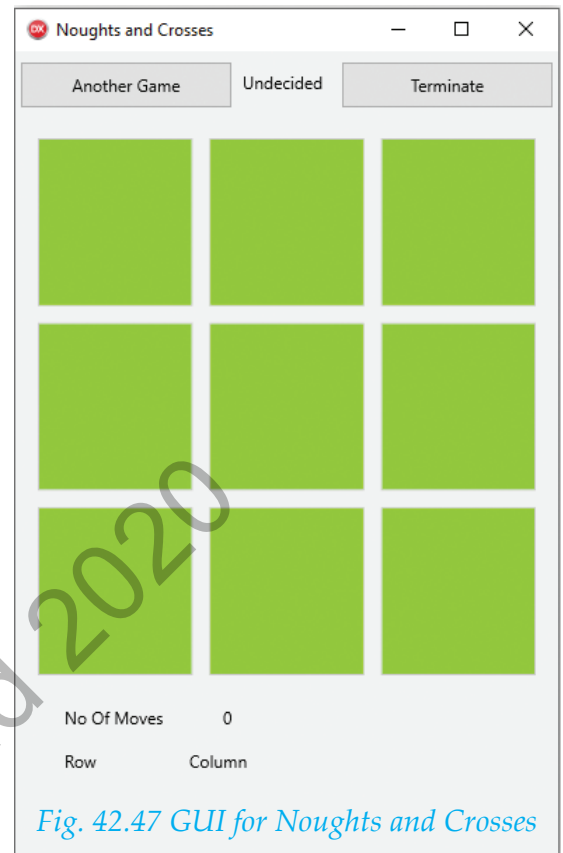


Fig. 42.47 GUI for Noughts and Crosses

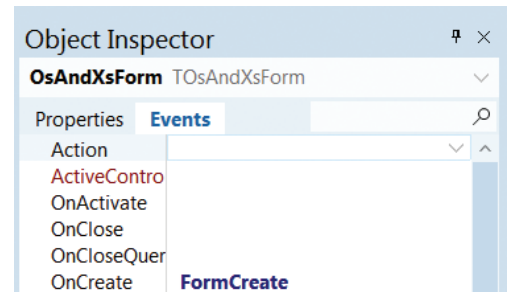


Fig. 42.48 FormCreate event handler

```
Private  
  Player1Symbol : TSymbol;  
  Player2Symbol : TSymbol;  
  NoOfMoves : Integer;  
  CurrentSymbol : TSymbol;
```

Table 42.2 Private fields of class TOSAndXsForm

Switch from the **Form** view to the **Unit** view and add code to the skeleton code for `FormCreate` so that it is as shown in *Table 42.1*.

Add the private fields shown in *Table 42.2* to the class `TOSAndXsForm`. Update the global variables section to that shown in *Table 42.43*.

HOW TO PROGRAM EFFECTIVELY IN DELPHI

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/Variables.txt)

Add the method header

```
Procedure InitialiseCurrentSymbol (InitialSymbol : TSymbol);
```

to the class `TOsAndXsForm` in the **Interface** section and the code shown in [Table 42.4](#) to the **Implementation** section.

```
Var
  OsAndXsForm: TOsAndXsForm;
  WhoGoesFirst : TSymbol;
  GridArray : TGridArray;
  GameOutcome : GameStateType = GameUndecided;
  NoOfGamesPlayed : Integer = 0;
  WinningRow : Integer = 0;
  WinningColumn : Integer = 0;
  WinningDiagonal : Integer = 0;
```

Table 42.3 Global variable declarations and initialisation

```
Procedure TOsAndXsForm.InitialiseCurrentSymbol (InitialSymbol : TSymbol);
Begin
  CurrentSymbol := InitialSymbol;
End;
```

Table 42.4 Method `TOsAndXsForm.InitialiseCurrentSymbol`

Select `Text1` in the Object Inspector and its **Events** tab. Double click in the **OnClick** empty field to create an event handler `Text1Click` - [Figure 42.49](#).

Add the code shown in [Table 42.5](#) to the body of this event handler.

Add `Text1Click` to the **OnClick** field of each of `Text2` to `Text9`.

Click **File|Save All**.

Now **Run (F9)** and test the program.

[Figure 42.50](#) shows the result of clicking on the middle square.

[Figure 42.51](#) shows the result of clicking on the square to the right of the middle square.

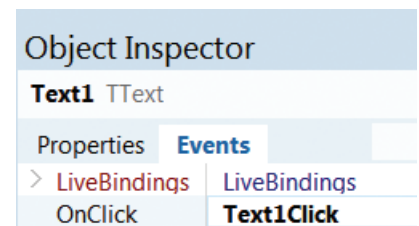


Fig. 42.49 `Text1Click` event handler

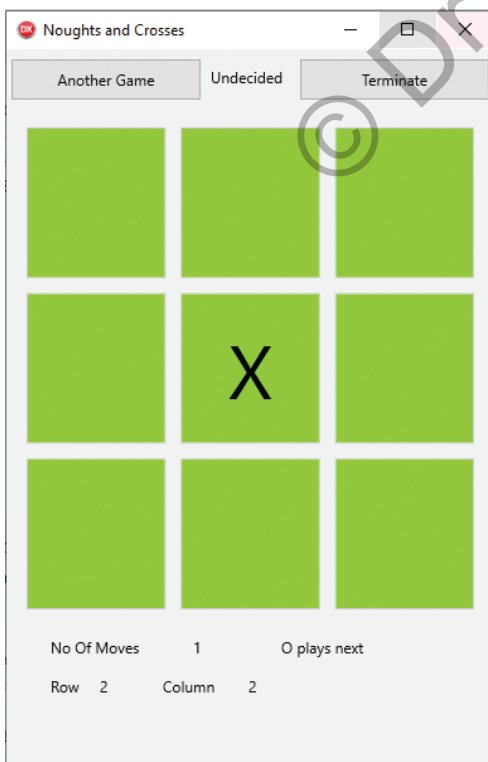


Fig. 42.50 Executing Noughts and Crosses and clicking on the middle square

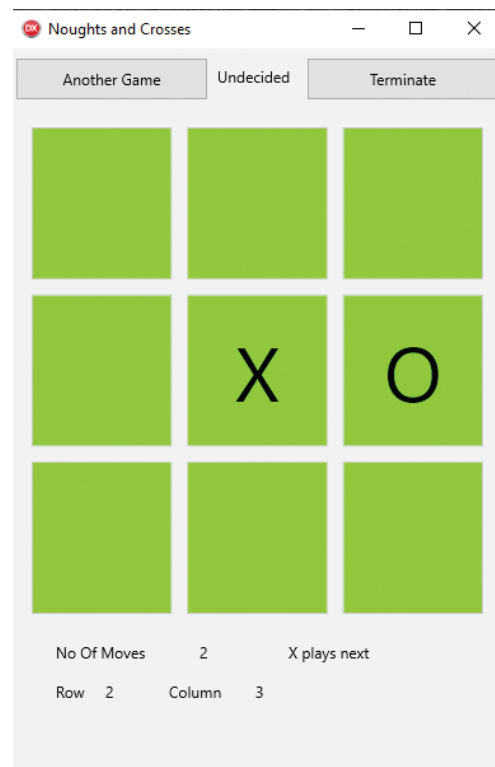


Fig. 42.51 Executing Noughts and Crosses and clicking on the square to the right of the middle square

```

Procedure TOsAndXsForm.Text1Click(Sender: TObject);
Var
  RowNo, ColumnNo : Integer;
Begin
  If GameOutcome = GameUndecided
  Then
    Begin
      If (Sender is TText)
      Then
        Begin
          (Sender As TText).TextSettings.Font.Size := 60;
          RowNo := (Sender As TText).Tag Div 10;
          ColumnNo := (Sender As TText).Tag Mod 10;
          If GridArray[RowNo, ColumnNo] = Empty
          Then
            Begin
              If CurrentSymbol = Cross
              Then
                Begin
                  (Sender As TText).Text := 'X';
                  GridArray[RowNo, ColumnNo] := SquareCross;
                  CurrentSymbol := Nought;
                  lblNextSymbol.Text := 'O plays next';
                End
              Else
                Begin
                  (Sender As TText).Text := 'O';
                  GridArray[RowNo, ColumnNo] := SquareNought;
                  CurrentSymbol := Cross;
                  lblNextSymbol.Text := 'X plays next';
                End;
              NoOfMoves := NoOfMoves + 1;
              lblRowNo.Text := IntToStr(RowNo);
              lblColumnNo.Text := IntToStr(ColumnNo);
              lblNumber.Text := IntToStr(NoOfMoves);
            End;
          End;
        End;
      End;
    End;
  End;
End;

```

P

Table 42.5 Initial code of OnClick event handler, Text1Click

The event handler `Text1Click` is called when a mouse click occurs over a square. The component which is clicked is identified in the parameter **Sender**. If it is a **TText** component and the game is not yet decided then the `RowNo` and `ColumnNo` of the clicked square is calculated from its **Tag** property. The `GridArray` cell with this `RowNo` and `ColumnNo` is then checked. If it is empty then the **Text** property of the **TText** component associated with the corresponding square is assigned an 'X' or an 'O' depending on the value of `CurrentSymbol`. The value of `CurrentSymbol` and the **Text** property of label `lblNextSymbol` are then changed as shown in Table 42.6.

```

If CurrentSymbol = Cross
Then
  Begin
    (Sender As TText).Text := 'X';
    GridArray[RowNo, ColumnNo] := SquareCross;
    CurrentSymbol := Nought;
    lblNextSymbol.Text := 'O plays next'
  End
Else
  Begin
    Sender As TText).Text := 'O';
    GridArray[RowNo, ColumnNo] := SquareNought;
    CurrentSymbol := Cross;
    lblNextSymbol.Text := 'X plays next'
  End;
End;

```

Casting from TObject to TText

(Sender As TText).Text := 'X';

GridArray[RowNo, ColumnNo] := SquareCross;

CurrentSymbol := Nought;

lblNextSymbol.Text := 'O plays next'

End

Else

Begin

Sender As TText).Text := 'O';

GridArray[RowNo, ColumnNo] := SquareNought;

CurrentSymbol := Cross;

lblNextSymbol.Text := 'X plays next'

End;

Table 42.6 Section of code from OnClick event handler, Text1Click, which updates game

HOW TO PROGRAM EFFECTIVELY IN DELPHI

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/TestStateOfBoard.txt)

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/GameOutcome.txt)

Following the successfully placement on the grid of an 'X' or an 'O', the state of the game must be checked by calling the function `TestStateOfBoard` and recording the result returned in variable `GameOutcome`, as shown in [Table 42.7](#). Add this code to the end of the event handler, `Text1Click`.

The code of function `TestStateOfBoard` is shown in [Table 42.8](#). Add Function `TestStateOfBoard : TGameState;` to class `TOsAndXsForm` in the **Interface** section.

Add Function `TOsAndXsForm.TestStateOfBoard : TGameState;` to the **Implementation** section.

Click **File|Save All**.

Now **Run (F9)** and test the program.

```
Function TOsAndXsForm.TestStateOfBoard : TGameState;
Var
  i, j : Integer;
Begin
  Result := GameUndecided;
  For i := 1 To 3
  Do
    Begin
      If (GridArray[i, 1] = GridArray[i, 2])
        And (GridArray[i, 2] = GridArray[i, 3])
        And (GridArray[i, 1] <> Empty)
      Then
        If GridArray[i, 1] = SquareCross
        Then
          Begin
            Result := CrossWin;
            WinningRow := i;
            Exit; // Exit function
          End
        Else
          Begin
            Result := NoughtWin;
            WinningRow := i;
            Exit;
          End;
        End;
      End;
    End;
  End;
End;
```

```
GameOutcome := TestStateOfBoard;
If (GameOutcome = NoughtWin)
Then lblStateOfGame.Text := 'Nought wins!';
If (GameOutcome = CrossWin)
Then lblStateOfGame.Text := 'Cross wins!';
If Not (GameOutcome = GameUndecided)
Then
  Begin
    {Change colour of winning squares to red}
  End;
```

Table 42.7 Section of code fromOnClick event handler, Text1Click, which updates game

```
For j := 1 To 3
Do
  Begin
    If (GridArray[1, j] = GridArray[2, j])
      And (GridArray[2, j] = GridArray[3, j])
      And (GridArray[1, j] <> Empty)
    Then
      If GridArray[1, j] = SquareCross
      Then
        Begin
          Result := CrossWin;
          WinningColumn := j;
          Exit;
        End
      Else
        Begin
          Result := NoughtWin;
          WinningColumn := j;
          Exit;
        End;
      End;
    End;
  End;
If (GridArray[1, 1] = GridArray[2, 2])
  And (GridArray[2, 2] = GridArray[3, 3])
  And (GridArray[1, 1] <> Empty)
Then
  Begin
    If GridArray[1, 1] = SquareCross
    Then Result := CrossWin
    Else Result := NoughtWin;
    WinningDiagonal := 1;
    Exit;
  End;
If (GridArray[1, 3] = GridArray[2, 2])
  And (GridArray[2, 2] = GridArray[3, 1])
  And (GridArray[1, 3] <> Empty)
Then
  Begin
    If GridArray[1, 3] = SquareCross
    Then Result := CrossWin
    Else Result := NoughtWin;
    WinningDiagonal := 2;
  End;
End;
```

Table 42.8 Function TOsAndXsForm.TestStateOfBoard

Double click button `Terminate` and add the following code to the event handler `TOsAndXsForm.btnTerminateClick`:

```
Application.Terminate;
```

Double click button `Another Game` and add the code shown in [Table 42.9](#) to the event handler `TOsAndXsForm.btnAnotherGameClick`.

```

Procedure TOSAndXsForm.btnAnotherGameClick(Sender: TObject);
Var
  Row, Column : Integer;
Begin
  NoOfGamesPlayed := NoOfGamesPlayed + 1;
  NoOfMoves := 0;
  Player1Symbol := Cross;
  Player2Symbol := Nought;
  If (NoOfGamesPlayed Mod 2) = 0
    Then InitialiseCurrentSymbol(Cross)
    Else InitialiseCurrentSymbol(Nought);
  For Row := 1 To 3
    Do
      For Column := 1 To 3
        Do GridArray[Row, Column] := Empty;
  Text1.Text := ''; Text2.Text := ''; Text3.Text := ''; Text4.Text := '';
  Text5.Text := ''; Text6.Text := ''; Text7.Text := ''; Text8.Text := '';
  Text9.Text := '';
  Panel11.StyleLookup := 'Panel1Style1'; Panel2.StyleLookup := 'Panel1Style1';
  Panel3.StyleLookup := 'Panel1Style1'; Panel4.StyleLookup := 'Panel1Style1';
  Panel5.StyleLookup := 'Panel1Style1'; Panel6.StyleLookup := 'Panel1Style1';
  Panel7.StyleLookup := 'Panel1Style1'; Panel8.StyleLookup := 'Panel1Style1';
  Panel9.StyleLookup := 'Panel1Style1';
  WinningRow := 0;
  WinningColumn := 0;
  WinningDiagonal := 0;
  lblStateOfGame.Text := 'Undecided';
  lblRowNo.Text := '';
  lblColumnNo.Text := '';
  GameOutcome := GameUndecided;
  If (NoOfGamesPlayed Mod 2) = 0
    Then lblNextSymbol.Text := 'X starts'
    Else lblNextSymbol.Text := 'O starts'
End;

```

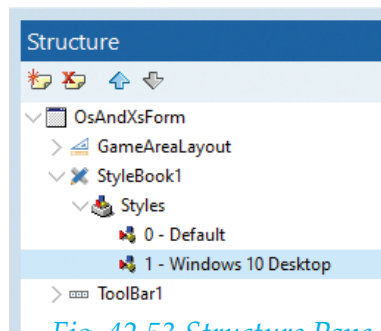
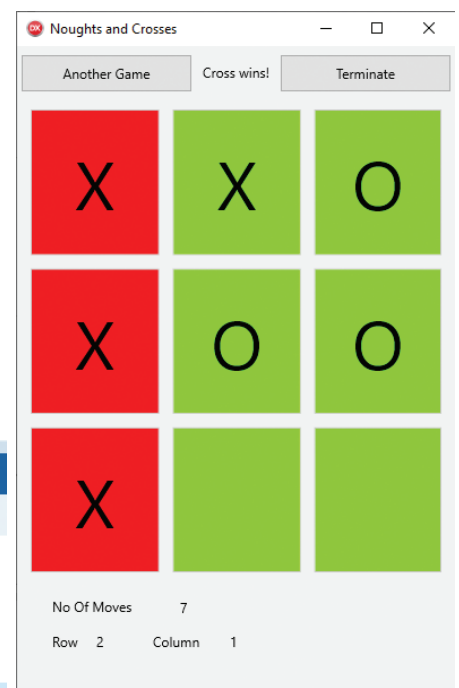
Table 42.9 Event handler TOSAndXsForm.btnAnotherGameClick

Figure 42.52 shows that it is possible to indicate a win state not only by text message, e.g. “Cross wins!” but also by changing the colour of the squares.

The code to do this is shown in Table 42.10. Use this code to replace the corresponding code from Table 42.7 in the event handler Text1Click. The custom style Panel1Style2 needs to be created. The easiest way to do this is to copy Panel1Style1, rename it Panel1Style2 then select **Fill|Color Red** before saving. Remember that the custom styles are platform specific, e.g. Windows 10 Desktop.

Select **StyleBook1|Styles|Windows 10 Desktop** in the **Structure Pane** as shown in Figure 42.53 (it will be **Windows 7 Desktop** if your O.S. is Windows 7).

In the **Object Inspector** click on the ellipsis (...) in the **Resource** property field as shown in Figure 42.54. This launches the **Style Designer**. Right click Panel1Style1 select **Edit|Copy**. Select and right click **StyleContainer**, select **Edit|Paste** to create

*Fig. 42.53 Structure Pane**Fig. 42.52 Winning configuration indicated in red*

HOW TO PROGRAM EFFECTIVELY IN DELPHI

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/RedSquares.txt)

a second `Panel1Style1` in the **Structure Pane**.

Select the copy `Panel1Style1` in the **Structure Pane** and change its **StyleName** property value to `Panel1Style2` as shown in *Figure 42.56*.

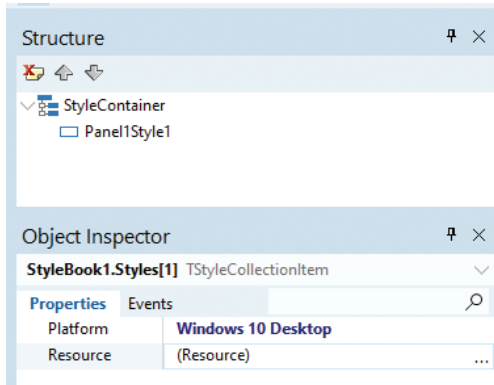


Fig. 42.54 Resource property

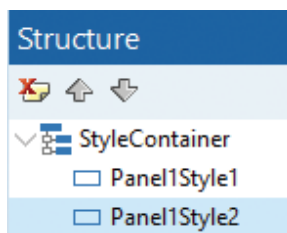


Fig. 42.56 Structure Pane

Select **Fill|Color|Red** for `Panel1Style2`.

Close the **Style Designer**, selecting **Save** to save the new style `Panel1Style2`.

Click **File|Save All**.

Now **Run (F9)** and test the program.

Finally, we need to take account of a draw, when all nine squares are occupied but no winner, by adding the following code to the end of event handler `Text1Click`:

```
If (GameOutcome = GameUndecided)
    And (NoOfMoves = 9)
Then
    Begin
        lblStateOfGame.Text := 'Draw';
        GameOutcome := Draw;
        lblNextSymbol.Text := '';
    End;
```

Click **File|Save All**.

Now **Run (F9)** and test the program.

```
If Not (GameOutcome = GameUndecided)
Then
    Begin
        lblNextSymbol.Text := '';
        If WinningRow <> 0
            Then
                Case WinningRow Of
                    1 : Begin
                        Panel1.StyleLookup := 'Panel1Style2';
                        Panel2.StyleLookup := 'Panel1Style2';
                        Panel3.StyleLookup := 'Panel1Style2';
                    End;
                    2 : Begin
                        Panel4.StyleLookup := 'Panel1Style2';
                        Panel5.StyleLookup := 'Panel1Style2';
                        Panel6.StyleLookup := 'Panel1Style2';
                    End;
                    3 : Begin
                        Panel7.StyleLookup := 'Panel1Style2';
                        Panel8.StyleLookup := 'Panel1Style2';
                        Panel9.StyleLookup := 'Panel1Style2';
                    End;
                End;
        If WinningColumn <> 0
            Then
                Case WinningColumn Of
                    1 : Begin
                        Panel1.StyleLookup := 'Panel1Style2';
                        Panel4.StyleLookup := 'Panel1Style2';
                        Panel7.StyleLookup := 'Panel1Style2';
                    End;
                    2 : Begin
                        Panel2.StyleLookup := 'Panel1Style2';
                        Panel5.StyleLookup := 'Panel1Style2';
                        Panel8.StyleLookup := 'Panel1Style2';
                    End;
                    3 : Begin
                        Panel3.StyleLookup := 'Panel1Style2';
                        Panel6.StyleLookup := 'Panel1Style2';
                        Panel9.StyleLookup := 'Panel1Style2';
                    End;
                End;
        End;
        If WinningDiagonal <> 0
            Then
                Case WinningDiagonal Of
                    1 : Begin
                        Panel1.StyleLookup := 'Panel1Style2';
                        Panel5.StyleLookup := 'Panel1Style2';
                        Panel9.StyleLookup := 'Panel1Style2';
                    End;
                    2 : Begin
                        Panel3.StyleLookup := 'Panel1Style2';
                        Panel5.StyleLookup := 'Panel1Style2';
                        Panel7.StyleLookup := 'Panel1Style2';
                    End;
                End;
        End;
    End;
```

Table 42.10 More code for Event handler `TOsAndXsForm.Text1Click`

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/NoughtsAndCrosses.rar)

To deploy `NoughtsAndCrosses` to an Android device successfully, requires that `Panel1Style1` and `Panel1Style2` are created for the Android platform, first. Connect an Android device via USB to the development computer. Select **Android 32-bit** or **Android 64-bit** and then select the corresponding target - *Figure 42.57*.



Fig. 42.57 Selecting Android 32-bit for a SamSung S8+target

Select **View: Master**.

Adding a **TPanel** component, `Panel10`, temporarily, is the surest way of being able to create the required custom styles. Right click the added panel and select **Edit Custom Style**. Ensure that the **Platform** field shows **AndroidL Light**, before renaming `Panel10Style1`, `Panel1Style1`. Change **Fill|Color** to **Chartreuse**. Copy and paste `Panel1Style1` then rename `Panel1Style2`. Change its **Fill|Color** to **Red**. Finally delete `Panel10`. *Figure 42.58* shows that the **StyleContainer** now contains `Panel1Style1` and `Panel1Style2`. Save the newly created styles. *Figure 42.59* shows the change to the **Structure Pane** which now includes the new style **AndroidL Light**. Click **File|Save All**.

Now **Run (F9)** and deploy the program to the connected Android device.

Figure 42.60 shows `NoughtsAndCrosses` executing on a SamSung S8+ connected to the development computer via USB.

The required custom styles must be created for each type of device to which the `NoughtsAndCrosses` program is to be deployed, e.g. an iPhone.

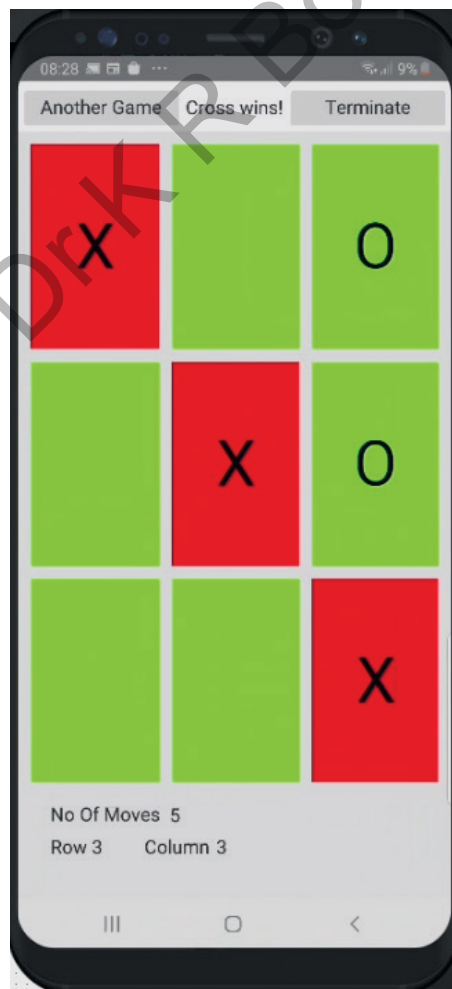


Fig. 42.60 Samsung S8+ running NoughtsAndCrosses

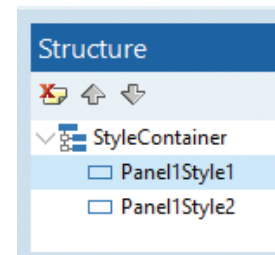


Fig. 42.58 Structure Pane showing the newly created styles in the Style Container

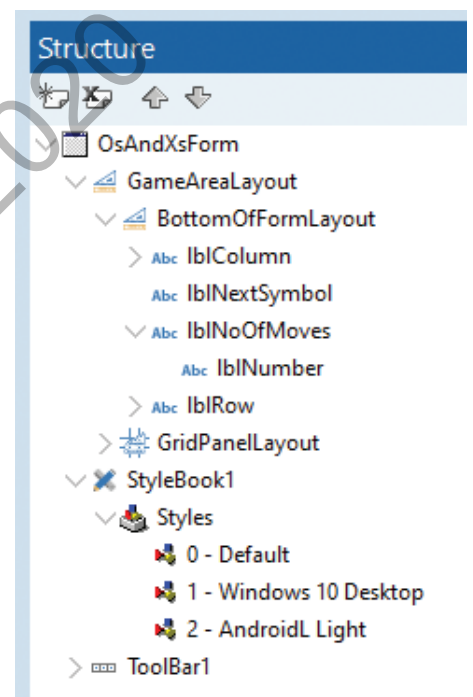


Fig. 42.59 Structure Pane showing the newly created AndroidL Light Style

TGridPanelLayout is one of several possible layout containers. **TGridLayout** is similar to **TGridPanelLayout** but unlike a **TGridPanelLayout** container doesn't allow a child control placed in a cell of the grid to be manually resized, aligned or anchored. With **TGridLayout**, the **Height** and **Width** properties of a child control are automatically set to fit the grid cell.

TFlowLayout arranges the child controls as if they were words in a paragraph, i.e. they are arranged and displayed in the layout in the order in which they were added. **TFlowLayoutBreak** is used to change to the next line.

Programming Task

- 1 Modify the Noughts and Crosses application to make it a computer versus human game. The computer should always go first and its symbol should always be an X. Use the **following rules** for the computer's moves:
Computer's moves:
Move 1: The computer must place an X in a corner.
Move 2: The computer must place an X in the opposite corner to move 1, if free, **Otherwise** it should place an X in a corner which is free.
Move 3: **If** 2 Xs and a space are in a line **Then** the computer should place an X in the space, **Else If** 2 Os and a space are in a line **Then** the computer should place an X in the space, **Otherwise** the computer should place an X in a free corner.
Move 4: The same as Move 3.
Move 5: The computer should place an X in the free space.

■ Purpose: To learn how to develop a multi-device application that uses rotation

The animation that we will create is an analogue clock¹ shown in [Figure 42.61](#).

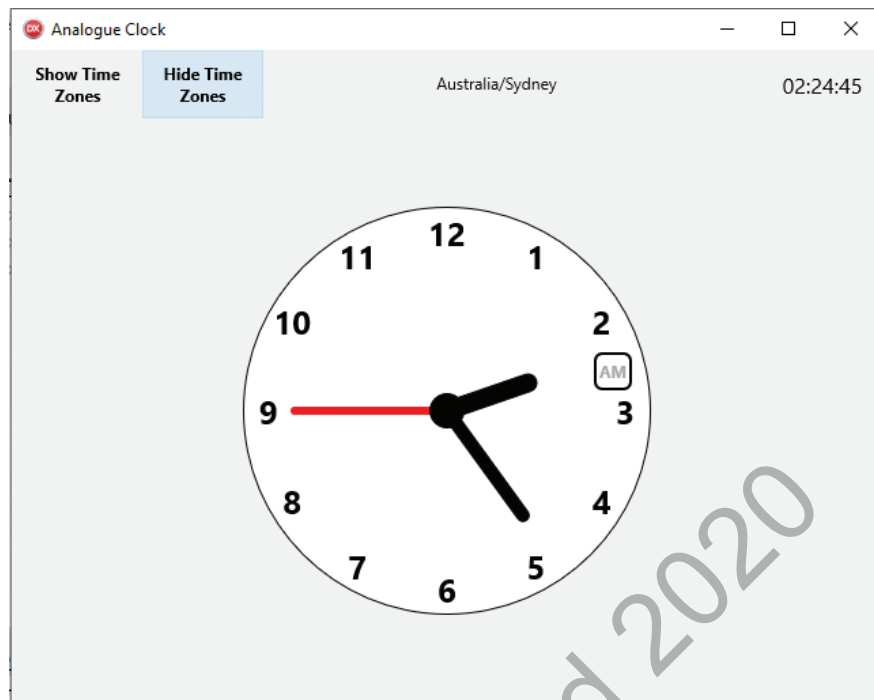


Fig. 42.61 AnalogueClockWithTimeZones in execution, time zone set to Sydney Australia

Create a Blank Multi-Device Application - **File|New|Multi-Device Application**. Save the project as `AnalogueClockWithTimeZones` and its unit as `AnalogueClockWithTimeZonesUnit`.

1. Set **Width** property of the form, `Form1`, to 640 and its **Height** property to 480. Set `Form1`'s **Caption** property to `Analogue Clock`.
2. Place a **TCircle** control, `Circle1`, on the form, `Form1`. Set its **Align** property to **Center**, its **Width** and **Height** properties to 300, its **Fill|Color** to **White**.
3. Place a **TLayout** container, `Layout1`, on the **TCircle** control, set its **Height** property to 300, and its **Width** to 37.5. Set **Position.X** to 131 and its **Position.Y** to 0. [Figure 42.62](#) shows the result of the first three steps.
4. Place a **TText** control, `Text1`, on the **TLayout** control, `Layout1`, set its **Height** property to 37.5, and its **Width** to 37.5. Set its **Position.X** to 0, and its **Position.Y** to 0. Set its **Text** property to 12. Set **TextSettings.HorzAlign** and **TextSettings.VertAlign** to **Center**. Set **TextSettings.Font.Size** to 24. Set **TextSettings.Font.Style** to **[fsBold]**. Set **TextSettings.Font.FontColor** to **Black** - [Figure 42.63](#).
5. Place a **TRoundRect** control on the the **TCircle** control, `Layout1`. Set its **Fill.Color** to **Black**. Set its **Width** to 14 and its **Height** to 70. Set **Position.X** to 143 and **Position.Y** to 80. Rename it `rrHour`.

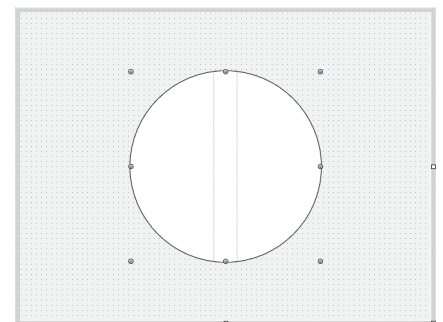


Fig. 42.62 TLayout container on TCircle on TForm

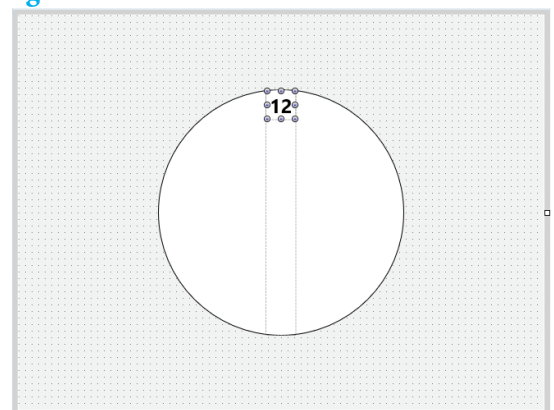


Fig. 42.63 TText on TLayout container on TCircle on TForm

¹ Based on an idea from Harry Stahl's book Cross-Platform Development with Delphi 10.2 & FireMonkey

HOW TO PROGRAM EFFECTIVELY IN DELPHI

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/FormCreateBasic.txt)

6. Place another **TRoundRect** control on the the **TCircle** control, *Layout1*. Set its **Fill.Color** property to **Red**. Set its **Width** to 10 and its **Height** to 100. Set **Position.X** to 145 and **Position.Y** to 50. Rename it *rrMinute*.
7. Place another **TRoundRect** control on the the **TCircle** control, *Layout1*. Set its **Fill|Color** to **Red**. Set its **Width** to 6 and its **Height** to 115. Set **Position.X** to 147 and **Position.Y** to 35. Rename it *rrSecond*. - *Figure 42.64*.
8. Place a **TLayout** container, *Layout2*, on *Form1*, set its **Align** property to **Top**, set its **Height** to 50.
9. Place two **TSpeedButton** controls on *Layout2* and set the **Align** property of *SpeedButton1* to **MostLeft**, and *SpeedButton2* to **Left**. Change the **Text** property of *SpeedButton1* to *Show Time Zones* and *SpeedButton2* to *Hide Time Zones*. Set **TextSettings.WordWrap** to **True** for both. Set the **Width** property of the *SpeedButton1* to 97 and the *SpeedButton2* to 89.
10. Place two **TText** controls on *Layout2*. Set the first's **Width** to 345 and **Height** to 33. Set the second's **Align** property to **Right** and **Width** to 88. Set **Position.X** of the first to 184 and **Position.Y** to 8.
11. Select *Layout1* and set its **StyleName** property to *ClockFaceNo*.
12. Select *Form1* and the **Events** tab in the Object Inspector. Double click the **OnCreate** event field to create the event handler, *TForm1.FormCreate*. Insert code from *Table 42.11* into the event handler. Add variable declaration `Var CurrentTimeZone : String;` after `{SR *.fmx}` in the the **Implementation** section.

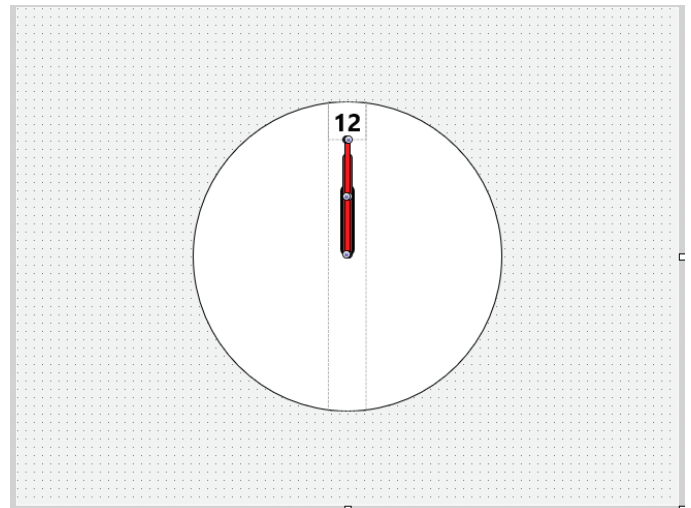


Fig. 42.64 Clockface with hour, minute and second hands

```

Procedure TForm1.FormCreate(Sender: TObject);
Var
  i : Integer;
  ClockFaceLayout : TLayout;
  ClockFaceNo : TText;
Begin
  For i := 1 To 11
  Do
    Begin
      ClockFaceLayout := TLayout(Circle1.FindStyleResource('ClockFaceNo', True));
      If ClockFaceLayout <> Nil
      Then
        Begin
          ClockFaceLayout.Parent := Circle1;
          ClockFaceLayout.RotationAngle := i * 30;
          ClockFaceNo := TText(ClockFaceLayout.Children[0]);
          If ClockFaceNo <> Nil
          Then
            Begin
              ClockFaceNo.Text := i.ToString;
              ClockFaceNo.RotationAngle := (i * 30) * -1;
            End;
          End;
        End;
      Text2.Text := 'etc/GMT';
      CurrentTimeZone := Text2.Text;
    End;
End;

```

Table 42.11 Event handler *TForm1.FormCreate*

```
ClockFaceLayout := TLayout(Circle1.FindStyleResource('ClockFaceNo', True));
```

After the above statement from event handler `FormCreate` is executed, `ClockFaceLayout` either contains `Nil` or a reference to a copy of a `TLayout` object with `StyleName` property `ClockFaceNo`, i.e. `Layout1`.

`FindStyleResource` returns the resource object linked directly to the control `Circle1` or if none exists then it searches amongst the control's children for a style resource `ClockFaceNo`. Of course, it matches resource object `Layout1` because this object has `StyleName` `ClockFaceNo` and is a child object of `Circle1`.

A copy of `Layout1` is therefore returned, referenced by variable `ClockFaceLayout`.

The following code sets the parent of this copy to `Circle1` before rotating the copy through a multiple of 30 degrees determined by the current value of `i` which may be an integer from 1 to 11.

```
ClockFaceLayout.Parent := Circle1;
ClockFaceLayout.RotationAngle := i * 30;
```

Next, a reference, `ClockFaceNo`, is obtained, using the following code, to the `TText` control child of `Layout1`

```
ClockFaceNo := TText(ClockFaceLayout.Children[0]);
```

The following code changes the `Text` property of the `TText` control referenced by `ClockFaceNo` to the current value of `i` (`i` cycles through 1 to 11) before rotation through a multiple of 30 degrees anticlockwise (times -1) from the x-axis of the rotated `TLayout` copy.

```
ClockFaceNo.Text := i.ToString;
ClockFaceNo.RotationAngle := (i * 30) * -1;
```

Figure 42.65 shows the `TLayout` copy rotated clockwise through 90 degrees ($i = 3$) and its `TText` control changed to showing `Text` property value 3. It is clear from this figure that the `TText` control must be rotated anticlockwise by 90 degrees if it is to display as shown in Figure 42.66.

Figure 42.66 shows the result when `AnalogueClockWithTimeZones` is executed.

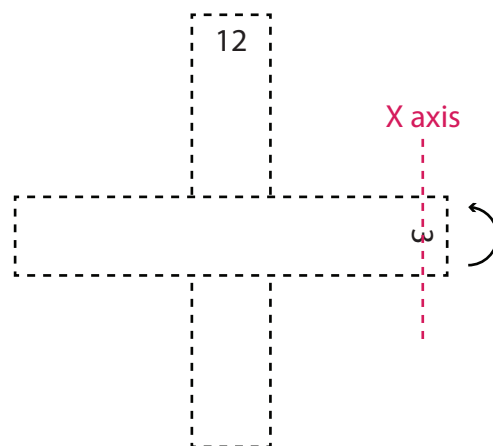
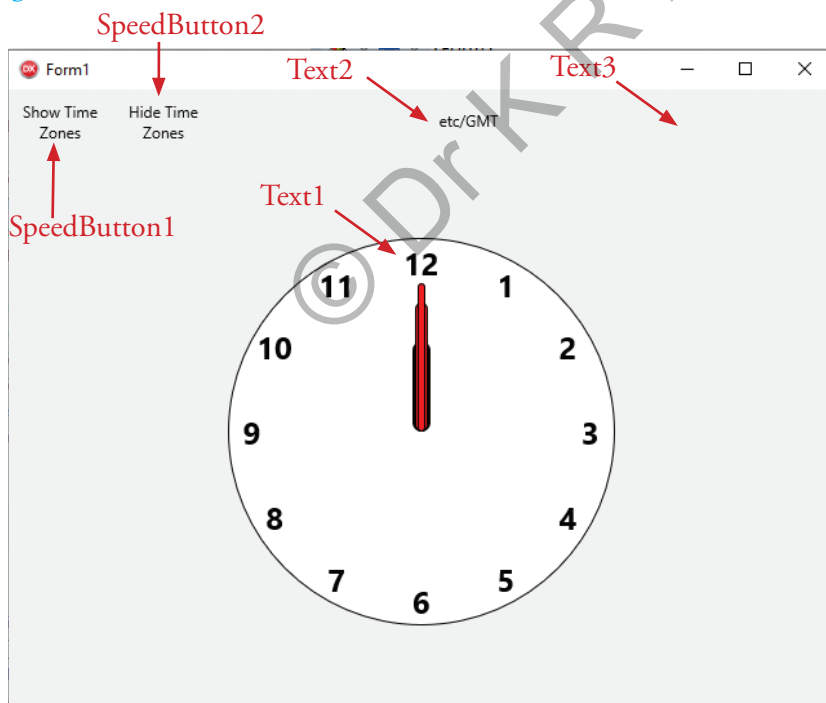


Fig. 42.65 TLayout original and a copy which has been rotated through 90 degrees clockwise

Fig. 42.66 AnalogueClockWithTimeZones in execution, time zone Greenwich Mean Time

13. Add a `TTimer` control, `Timer1`, to the form. Set `Enabled` property to `True` and `Interval` to 1000 - Figure 42.67. The `Structure` pane in Figure 42.68 shows the design so far.

```

Procedure TForm1.Timer1Timer(Sender: TObject);
Var
    Hour, TwelveHour, Minute, Second : Word;
    strHour, strMinute, strSecond : String;
    DateTime : TDateTime;
Begin
    DateTime := Now;
    Hour := HourOf(DateTime);
    TwelveHour := Hour Mod 12;
    Minute := MinuteOf(DateTime);
    Second := SecondOf(DateTime);
    rrHour.RotationAngle := 30 * TwelveHour + Round(Minute/2.17);
    rrMinute.RotationAngle := 6 * Minute;
    rrSecond.RotationAngle := 6 * Second;
    strHour := Hour.ToString;
    strMinute := Minute.ToString;
    strSecond := Second.ToString;
    If (Length(strMinute) < 2)
    Then strMinute := '0' + strMinute;
    If (Length(strSecond) < 2)
    Then strSecond := '0' + strSecond;
    If (Length(strHour) < 2)
    Then strHour := '0' + strHour;
    Text3.Text := strHour + ':' + strMinute + ':' + strSecond;
End;

```

Table 42.12 Event handler TForm1.Timer1Timer

14. Select Timer1 and the **Events** tab in the Object Inspector. Double click the **OnTimer** event field to create the event handler, TForm1.Timer1Timer. Insert code from Table 42.12 into the event handler.
15. Add **System.DateUtils** to the **Uses** clause of the **Interface** section.

The next stage is to configure the rotation centre of rectangles rrHour, rrMinute, rrSecond. The coordinates of the rotation centre take values in the range 0 through 1. The point with the coordinates (0, 0) corresponds to the upper-left corner of the control as shown in Figure 42.69.

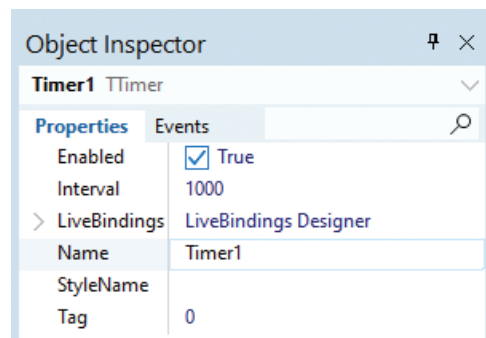


Fig. 42.67 Timer1 property settings

To rotate the hour, minute and second hands about the midpoint of the bottom edge of each, we need to set their **RotationCenter.X** property to 0.5 and their **RotationCenter.Y** property to 1.

16. Set **RotationCenter.X** property to 0.5 and **RotationCenter.Y** property to 1 for rrHour, rrMinute and rrSecond.

Click **File|Save All**.

Now **Run (F9)** and test the program.

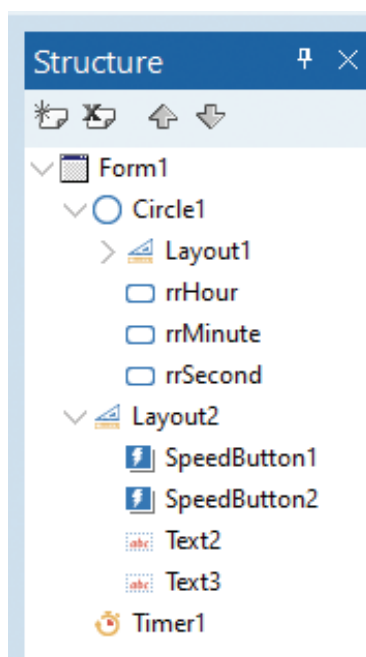


Fig. 42.68 Structure pane showing design so far

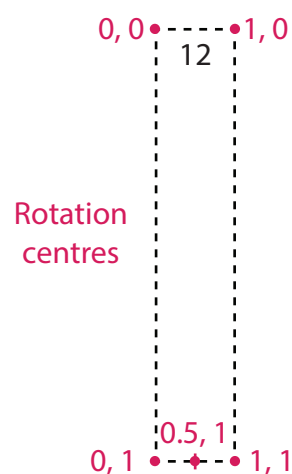


Fig. 42.69 Coordinates of some rotation centres for a rectangular-shaped control

Figure 42.70 shows `AnalogueClockWithTimeZones` in execution with clock hands movement and time displayed digitally. The event handler `Timer1Timer` is called every second because its `Interval` property has been set to 1000. This property's units are milliseconds, 1000 milliseconds = 1 second.

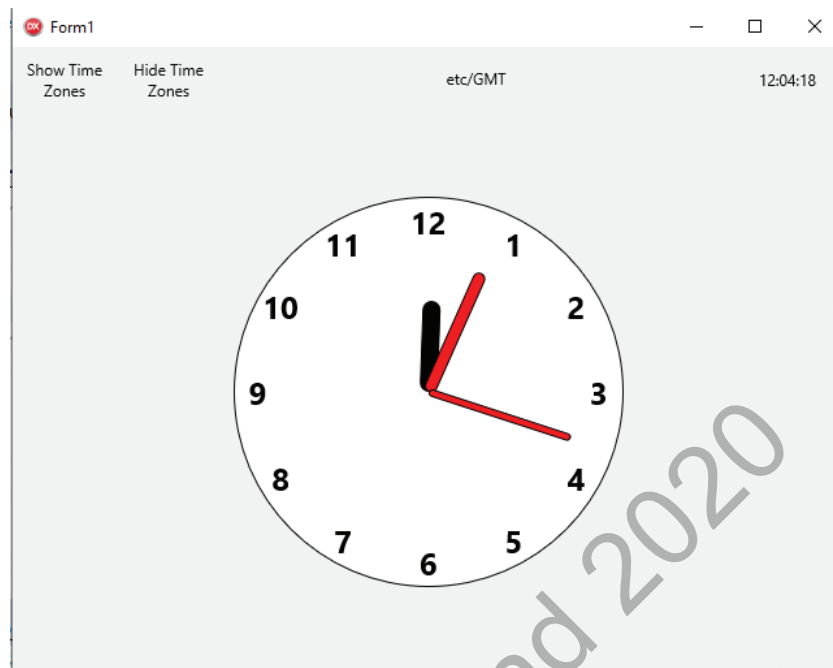


Fig. 42.66 `AnalogueClockWithTimeZones` in execution with clock hands movement

Questions

- 1 Explain the following snippets of code extracted from event handler `TForm1.Timer1Timer` shown in Table 42.12.

(a)

```
DateTime := Now;
Hour := HourOf(DateTime);
TwelveHour := Hour Mod 12;
Minute := MinuteOf(DateTime);
Second := SecondOf(DateTime);
```

(b)

```
rrHour.RotationAngle := 30 * TwelveHour + Round(Minute/2.17);
rrMinute.RotationAngle := 6 * Minute;
rrSecond.RotationAngle := 6 * Second;
```

(c)

```
strMinute := Minute.ToString;
strSecond := Second.ToString;
If (Length(strMinute) < 2)
Then strMinute := '0' + strMinute;
If (Length(strSecond) < 2)
Then strSecond := '0' + strSecond;
If (Length(strHour) < 2)
Then strHour := '0' + strHour;
Text3.Text := strHour + ':' + strMinute + ':' + strSecond;
```


HOW TO PROGRAM EFFECTIVELY IN DELPHI

It is not possible with the current design to distinguish AM from PM.

17. Add a **TRectangle** control, `Rectangle1`, to `Circle1`. Set its **Position.X** property to 258 and its **Position.Y** property to 107. Set its **Height** to 28 and its **Width** to 28. Set its **XRadius** property to 5 and its **YRadius** property to 5.

Set **Fill.Color** to **White**.

18. Add a **TText** control, `Text4`, to `Rectangle1`. Set its **Height** to 26 and its **Width** to 26. Set **Align** to **Center**.

19. Add the following code to the end of event handler `Timer1Timer`:

```
If IsPm(DateTime)
Then Text4.Text := 'PM'
Else Text4.Text := 'AM';
```

Click **File|Save All**.

Now **Run (F9)** and test the program.

Figure 42.67 shows the result.

The clock hands need to show that they are anchored.

20. Add a **TCircle** control, `Circle2`, to `Circle1`. Set its **Align** property to **Center**. Set its **Width** and **Height** to 26. Set its **Fill.Color** property to **Black**.

Click **File|Save All**.

Now **Run (F9)** and test the program.

Figure 42.68 shows the result.

The analogue clock application is only capable of showing the time for a single time zone. To change the application to one that supports selection of other time zone requires that a list of possible time zones be added.

We need to add a time zone unit **TZDB** to the **Uses** clause of the **Interface** section and install **TZDB.pas**, **TZDB.ico** and **TZDB.res** into the application's source code folder.

This unit contains the whole pre-compiled **TZ** database and all the code required to interpret it. The **TZ** database is a collaborative compilation of information about the world's time zones, primarily intended for use with computer programs and operating systems.

(<https://www.iana.org/time-zones>).

View **TZDB.pas** at <https://github.com/pavkam/tzdb/blob/master/dist/TZDB.pas>.

Download the **TZDB** library from <https://github.com/pavkam/tzdb> by clicking on **Clone or download|Download ZIP**.

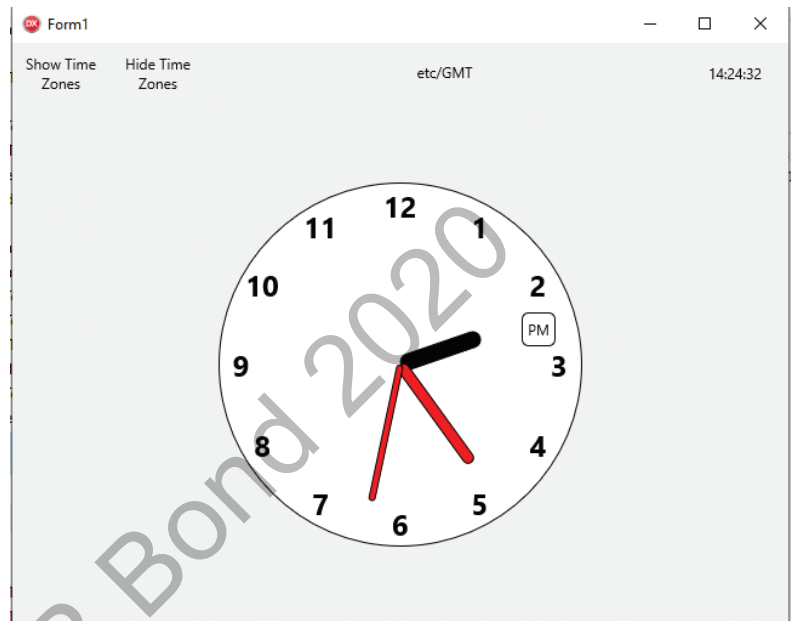


Fig. 42.67 AnalogueClockWithTimeZones in execution with clock hands movement and AM/PM indication

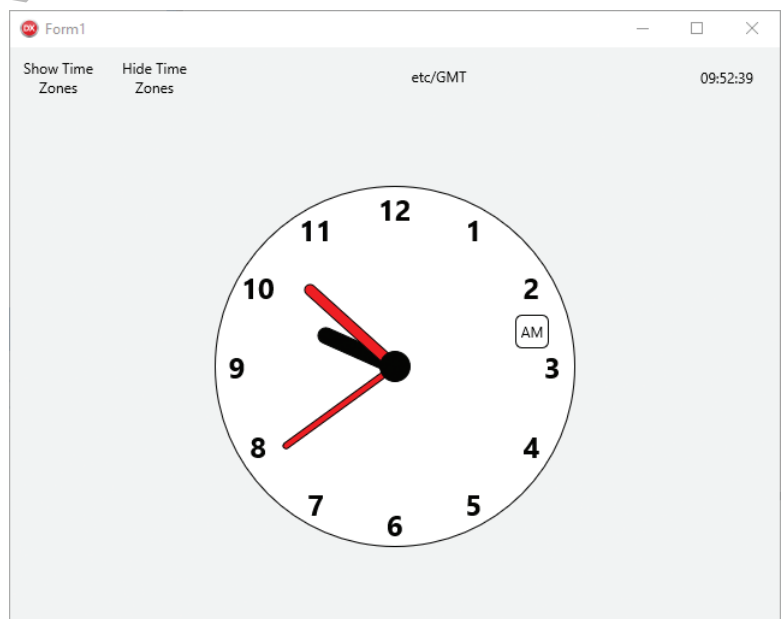


Fig. 42.68 AnalogueClockWithTimeZones in execution with clock hands anchored

Unzip the download and save to a folder.

21. Copy `TZDB.pas`, `TZDB.ico` and `TZDB.res` from `tzdb-master\dist` folder to the folder containing `AnalogueClockWithTimeZones.dproj`.
22. Add `LTimeZone : TBundledTimeZone;` to the **Var** declaration region of the **Implementation** section.
23. Add the time zone unit `TZDB.pas` to the **Uses** clause of the **Interface** section.
24. Add a **TListView** control, `ListView1`, to `Form1`. Set its **CanSwipeDelete** to **False**. Set its **Width** to 1. Set its **StyleLookUp** property to **listviewstyle**. Set its **EditMode** to **True**.
25. Add `LTZID : String; LItem : TListViewItem;` to **Var** declaration of event handler `FormCreate`.
26. Insert the code shown in red in [Table 42.13](#) into event handler `FormCreate`.
27. Double click `SpeedButton1` (Show Time Zones on form) to create event handler `SpeedButton1Click`.

```
Procedure TForm1.FormCreate(Sender: TObject);
.....
Begin
    .....
    For i := 1 To 11
        Do
            Begin
                .....
            End;
    ListView1.BeginUpdate;
    Try
        For LTZID in TBundledTimeZone.KnownTimeZones(True)
            Do
                Begin
                    LItem := ListView1.Items.Add;
                    LItem.Text := LTZID;
                End;
    Finally
        ListView1.EndUpdate;
    End;
    ListView1.SetFocus;
    LTimeZone := TBundledTimeZone.GetTimeZone('etc/GMT');
    Text2.Text := 'etc/GMT';
    CurrentTimeZone := Text2.Text;
End;
```

Table 42.13 Code in red to insert into event handler `TForm1.FormCreate`

28. Add the statement `ListView1.Width := 250;` to this event handler.
29. Double click `SpeedButton2` (Hide Time Zones on form) to create event handler `SpeedButton1Click`.
30. Add the statement `ListView1.Width := 0;` to this event handler.

Click **File|Save All**.

Now **Run (F9)** and test the program.

[Figure 42.69](#) shows the result when `Show Time Zones` is clicked.

Changing time zone requires an event handler to this when a different time zone is selected from `ListView1`.

31. Change to the **Events** tab of `ListView1` and double click in the field of **OnItemClick** to create event handler `ListView1ItemClick`.

HOW TO PROGRAM EFFECTIVELY IN DELPHI

32. Insert the code shown in red in [Table 42.14](#) into event handler `FormCreate`.

33. Add `SelectedItem : TListViewItem;` to **Var** declaration of **Implementation** section.

Click **File|Save All**.

Now **Run (F9)** and test the program.

[Figure 42.70](#) shows the result when `Show Time Zones` and then `Africa/Asmara` are clicked.

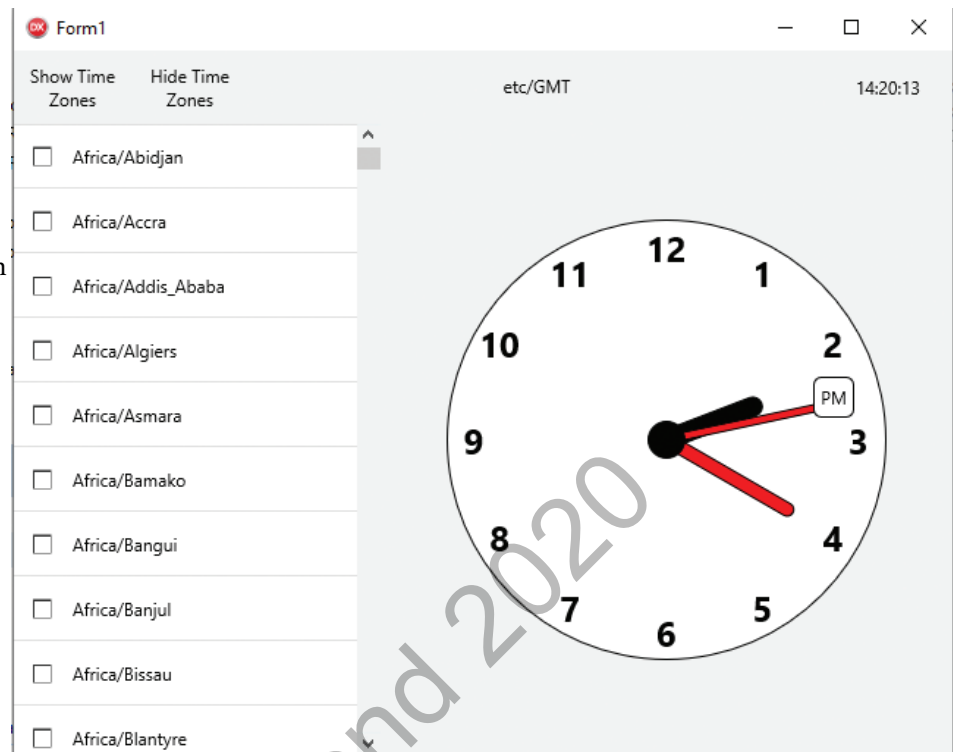


Fig. 42.69 AnalogueClockWithTimeZones in execution with Show Time Zones clicked

To apply the selected time zone to the clock, change

```
DateTime := Now;
```

to

```
DateTime := LTimeZone.ToLocalTime(Now);
```

in event handler `Timer1Timer`.

Click **File|Save All**.

Now **Run (F9)** and test the program.

[Figure 42.71](#) shows the result when `Show Time Zones` and then `Australia/Sydney` are clicked. The time shown by the clock changes to show Sydney time.

```
Procedure TForm1.ListView1ItemClick(Sender: TObject);
Var
    SelectedItem : TListViewItem;
Begin
    If SelectedItem <> Nil
    Then SelectedItem.Checked := False;
    If AItem <> Nil
    Then
        Begin
            Text2.Text := AItem.Text;
            CurrentTimeZone := Text2.Text;
            SelectedItem := AItem;
        End
    Else Text2.Text := CurrentTimeZone;
    LTimeZone := TBundledTimeZone.GetTimeZone(Text2.Text);
End;
```

Table 42.14 Code to insert into event handler `TForm1.ListView1ItemClick`

34. Finally change the **Caption** property of `Form1` to `Analogue Clock`.

Click **File|Save All**.

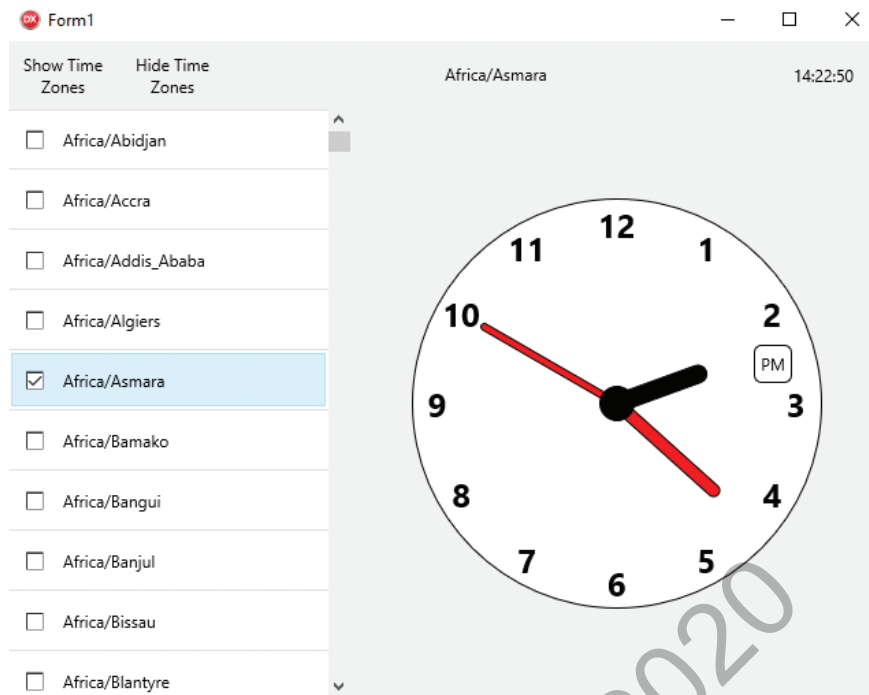


Fig. 42.70 AnalogueClockWithTimeZones in execution with Show Time Zones and then Africa/Asmara clicked

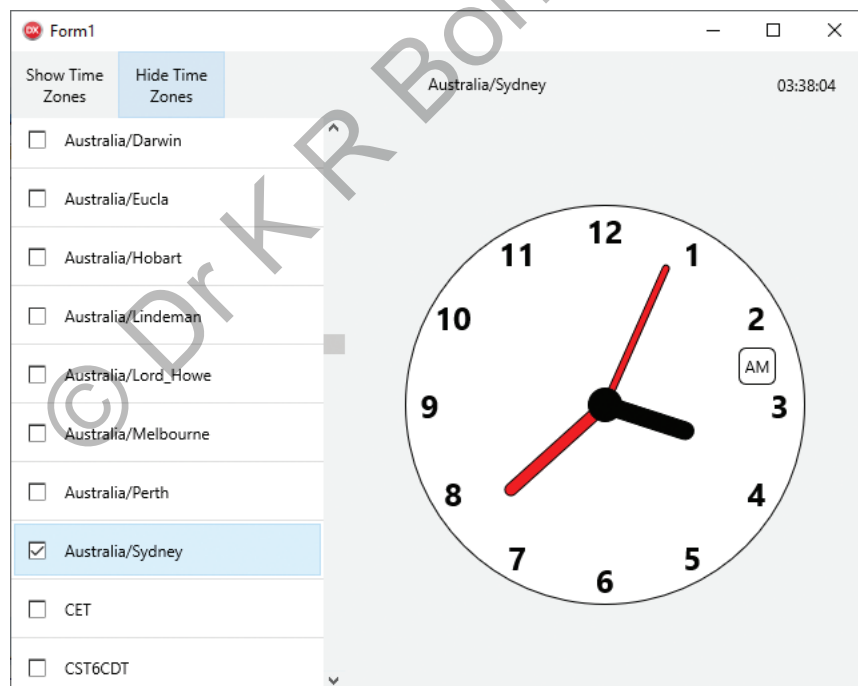


Fig. 42.71 AnalogueClockWithTimeZones in execution fully operational

(Download from www.educational-computing.com/DelphiBook/Code/Chapter42/AnalogueClockWithTimeZones.rar)

Purpose: To learn how to configure a multi-device application for a different target device

The Analogue Clock application was developed on a Windows 10 PC with target operating system set to Windows 64-bit and target device Windows desktop.

Adjustments may be needed to fit the application to a different target device, e.g. a Samsung S8+ mobile phone.

In this section, the target device will be a Samsung S8+ device in developer's mode connected via USB to the Windows 10 development PC.

1. First select the operating system and the target device as shown in [Figure 42.72](#) and [Figure 42.73](#).

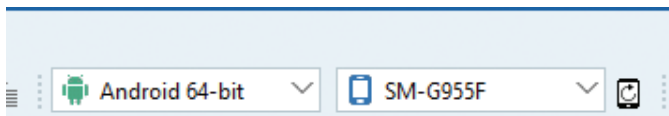


Fig. 42.72 Toolbar options windows for target operating system and device

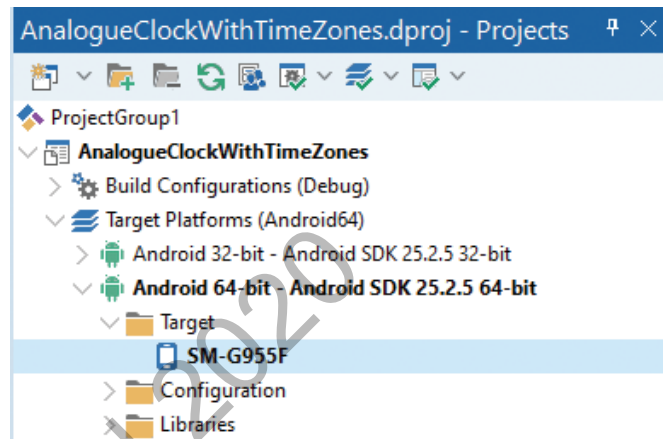


Fig. 42.73 Projects pane for AnalogueClockWithTimeZones

2. Select **Style: Android** and **View: Android 5" Phone**. The UI design changes to that shown in [Figure 42.73](#).
3. Set **Align** property for `SpeedButton2` to **Most Left**.
4. Set **Align** property for `Text2` to **Most Left**. Set its **Width** to 169.

Click **File|Save All**.

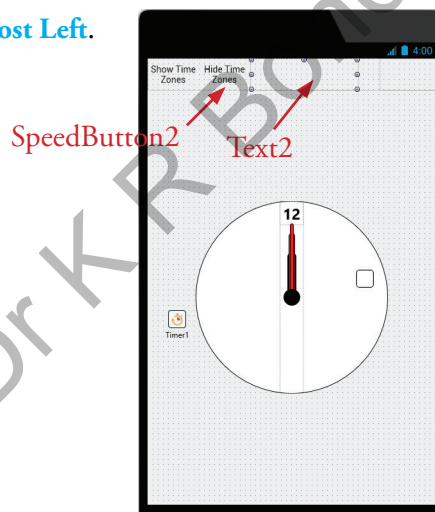


Fig. 42.73 UI design mode for Android 5" Phone

Now **Run (F9)** and deploy to the target device, a Samsung S8+ mobile phone.

[Figure 42.74](#) shows `AnalogueClockWithTimeZones` executing on the target device with time zone `Africa/Casablanca` selected.

[Figure 42.75](#) shows the application executing on the target device after `Show Time Zones` clicked and `America/Blanc-Sabon` selected.



Fig. 42.74 AnalogueClockWithTimeZones executing on Samsung S8+

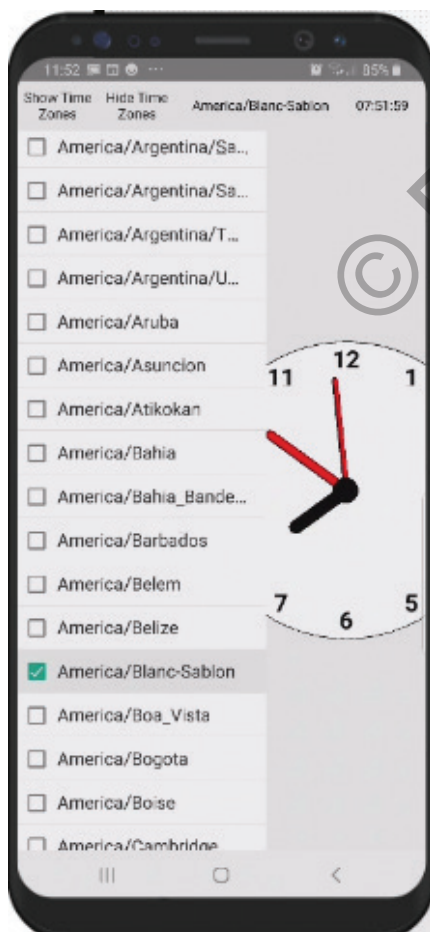


Fig. 42.75 ShowTimeZones clicked

The original settings are restored on switching back to Windows 64-bit, target Windows desktop. This means that the project maintains one set of settings for the Windows target and a different set for the Android target. Delphi does this using the resource directive **\$R** placed at the beginning of the **implementation** section of the unit as shown below for Windows desktop and Android 5" Phone

```
Implementation
{$R *.fmx}
{$R *.LgXhdpiPh.fmx ANDROID}
```

A resource directive is added for each target. [Figure 42.76](#) shows the resource directive list for three different Android targets, an Apple Mac laptop and Windows desktop., which is placed in the **implementation** section of the unit.

```
{ $R *.fmx }
{ $R *.NmXhdpiPh.fmx ANDROID }
{ $R *.LgXhdpiPh.fmx ANDROID }
{ $R *.SmXhdpiPh.fmx ANDROID }
{ $R *.Macintosh.fmx MACOS }
```

Fig. 42.76 Resource directives for different targets

Programming Task

- 2 Build and deploy `AnalogueClockWithTimeZones` to an Android or Apple device.