White Paper

embarcadero®

# Seeing is Believing: Data Visualization in Multi-Device Apps with RAD Studio

Ray Konopka

Embarcadero MVP

October 2013

Americas Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
Level 2
100 Clarence Street
Sydney, NSW 2000
Australia

# Seeing is Believing: Data Visualization in Multi-Device Apps with RAD Studio

Conveying the meaning of data quickly and concisely is a focal point of today's applications. This is especially true for mobile devices where real-estate and user attention are in short supply. Delphi/C++ and the FM Application Platform, along with native code performance, provide the tools necessary to create vibrant, information rich displays leveraging data visualization techniques and infographics. Along with access to standard libraries and 2D/3D graphics support, the extensible FM Application Platform allows developers to reuse their custom data visualizations by creating new FM controls.

Throughout this paper, we will be investigating many of the capabilities of the FM Application Platform involved in data visualization. We will cover the important role that FM primitives and vector graphics play in visualizing data. We will take a look at charting and how we can incorporate modern looking charts into our own applications. We will also be taking a deep dive into customized lists on mobile devices and leveraging custom FM controls to achieve very rich displays. And finally, we will investigate how to create truly stunning hi-res images generated from mathematical data.

## Data Visualization

Before we delve into the specific examples, it is important to have an understanding of just what we mean when we use the term data visualization. As its name suggests, data visualization leverages our visual system for the communication of information. The old adage "a picture is worth a thousand words" is an accurate way to describe what is taking place. That is, the appropriate use of graphics and imagery can convey an enormous amount of information very quickly.

Effective data visualizations do more than just present information. They can inspire new questions and encourage further exploration. They can be used to help identify trends and patterns. Data Visualizations generally fall into two categories: exploratory or explanatory.

Exploratory data visualizations are used in situations when there is no clear understanding of what conclusions can be drawn from the data. These types of visualizations are often interactive in that they allow the reader/viewer to "explore" the data or subsets of the data. A common goal of exploratory visualizations is to identify trends and patterns in the data.

Explanatory data visualizations are used in situations where the author (or designer) has already analyzed the data and is now trying to explain the results. These types of

visualizations are generally static and are often designed in such a way to emphasize the position the author wants to convey.

# Data Visualization Techniques

For many developers, Data Visualization is an advanced, specialized topic that is only required in certain circumstances. However, except for the most trivial of applications, every developer needs to visualize data in their applications. As we proceed through the following list, think of how you might apply these visualization techniques in your own applications.

## Labels, Tables, and Lists

Yes, labels--arguably the most common data visualization technique used in software. Text based tables and lists also fall into this category. These text based visualizations are extremely simple to use and very affective. However, they are rather plain and as the volume increases their effectiveness decreases.

## Charts & Graphs

While text based visualizations may be the most common, it is the Bar Chart that is most often associated with Data Visualization. Bar Charts are very effective at comparing discrete data values within or across categories. The Pie Chart is the visually flashy cousin to the Bar Chart. An effective way to compare parts of a whole with a low degree of precision, care must be taken to ensure that Pie Charts are only used in situations where you are indeed trying to compare parts to a whole. Consider this, every Pie Chart can be converted into a Bar Chart, but not every Bar Chart can be converted into a Pie Chart. Line Graphs are another relative of the Bar Chart. Instead of displaying bars representing a value, just a point is displayed for each value. However, the points in a series are usually connected to show a trend. This makes line graphs very effective in comparing one or more sets of continuous data values.

## Hi-Resolution Imaging

Some datasets are best visualized as an image rather than individual data points. For example, X-Rays, CT Scans, MRIs, etc. Hi-Resolution displays and appropriate use of color provide exceptional detail to two dimensional data.

## Relational Diagrams

For some data sets, there are intrinsic relationships between data values. For these situations, a relational diagram is more appropriate. For example, trees, network diagrams, data flow diagrams, ER diagrams, and flow charts.

### Maps

When dealing with geographic data, it is natural to consider displaying that data on map, especially if the data that is being visualized is related to the spatial properties of the map. For example, population, voting districts, etc. However, just because a dataset contains a geographic element that does not necessary mean that a map is the most effective. For example, sales figures for a company's East Coast, Midwest, and West Coast divisions do not necessary gain much from being displayed on a map of the United States.

## What about Infographics?

When discussing data visualization, it is very common to hear the term *infographics*. In fact, in some contexts, the two terms are used interchangeably. However, there is a growing consensus that there are differences between the two.

Infographics, short for *information graphics*, are typically manually drawn usually with graphics design software. As a result, infographics utilize visual effects to create a visually stunning display. An infographic is concerned with a specific set (or subset) of data values, and only that set of values. Using a different set of data would require manually recreating the presentation.

Data Visualizations, on the other hand, are typically drawn using an automated or algorithmic process that can easily handle varying sets of data values. Data visualizations also generally handle larger volumes of data than an infographic. This makes sense because an automated process can handle large volumes of data much more effectively and a manual process. Data Visualizations sacrifice some of the visual flair of infographics, but they counter that with greater data precision.

## Overview of FM Primitives

Before we move onto the actual examples, there is one more topic that needs to be covered: *FM Primitives*. At the very core of the FM Framework are the FM Primitives, which are vector based controls designed for displaying content. Every style in every control in FM makes use of the FM Primitives. They are truly the building blocks of FM. Figure 1 shows a few of the primitives on an HD form.

The screen shot shows the **TRectangle**, **TLine**, and **TPie** controls in action. Each primitive is responsible for displaying a primitive shape. For shapes that are closed, there is a **Fill** property can controls how the interior of the shape is filled. Likewise, there is a **Stroke** property that defines how the outline of the control is drawn.

The pie pieces are all contained inside a **TLayout** control. Think of the **TLayout** as a **TPanel** control but without the visual representation. It is another core class in the FM

Application Platform. In this example, I am using the layout to group all of the pie pieces together.

The important feature of the FM Primitives with respect to data visualization is that all of the shapes are vector-based and not raster-based. This means that as we adjust the size of the shapes, we do not lose any fidelity in their appearance. This is not true with raster based displays. As a result, if we resize the layout that contains all 4 pie pieces, all of the pie pieces get scaled automatically.

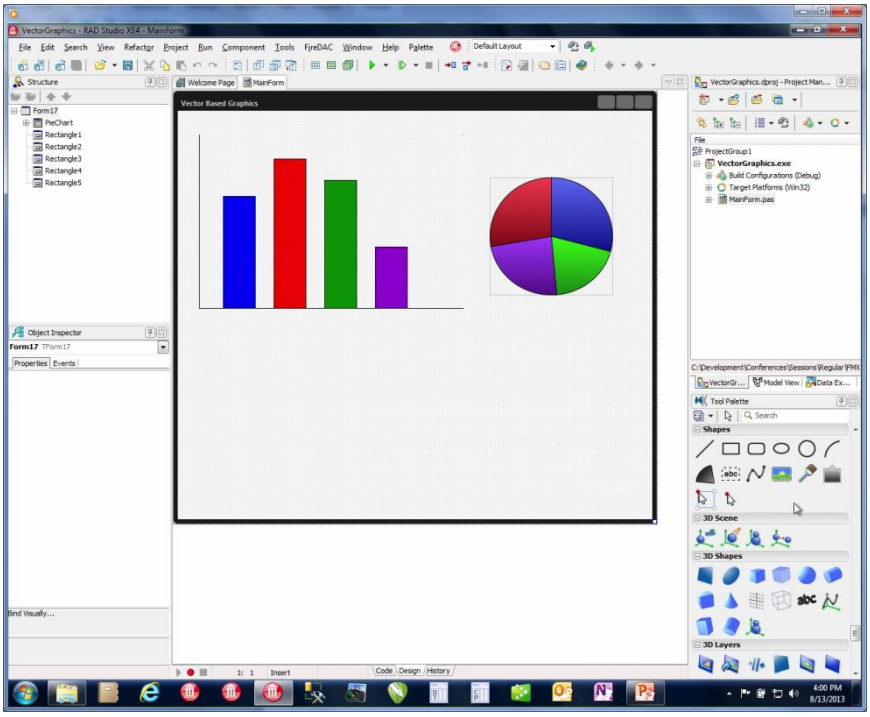


**Figure 1:** The FM Primitives

Another very useful primitive in FM is the **TPath**. The control allows you to specify a custom shape that is defined by a set of points that are connected using curves. Many examples that you see on the web add points to a path programmatically, which, unfortunately, can be cumbersome. Fortunately, it is possible to initialize a path at design-time using an SVG file. However, the steps are not really obvious.

Figure 2 shows the property editor for the **TPath.Data** property. The "Type pathData (like SVG or XAML):" prompt is not very precise and suggests that you could paste the contents of an SVG file into this property editor. However, this will only work if you select and paste a specific portion of the SVG file.

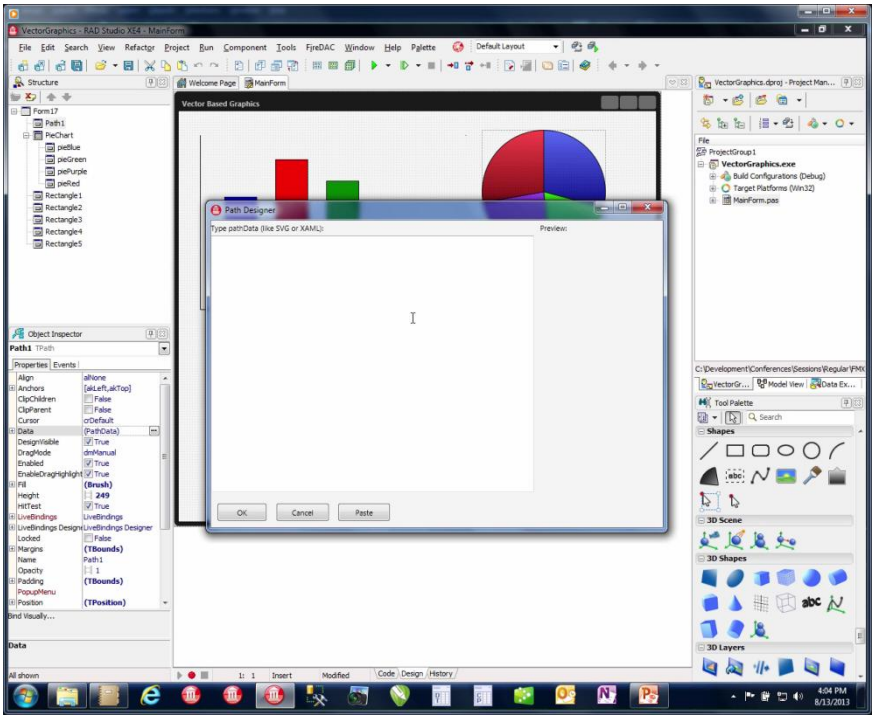


Figure 2: Editing Path Data

Listing 1 shows a sample SVG file created in a vector graphics application, specifically CorelDRAW, but Adobe Illustrator could also be used. To use the data in this SVG file in the TPath control, select just the data elements as shown in the highlighted text. Those elements can then be copied to the clipboard and then pasted into the TPath.Data property editor. Figure 3 shows the results.

**Listing 1:** Sample SVG File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<!-- Creator: CorelDRAW X6 -->
<svg xmlns="http://www.w3.org/2000/svg" xml:space="preserve" width="4.00346in" height="4.00346in"
version="1.1" style="shape-rendering:geometricPrecision; text-rendering:geometricPrecision; image-
rendering:optimizeQuality; fill-rule:evenodd; clip-rule:evenodd"
viewBox="0 0 4003 4003"
 xmlns:xlink="http://www.w3.org/1999/xlink">
 <defs>
  <style type="text/css">
   <![CDATA[
    .fil0 {fill:red}
   ]]>
  </style>
 </defs>
 <g id="Layer_x0020_1">
  <metadata id="CorelCorpID_0Corel-Layer"/>
  <path class="fil0" d="M2002 0c1105,0 2001,896 2001,2002 0,1105 -896,2001 -2001,2001 -1106,0 -2002,-
896 -2002,-2001 0,-1106 896,-2002 2002,-2002zm1552 2002c0,857 -695,1552 -1552,1552 -857,0 -1552,-695 -
1552,-1552l3104 0z"/>
 </g>
</svg>
```

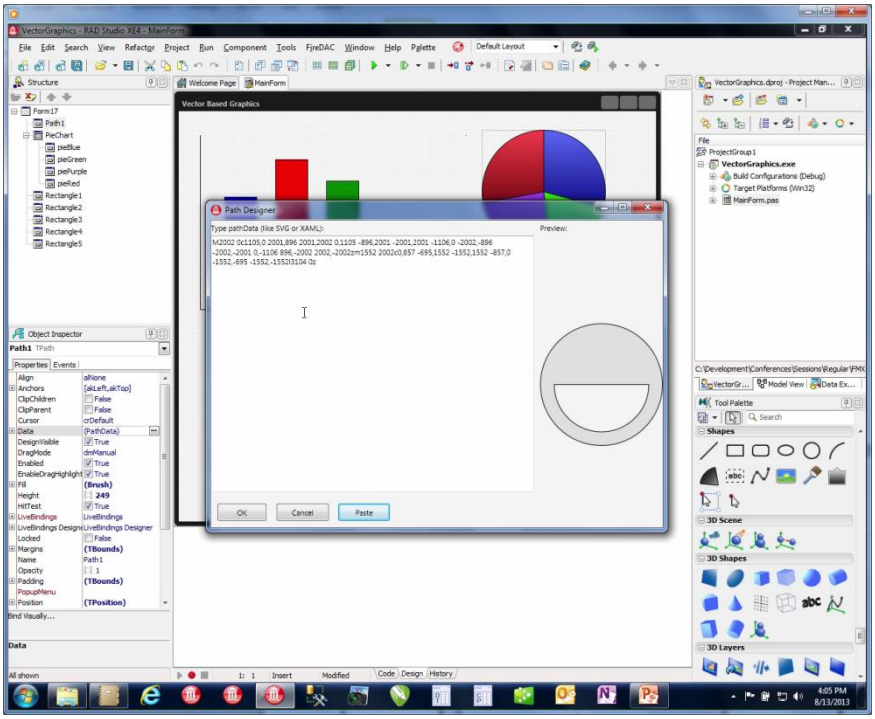

Figure 3: Custom Path Data

In many respects, the primitives in FM are very similar to primitives in vector design tools such as Adobe Illustrator and CorelDRAW. As a result, we can blur the lines between data visualization and infographics by providing enhanced visual effects even in our algorithmically drawn visualizations. We will be leveraging the FM Primitives extensively when we get to the Ratings Mobile Application after the next section.

# CUSTOMIZING THE APPEARANCE OF TCHART

As I noted earlier, charting is a very common form of data visualization. Fortunately, Delphi comes with a **TChart** component, which is part of the **TeeChart** Lite product by Steema Software. Don't be fooled by the "lite" designation. There is a lot of functionality that is available in **TChart**.

Unfortunately, the default appearance of data presented by **TChart** is a bit dated. For example, consider Figure 4. The Bar Chart in the upper right is the default appearance. The chart on the upper left has been customized to provide a more modern appearance. The key to modifying a **TChart** is to select Edit Chart from the context menu. The dialog that is displayed provides many, many options for customizing the appearance.

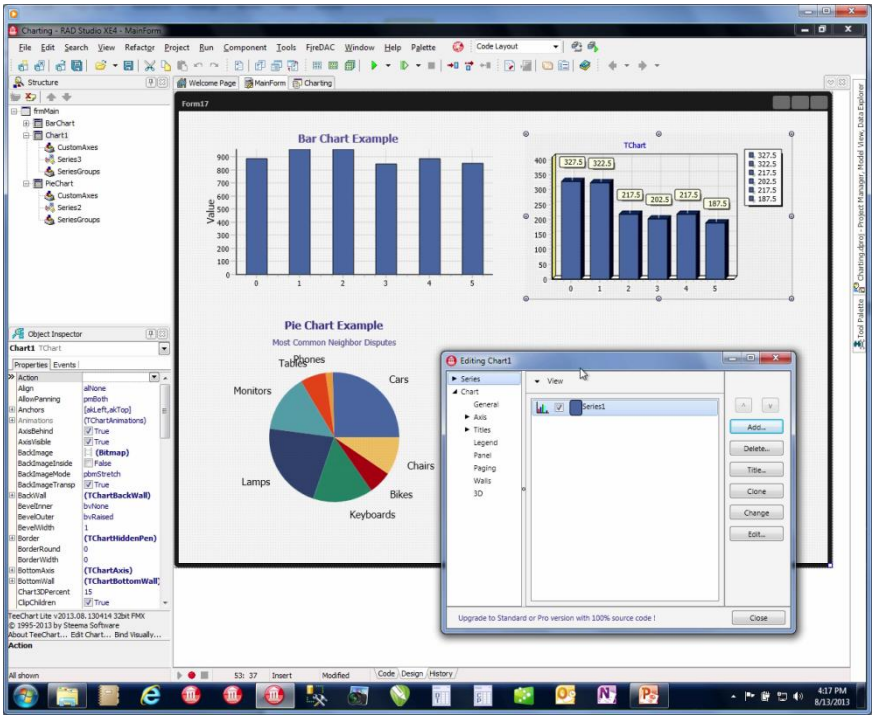


Figure 4: TChart examples

For example, click on the Chart > 3D node in the side tree and then uncheck the "3 Dimensions" check box. Next click on the Chart > Walls node in the side tree and uncheck the "Visible Walls" check box at the top of the editor. Next, to have the background color of the chart match the form, we need to click on Chart > Panel and click the Default check box. To remove the border of the panel, on this same page, click on the Borders tab and select None for the Bevel outer. Click on the Chart > Legend node to modify the legend box. For example, we can hide it. And finally, we can click on the Series > Series N node in the side tree. This allows us to modify various settings related to the data that is being charted. For example to remove the labels that appear above each bar, we simply have to switch to the Marks tab for the selected series and then uncheck the Visible check box.

# Ratings Mobile Application

In this section, we are going to be taking an in depth look at ways we can customize the appearance of items in a list. The goal, of course, is to provide a more effective visualization of the data being presented. That data is going to be product ratings. If you are familiar with Consumer Reports magazine in the United States, then you have undoubtedly seen one of their ratings lists. Consumer Reports reviews and rates all kinds of consumer products, from cars to computers to power drills to food processors. As you can imagine, there is a lot of data that is involved, and much of it is tabular. However, Consumer Reports does not just rely on simple text tables to present results.

# RATINGS 1

The sample app that we will be working with in this section is a mobile Ratings app inspired by Consumer Reports. The actual ratings data we will be using is for Food Processors, but the actual data is not that important. What is important is the method by which we will display that data.
Figure 5 shows the main form for the Ratings1 mobile app for the iPhone. The app simply consists of a title bar (a list box header) and a **TListBox**. When the application starts, the products in the **cdsRatings** client dataset will be loaded into the **lstRatings** list box.

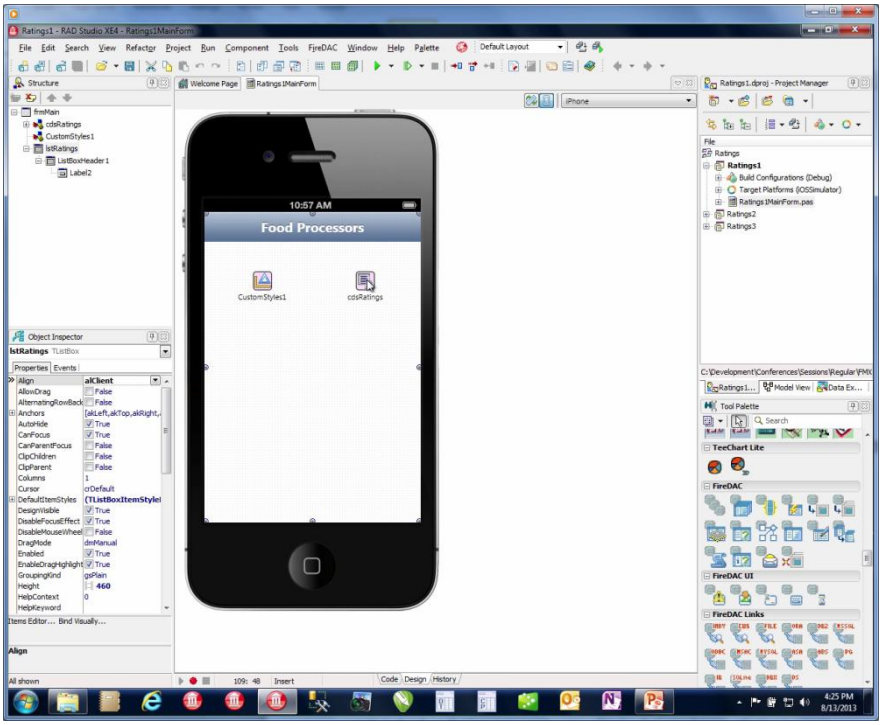


Figure 5: Ratings Mobile App

The default appearance of a **TListBox** is to simply display the text of the item. However, there is a lot more data associated with a product's rating than just a name. In this example, each food processor has a brand and model, cost, an overall score, and individual ratings for chopping, slicing, shredding, pureeing, grating, and noise. In order to display all of this information for each product in the list, we need to create a custom style.

Figure 6 shows the **RatingsHeaderItem** style selected in the Style Designer. The Style Designer is invoked by dropping a **TStyleBook** component onto the form and double clicking the component. The Structure Pane in the IDE is used to navigate between the various controls that are used in the style. In this example, the **TLayout** component is used to manage the positioning of several **TText** controls and other **TLayout** controls.

The actual steps needed to create this custom style are beyond the scope of this paper, but the full source for the application including the custom styles is available on CodeCentral.

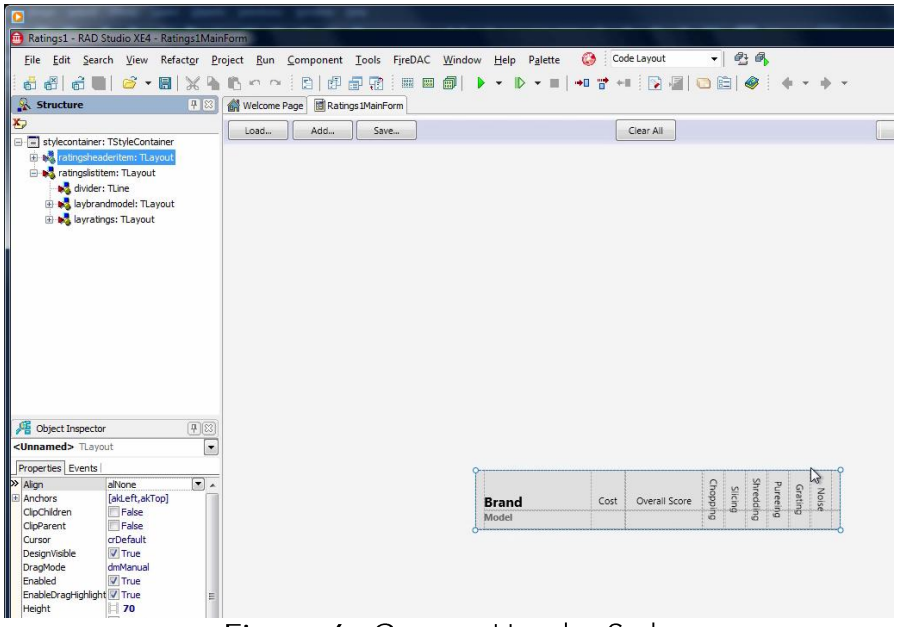


Figure 6: Custom Header Style

You may have noticed that the **RatingsHeaderItem** style is pretty wide and that it will not fit on an iPhone that is in portrait orientation. That's okay, because we are going to adjust the style based on the orientation. When the phone is in portrait mode, we will hide some of the columns and shorten the Overall Score column.

Figure 7 shows the **RatingsListItem** style. The list box will be displaying just one header item (the first one in the list), but the list box will be displaying many items with the **RatingsListItem** style applied. Furthermore, the data values displayed for each list item line up with the titles in the header style. Again, if the phone is oriented in portrait, some of the columns will be hidden.

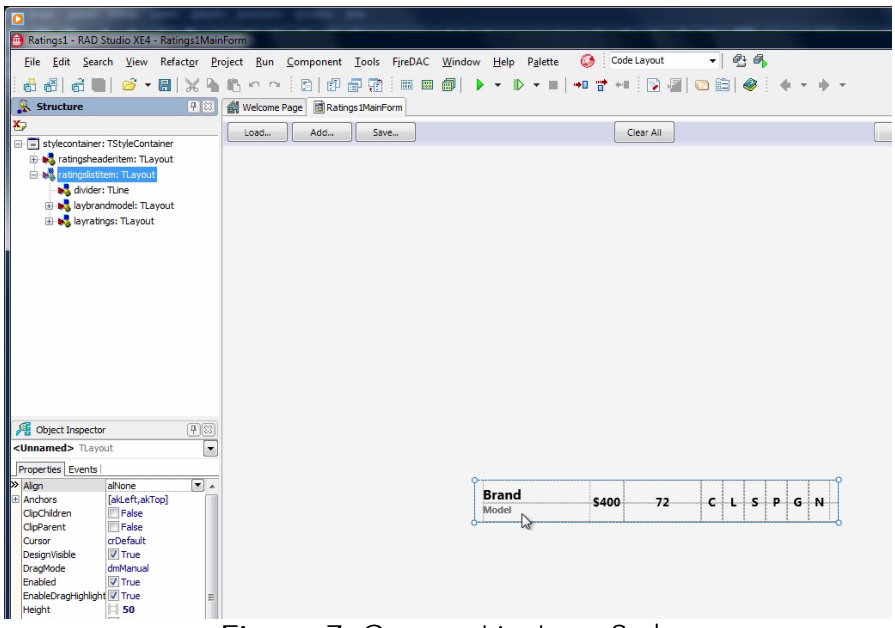


Figure 7: Custom List Item Style

Listing 2 shows the source code for the Ratings1 main form. In FormCreate event handler, LoadRatings is called to populate the list box. LoadRatings first creates a new TListBoxItem to represent the header. After the item is created, the OnApplyStyleLookup event is hooked up to the ListItemApplyStyleLookupHandler method. Then the item's StyleLookup property is set to RatingsHeaderItemStyleName which maps to the StyleName of the custom style we created.

The OnApplyStyleLookup event is handled so that the style can be adjusted based on the orientation of the device. As noted earlier, if the device is in portrait mode, some of the ratings will be hidden.

LoadRatings then iterates through the client dataset and for each record creates a new TListBoxItem and like the header item, assigns the OnApplyStyleLookup event. The item's StyleLookup property is then assigned to RatingsListItemStyleName, which maps to the StyleName of the custom style we created for the items.

In addition, the actual rating information needs to be added to the item. This is done using the StylesData array property of the TListBoxItem. StylesData operates like an array but we index the array with the StyleName of the element in the style we want to modify. StylesData[] returns a TValue, which is a record type that knows how to convert between many different data types. The TFmxObject class defines a Data property that descendant classes can override to provide a generic method of updating an object's primary property value. For example, for text-based controls, the Data property will map to the Text property, but for the TCalendar, Data maps to the Date property.

Listing 2: Ratings1MainForm.pas

```
unit Ratings1MainForm;
```

```
interface

uses
  System.SysUtils,
  System.Types,
  System.UITypes,
  System.Classes,
  System.Variants,
  FMX.Types,
  FMX.Objects,
  FMX.Controls,
  FMX.Forms,
  FMX.Dialogs,
  FMX.Layouts,
  FMX.ListBox,
  FMX.StdCtrls,
  Data.DB,
  Datasnap.DBClient;

type
  TfrmMain = class(TForm)
    lstRatings: TListBox;
    cdsRatings: TClientDataSet;
    CustomStyles1: TStyleBook;
    ListBoxHeader1: TListBoxHeader;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    procedure LoadRatings;
    procedure ListItemApplyStyleLookupHandler( Sender: TObject );

    function FormIsPortrait: Boolean;
  public
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.fmx}

uses
  iOSApi.UIKit;

const
  RatingsListItemStyleName   = 'RatingsListItem';
  RatingsHeaderItemStyleName = 'RatingsHeaderItem';

  txtBrandStyleName          = 'txtBrand';
  txtModelStyleName          = 'txtModel';
  txtCostStyleName           = 'txtCost';
  txtOverallScoreStyleName   = 'txtOverallScore';
  txtChoppingStyleName       = 'txtChopping';
  txtSlicingStyleName        = 'txtSlicing';
  txtShreddingStyleName      = 'txtShredding';
  txtPureeingStyleName       = 'txtPureeing';
  txtGratingStyleName        = 'txtGrating';
  txtNoiseStyleName          = 'txtNoise';

  layOverallScoreStyleName   = 'layOverallScore';
  layChoppingStyleName       = 'layChopping';
  laySlicingStyleName        = 'laySlicing';
  layShreddingStyleName      = 'layShredding';
  layPureeingStyleName       = 'layPureeing';
  layGratingStyleName        = 'layGrating';
```

```
  layNoiseStyleName          = 'layNoise';


procedure TfrmMain.FormCreate(Sender: TObject);
begin
  LoadRatings;
end;


procedure TfrmMain.LoadRatings;
var
  I: Integer;
  Item: TListBoxItem;
begin
  lstRatings.Clear;

  // Add Header Item

  Item := TListBoxItem.Create( lstRatings );
  Item.Parent := lstRatings;
  Item.Height := 70;

  Item.OnApplyStyleLookup := ListItemApplyStyleLookupHandler;
  Item.StyleLookup := RatingsHeaderItemStyleName;

  // Add Product Items

  cdsRatings.First;
  for I := 1 to cdsRatings.RecordCount do
  begin
    Item := TListBoxItem.Create( lstRatings );
    Item.Parent := lstRatings;
    Item.Height := 50;

    Item.OnApplyStyleLookup := ListItemApplyStyleLookupHandler;
    Item.StyleLookup := RatingsListItemStyleName;

    Item.StylesData[ txtBrandStyleName ] := cdsRatings.FieldByName( 'Brand' ).AsString;
    Item.StylesData[ txtModelStyleName ] := cdsRatings.FieldByName( 'Model' ).AsString;
    Item.StylesData[ txtCostStyleName ] := '$' + cdsRatings.FieldByName( 'Cost' ).AsString;
    Item.StylesData[ txtOverallScoreStyleName ] := cdsRatings.FieldByName( 'OverallScore' ).AsInteger;
    Item.StylesData[ txtChoppingStyleName ] := cdsRatings.FieldByName( 'Chopping' ).AsInteger;
    Item.StylesData[ txtSlicingStyleName ] := cdsRatings.FieldByName( 'Slicing' ).AsInteger;
    Item.StylesData[ txtShreddingStyleName ] := cdsRatings.FieldByName( 'Shredding' ).AsInteger;
    Item.StylesData[ txtPureeingStyleName ] := cdsRatings.FieldByName( 'Pureeing' ).AsInteger;
    Item.StylesData[ txtGratingStyleName ] := cdsRatings.FieldByName( 'Grating' ).AsInteger;
    Item.StylesData[ txtNoiseStyleName ] := cdsRatings.FieldByName( 'Noise' ).AsInteger;

    cdsRatings.Next;
  end;
end;


procedure TfrmMain.ListItemApplyStyleLookupHandler( Sender: TObject );
var
  Item: TListBoxItem;
  C: TControl;
  Portrait: Boolean;

  procedure UpdateStyleElement( Item: TListBoxItem; const StyleName: string; HideElement: Boolean );
  var
    C: TControl;
  begin
    C := Item.FindStyleResource( StyleName ) as TControl;
    if C <> nil then
      C.Visible := not HideElement;
  end;
```

```
begin
  Item := TListBoxItem( Sender );

  Portrait := FormIsPortrait;

  C := Item.FindStyleResource( layOverallScoreStyleName ) as TControl;
  if C <> nil then
  begin
    if Portrait then
      C.Width := 60
    else
      C.Width := 90;
  end;


  // Hide all ratings
  UpdateStyleElement( Item, layChoppingStyleName, True );
  UpdateStyleElement( Item, laySlicingStyleName, True );
  UpdateStyleElement( Item, layShreddingStyleName, True );
  UpdateStyleElement( Item, layPureeingStyleName, True );
  UpdateStyleElement( Item, layGratingStyleName, True );
  UpdateStyleElement( Item, layNoiseStyleName, True );

  // Show only those that apply to orientation
  UpdateStyleElement( Item, layNoiseStyleName, Portrait );
  UpdateStyleElement( Item, layGratingStyleName, False );
  UpdateStyleElement( Item, layPureeingStyleName, False );
  UpdateStyleElement( Item, layShreddingStyleName, Portrait );
  UpdateStyleElement( Item, laySlicingStyleName, False );
  UpdateStyleElement( Item, layChoppingStyleName, Portrait );
end;



function TfrmMain.FormIsPortrait: Boolean;
var
  Orientation: Cardinal;
begin
  Orientation := TUIApplication.Wrap(TUIApplication.OCClass.sharedApplication).statusBarOrientation;
  Result := not ( Orientation in [ UIDeviceOrientationLandscapeLeft, UIDeviceOrientationLandscapeRight
] );
end;


procedure TfrmMain.FormResize(Sender: TObject);
begin
  LoadRatings;
end;

end.
```

The ListItemApplyStyleLookupHandler calls the FormIsPortrait helper function, which uses the application's statusBarOrientation setting to determine if the device is in portrait orientation. Next, the individual ratings style elements are updated. Note that all of the elements for the ratings columns are hidden and then reset based on the Portrait setting. This is because the elements are aligned to the right in the item style. And just like the VCL, when you hide and show controls that are aligned, it is possible that their order can get mixed up when you make the control visible again. The code in Listing 2 ensures that the columns remain in their correct order.

Figure 8 shows the Ratings1 app running on the iPhone Simulator. Since the device is in portrait mode, only the Slicing, Pureeing, and Grating ratings are visible and the Overall

Score value is thinner. However, all that we see are numbers, and although the order appears to be based on Overall Score, it is not very clear as to why one processor scored better than others.

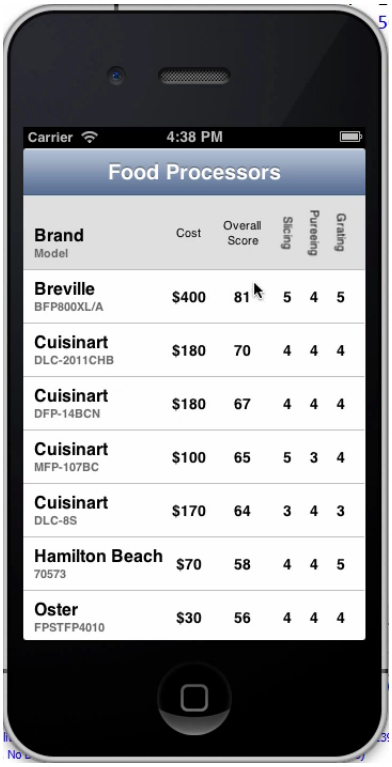


Figure 8: Ratings1 running on the iPhone Simulator

This is the problem with text based visualizations. The table data is very precise, but for the purpose of this application, data precision does not really help the user. Fortunately, we can customize out item style to be more visual and as a result convey much more information more quickly to the user. This is what we do in Ratings2.

## RATINGS 2

Figure 9 shows the Style Designer editing the **TStyleBook** for the Ratings2 application. The **StyleBook** contains several more styles than Ratings1. Specifically, there are separate style entries representing the different ratings levels (1-5). Each rating level is defined using a **TLayout** and each layout contains a **TPath** primitive, which has been loaded with the data from a custom SVG file. The various ratings match the appearance of the Consumer Reports circles that are used for ratings. In addition, rather than display just a number for the overall score, a bar will be displayed.

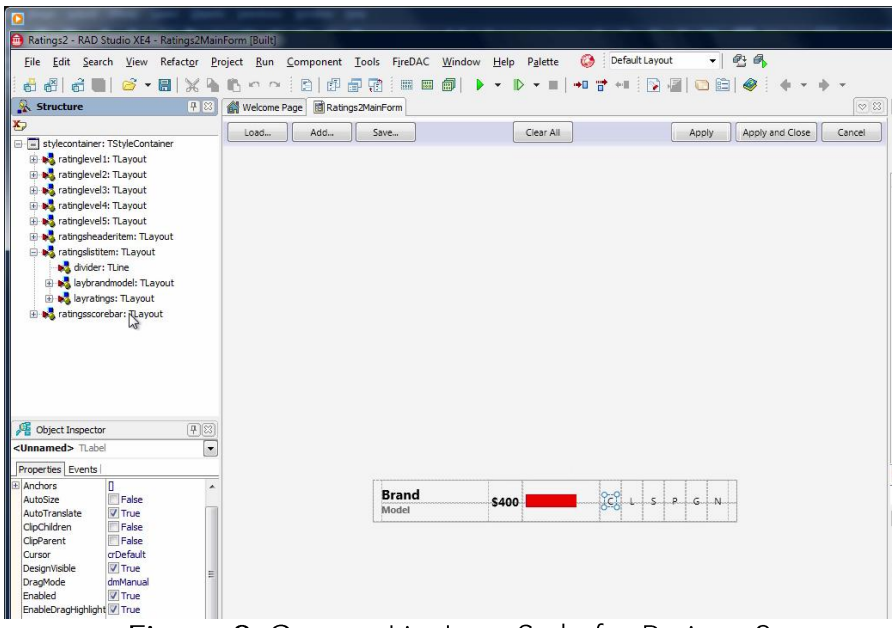


Figure 9: Custom List Item Style for Ratings 2

The challenge in setting up this new style is that in Ratings1, TText controls were used to display the individual ratings values and overall score. However, TText controls cannot be styled. They are primitives. Therefore, for Ratings2, we need to replace the TText controls with TLabel controls, which can be styled.

For the overall score, we use a TProgressBar because that is closer to what we want to display. Inside the StyleBook, we create RatingsScoreBar style, which is defined to match the style elements that are expected to be present by the TProgressBar class. This way, when we set the progress bar's StyleLookup property to the custom RatingsScoreBar style, the control will be displayed correctly.

However, we cannot simply assign the StyleLookup property for the ratings label controls because these will need to change depending on what rating value is actually assigned for that item. Listing 3 shows the source code for the Ratings2MainForm.pas unit. It is very similar to the main form for Ratings1. The notable changes are highlighted in blue in the source listing.

The first change is the list of constants used to identify the style elements. Because the TText controls were changed to TLabel controls, the constant names were updated. Next, the LoadRatings method was modified so that after each rating value was assigned to the appropriate StylesData element, the new UpdateItemRating method is called. UpdateItemRating looks at the Text value of the style element (Data maps to Text in TLabel controls), to determine the correct style to assign to the StyleLookup property. UpdateItemRating is also called at the end of the ListItemApplyStyleLookupHandler method to ensure the correct rating style is used for the values of the item.

## Listing 3: Ratings2MainForm.pas

```
unit Ratings2MainForm;

interface

uses
  System.SysUtils,
  System.Types,
  System.UITypes,
  System.Classes,
  System.Variants,
  FMX.Types,
  FMX.Objects,
  FMX.Controls,
  FMX.Forms,
  FMX.Dialogs,
  FMX.Layouts,
  FMX.ListBox,
  FMX.StdCtrls,
  Data.DB,
  Datasnap.DBClient;

type
  TfrmMain = class(TForm)
    lstRatings: TListBox;
    cdsRatings: TClientDataSet;
    CustomStyles2: TStyleBook;
    ListBoxHeader1: TListBoxHeader;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    procedure LoadRatings;
    procedure ListItemApplyStyleLookupHandler( Sender: TObject );
    procedure UpdateItemRating( Item: TListBoxItem; const StyleName: string );

    function FormIsPortrait: Boolean;
  public
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.fmx}


uses
  iOSApi.UIKit;


const
  RatingsListItemStyleName   = 'RatingsListItem';
  RatingsHeaderItemStyleName = 'RatingsHeaderItem';

  txtBrandStyleName          = 'txtBrand';
  txtModelStyleName          = 'txtModel';
  txtCostStyleName           = 'txtCost';
  pbrOverallScoreStyleName   = 'pbrOverallScore';
  lblChoppingStyleName       = 'lblChopping';
  lblSlicingStyleName        = 'lblSlicing';
  lblShreddingStyleName      = 'lblShredding';
  lblPureeingStyleName       = 'lblPureeing';
  lblGratingStyleName        = 'lblGrating';
  lblNoiseStyleName          = 'lblNoise';
```

```
  layOverallScoreStyleName   = 'layOverallScore';
  layChoppingStyleName       = 'layChopping';
  laySlicingStyleName        = 'laySlicing';
  layShreddingStyleName      = 'layShredding';
  layPureeingStyleName       = 'layPureeing';
  layGratingStyleName        = 'layGrating';
  layNoiseStyleName          = 'layNoise';



procedure TfrmMain.FormCreate(Sender: TObject);
begin
  LoadRatings;
end;


procedure TfrmMain.LoadRatings;
var
  I: Integer;
  Item: TListBoxItem;
  L: TLabel;
begin
  lstRatings.Clear;

  // Add Header Item

  Item := TListBoxItem.Create( lstRatings );
  Item.Parent := lstRatings;
  Item.Height := 70;

  Item.OnApplyStyleLookup := ListItemApplyStyleLookupHandler;
  Item.StyleLookup := RatingsHeaderItemStyleName;

  // Add Product Items

  cdsRatings.First;
  for I := 1 to cdsRatings.RecordCount do
  begin
    Item := TListBoxItem.Create( lstRatings );
    Item.Parent := lstRatings;
    Item.Height := 50;

    Item.OnApplyStyleLookup := ListItemApplyStyleLookupHandler;
    Item.StyleLookup := RatingsListItemStyleName;

    Item.StylesData[ txtBrandStyleName ] := cdsRatings.FieldByName( 'Brand' ).AsString;
    Item.StylesData[ txtModelStyleName ] := cdsRatings.FieldByName( 'Model' ).AsString;
    Item.StylesData[ txtCostStyleName ] := '$' + cdsRatings.FieldByName( 'Cost' ).AsString;
    Item.StylesData[ pbrOverallScoreStyleName ] := cdsRatings.FieldByName( 'OverallScore' ).AsInteger;

    Item.StylesData[ lblChoppingStyleName ] := cdsRatings.FieldByName( 'Chopping' ).AsInteger;
    UpdateItemRating( Item, lblChoppingStyleName );

    Item.StylesData[ lblSlicingStyleName ] := cdsRatings.FieldByName( 'Slicing' ).AsInteger;
    UpdateItemRating( Item, lblSlicingStyleName );

    Item.StylesData[ lblShreddingStyleName ] := cdsRatings.FieldByName( 'Shredding' ).AsInteger;
    UpdateItemRating( Item, lblShreddingStyleName );

    Item.StylesData[ lblPureeingStyleName ] := cdsRatings.FieldByName( 'Pureeing' ).AsInteger;
    UpdateItemRating( Item, lblPureeingStyleName );

    Item.StylesData[ lblGratingStyleName ] := cdsRatings.FieldByName( 'Grating' ).AsInteger;
    UpdateItemRating( Item, lblGratingStyleName );

    Item.StylesData[ lblNoiseStyleName ] := cdsRatings.FieldByName( 'Noise' ).AsInteger;
    UpdateItemRating( Item, lblNoiseStyleName );
```

```
    cdsRatings.Next;
  end;
end;


procedure TfrmMain.ListItemApplyStyleLookupHandler( Sender: TObject );
var
  Item: TListBoxItem;
  C: TControl;
  Portrait: Boolean;

  procedure UpdateStyleElement( Item: TListBoxItem; const StyleName: string; HideElement: Boolean );
  var
    C: TControl;
  begin
    C := Item.FindStyleResource( StyleName ) as TControl;
    if C <> nil then
      C.Visible := not HideElement;
  end;

begin
  Item := TListBoxItem( Sender );

  Portrait := FormIsPortrait;

  C := Item.FindStyleResource( layOverallScoreStyleName ) as TControl;
  if C <> nil then
  begin
    if Portrait then
      C.Width := 60
    else
      C.Width := 90;
  end;


  // Hide all ratings
  UpdateStyleElement( Item, layChoppingStyleName, True );
  UpdateStyleElement( Item, laySlicingStyleName, True );
  UpdateStyleElement( Item, layShreddingStyleName, True );
  UpdateStyleElement( Item, layPureeingStyleName, True );
  UpdateStyleElement( Item, layGratingStyleName, True );
  UpdateStyleElement( Item, layNoiseStyleName, True );

  // Show only those that apply to orientation
  UpdateStyleElement( Item, layNoiseStyleName, Portrait );
  UpdateStyleElement( Item, layGratingStyleName, False );
  UpdateStyleElement( Item, layPureeingStyleName, False );
  UpdateStyleElement( Item, layShreddingStyleName, Portrait );
  UpdateStyleElement( Item, laySlicingStyleName, False );
  UpdateStyleElement( Item, layChoppingStyleName, Portrait );

  // Update Rating Icons

  if Item.StyleLookup = RatingsListItemStyleName then
  begin
    UpdateItemRating( Item, lblChoppingStyleName );
    UpdateItemRating( Item, lblSlicingStyleName );
    UpdateItemRating( Item, lblShreddingStyleName );
    UpdateItemRating( Item, lblPureeingStyleName );
    UpdateItemRating( Item, lblGratingStyleName );
    UpdateItemRating( Item, lblNoiseStyleName );
  end;
end;


procedure TfrmMain.UpdateItemRating( Item: TListBoxItem; const StyleName: string );
var
  L: TLabel;
```

```
begin
  L := Item.FindStyleResource( StyleName ) as TLabel;
  if L <> nil then
    L.StyleLookup := 'RatingLevel' + L.Text;
end;



function TfrmMain.FormIsPortrait: Boolean;
var
  Orientation: Cardinal;
begin
  Orientation := TUIApplication.Wrap(TUIApplication.OCClass.sharedApplication).statusBarOrientation;
  Result := not ( Orientation in [ UIDeviceOrientationLandscapeLeft, UIDeviceOrientationLandscapeRight
] );
end;


procedure TfrmMain.FormResize(Sender: TObject);
begin
  LoadRatings;
  lstRatings.Repaint;
end;

end.
```

Figure 10 shows the Ratings2 app running on the iPhone Simulator. As you can see, the displayed data is much more visual and more importantly, it is much easier to quickly extract information about each item in the list. The use of symbols to convey rating values makes it much easier to compare items.

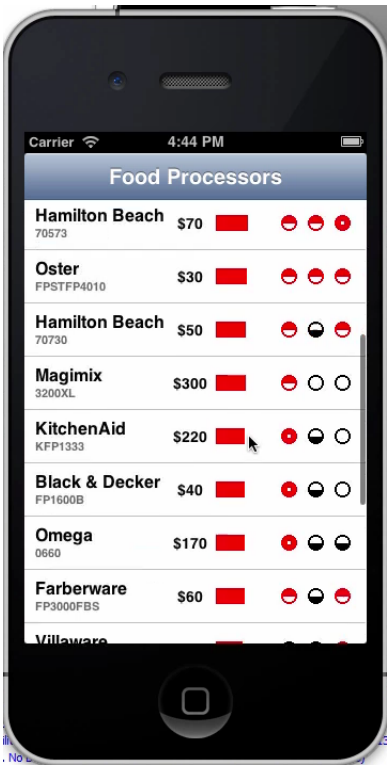


Figure 10: Ratings 2 utilizing Custom List Items

Figure 11 shows the same app in landscape orientation. Notice how the additional ratings columns are visible and the Overall Score column has gotten wider.

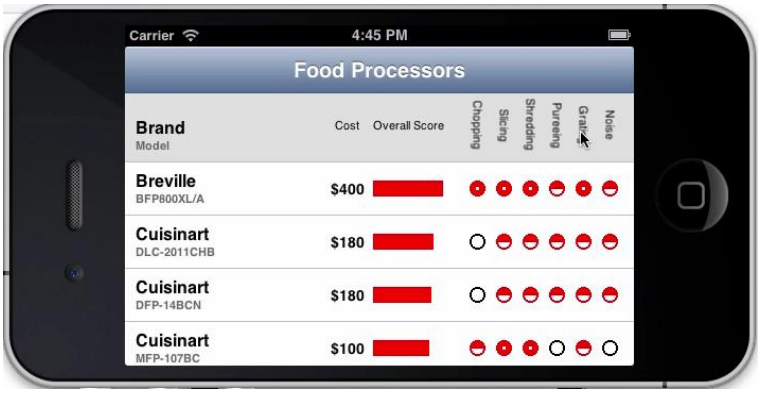


Figure 11: Ratings2 in Landscape showing additional ratings values

Although the display of Ratings2 has greatly improved, there are still some concerns. For example, the overall score just shows a filled bar and does not show the actual value of the score. It would be nice if actual score would be visible. Unfortunately, this is not possible because the TProgressBar does not support displaying text.

Furthermore, the visual effects illustrated in Figures 10 and 11 required a combination of custom styling and custom coding. Neither of which encourage reuse. In Ratings3, we will utilize two custom FM controls that will encapsulate the functionality of the rating icons and the score bar.

## RATINGS 3

Instead of creating custom styles in the StyleBook for the application, Ratings3 utilizes two new custom FM controls in the style RatingsListItem style. Figure 12 shows the immediate impact of this. Instead of TLabel controls displaying text until we change the StyleLookup property at runtime, the new TRkRatingIcon control is used and since it knows how to display the appropriate icon for a rating value, we can see this immediately. The TRkScoreBar is used in place of the TProgressBar and likewise, its display has been customized to what we want.

Both custom controls utilize the style settings that we created before in Ratings2, but rather than be added directly to the application's StyleBook, the styles are contained in the custom control. Not only does the application's StyleBook get simplified, the source code is simplified as well.

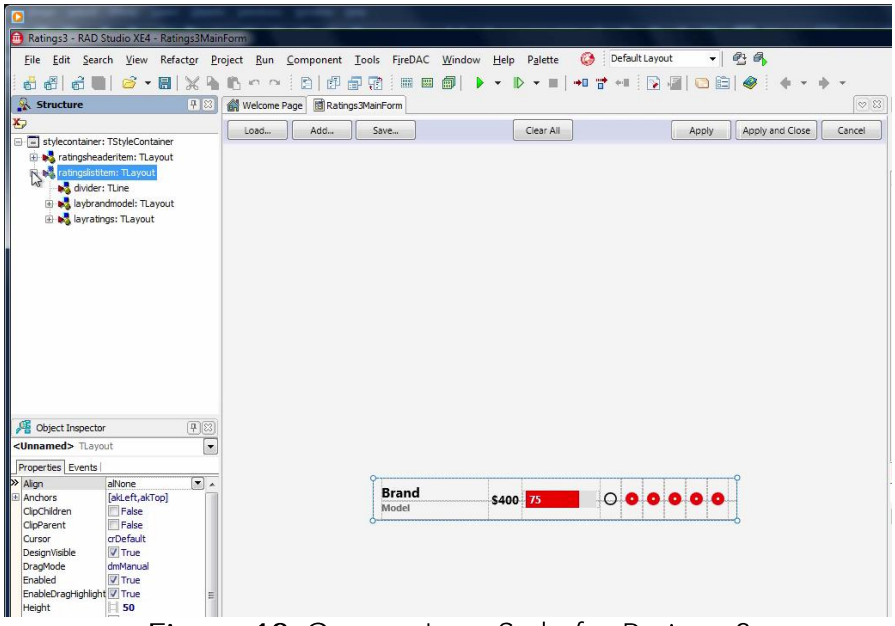


Figure 12: Custom Item Style for Ratings 3

Listing 4 contains the source code for the main form of Ratings3. It is nearly identical to Ratings1 except that the style element names have changed because the **TText** controls for the ratings have been replaced with **TRkRatingIcon** controls, and the **TProgressBar** has been replaced with the **TRkScoreBar**. We no longer need to worry about changing the **StyleLookup** property of the individual ratings icons, because the custom control takes care of that for us.

The **TRkRatingIcon** has a **Rating** property, which the inherited **Data** property maps to, so that using the **StylesData** array property will correctly update the icon. Internally, the **TRkRatingIcon** looks at the Rating property value and uses the appropriate style for its display.

**Listing 4:** Ratings3MainForm.pas

```
unit Ratings3MainForm;

interface

uses
  System.SysUtils,
  System.Types,
  System.UITypes,
  System.Classes,
  System.Variants,
  FMX.Types,
  FMX.Objects,
  FMX.Controls,
  FMX.Forms,
  FMX.Dialogs,
  FMX.Layouts,
  FMX.ListBox,
  FMX.StdCtrls,
  Data.DB,
  Datasnap.DBClient,
```

```
  RkRatings;

type
  TfrmMain = class(TForm)
    lstRatings: TListBox;
    cdsRatings: TClientDataSet;
    CustomStyles3: TStyleBook;
    ListBoxHeader1: TListBoxHeader;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    procedure LoadRatings;
    procedure ListItemApplyStyleLookupHandler( Sender: TObject );

    function FormIsPortrait: Boolean;
  public
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.fmx}


uses
  iOSApi.UIKit;


const
  RatingsListItemStyleName   = 'RatingsListItem';
  RatingsHeaderItemStyleName = 'RatingsHeaderItem';

  txtBrandStyleName          = 'txtBrand';
  txtModelStyleName          = 'txtModel';
  txtCostStyleName           = 'txtCost';
  sbrOverallScoreStyleName   = 'sbrOverallScore';
  icoChoppingStyleName       = 'icoChopping';
  icoSlicingStyleName        = 'icoSlicing';
  icoShreddingStyleName      = 'icoShredding';
  icoPureeingStyleName       = 'icoPureeing';
  icoGratingStyleName        = 'icoGrating';
  icoNoiseStyleName          = 'icoNoise';

  layOverallScoreStyleName   = 'layOverallScore';
  layChoppingStyleName       = 'layChopping';
  laySlicingStyleName        = 'laySlicing';
  layShreddingStyleName      = 'layShredding';
  layPureeingStyleName       = 'layPureeing';
  layGratingStyleName        = 'layGrating';
  layNoiseStyleName          = 'layNoise';



procedure TfrmMain.FormCreate(Sender: TObject);
begin
  LoadRatings;
end;


procedure TfrmMain.LoadRatings;
var
  I: Integer;
  Item: TListBoxItem;
  L: TLabel;
begin
```

```
  lstRatings.Clear;

  // Add Header Item

  Item := TListBoxItem.Create( lstRatings );
  Item.Parent := lstRatings;
  Item.Height := 70;

  Item.OnApplyStyleLookup := ListItemApplyStyleLookupHandler;
  Item.StyleLookup := RatingsHeaderItemStyleName;

  // Add Product Items

  cdsRatings.First;
  for I := 1 to cdsRatings.RecordCount do
  begin
    Item := TListBoxItem.Create( lstRatings );
    Item.Parent := lstRatings;
    Item.Height := 50;

    Item.OnApplyStyleLookup := ListItemApplyStyleLookupHandler;
    Item.StyleLookup := RatingsListItemStyleName;

    Item.StylesData[ txtBrandStyleName ] := cdsRatings.FieldByName( 'Brand' ).AsString;
    Item.StylesData[ txtModelStyleName ] := cdsRatings.FieldByName( 'Model' ).AsString;
    Item.StylesData[ txtCostStyleName ] := '$' + cdsRatings.FieldByName( 'Cost' ).AsString;
    Item.StylesData[ sbrOverallScoreStyleName ] := cdsRatings.FieldByName( 'OverallScore' ).AsInteger;
    Item.StylesData[ icoChoppingStyleName ] := cdsRatings.FieldByName( 'Chopping' ).AsInteger;
    Item.StylesData[ icoSlicingStyleName ] := cdsRatings.FieldByName( 'Slicing' ).AsInteger;
    Item.StylesData[ icoShreddingStyleName ] := cdsRatings.FieldByName( 'Shredding' ).AsInteger;
    Item.StylesData[ icoPureeingStyleName ] := cdsRatings.FieldByName( 'Pureeing' ).AsInteger;
    Item.StylesData[ icoGratingStyleName ] := cdsRatings.FieldByName( 'Grating' ).AsInteger;
    Item.StylesData[ icoNoiseStyleName ] := cdsRatings.FieldByName( 'Noise' ).AsInteger;

    cdsRatings.Next;
  end;
end;


procedure TfrmMain.ListItemApplyStyleLookupHandler( Sender: TObject );
var
  Item: TListBoxItem;
  C: TControl;
  Portrait: Boolean;

  procedure UpdateStyleElement( Item: TListBoxItem; const StyleName: string; HideElement: Boolean );
  var
    C: TControl;
  begin
    C := Item.FindStyleResource( StyleName ) as TControl;
    if C <> nil then
      C.Visible := not HideElement;
  end;

begin
  Item := TListBoxItem( Sender );

  Portrait := FormIsPortrait;

  C := Item.FindStyleResource( layOverallScoreStyleName ) as TControl;
  if C <> nil then
  begin
    if Portrait then
      C.Width := 60
    else
      C.Width := 90;
  end;
```

```
  // Hide all ratings
  UpdateStyleElement( Item, layChoppingStyleName, True );
  UpdateStyleElement( Item, laySlicingStyleName, True );
  UpdateStyleElement( Item, layShreddingStyleName, True );
  UpdateStyleElement( Item, layPureeingStyleName, True );
  UpdateStyleElement( Item, layGratingStyleName, True );
  UpdateStyleElement( Item, layNoiseStyleName, True );

  // Show only those that apply to orientation
  UpdateStyleElement( Item, layNoiseStyleName, Portrait );
  UpdateStyleElement( Item, layGratingStyleName, False );
  UpdateStyleElement( Item, layPureeingStyleName, False );
  UpdateStyleElement( Item, layShreddingStyleName, Portrait );
  UpdateStyleElement( Item, laySlicingStyleName, False );
  UpdateStyleElement( Item, layChoppingStyleName, Portrait );
end;


function TfrmMain.FormIsPortrait: Boolean;
var
  Orientation: Cardinal;
begin
  Orientation := TUIApplication.Wrap(TUIApplication.OCClass.sharedApplication).statusBarOrientation;
  Result := not ( Orientation in [ UIDeviceOrientationLandscapeLeft, UIDeviceOrientationLandscapeRight
] );
end;


procedure TfrmMain.FormResize(Sender: TObject);
begin
  LoadRatings;
  lstRatings.Repaint;
end;


end.
```

Figure 13 shows the Ratings3 application running in the iPhone Simulator. Like Ratings2, it shows icons for the ratings values, but unlike Ratings2, the custom TRkScoreBar allows the Overall Score column to show the actual score along with a visual bar. Plus the use of a background color for the score bar makes it much easier for users to visually compare bars.

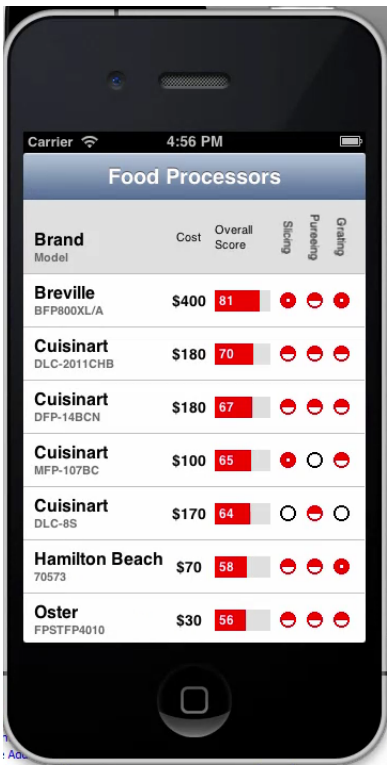


Figure 13: Ratings 3 Utilizing Custom Ratings Components

# MANDELFIRE

Back in the 1980s, as an undergrad, I did a research project on the Mandelbrot Set. As a Computer Science and Mathematics major, it was a very interesting project. The Mandelbrot set is a set of complex numbers which remain bounded when applied to a particular complex polynomial. The actual math involved in the Mandelbrot set is beyond the scope of this paper, but the representation of that set is definitely of interest. Complex numbers (a + bi) are often represented in a two-dimensional plane where the real part of the number is the x-coordinate, and the imaginary part is the y-coordinate. Using this approach, it is possible to visualize the Mandelbrot set.

The MandelFire application is designed to explore the Mandelbrot Set using the FM Application Platform. The MandelFire project group contains a Desktop application with Win32, Win64, and OSX targets. The project group also contains a Mobile application with iOS Simulator, and iOS Device targets.

Figures 14 and 15 show the main forms for the desktop and mobile applications, respectively. Aside from a title bar used in the mobile version, but forms utilize a TPaintBox to display the Mandelbrot Set. It should be noted that both forms are nearly identical in their layout and even their source code. However, in practice, it is usually better to use a layout that is optimized for a mobile device and a separate one for the

desktop. The key will be to minimize any duplication of code, which is exactly what we will focus on in MandelFire.

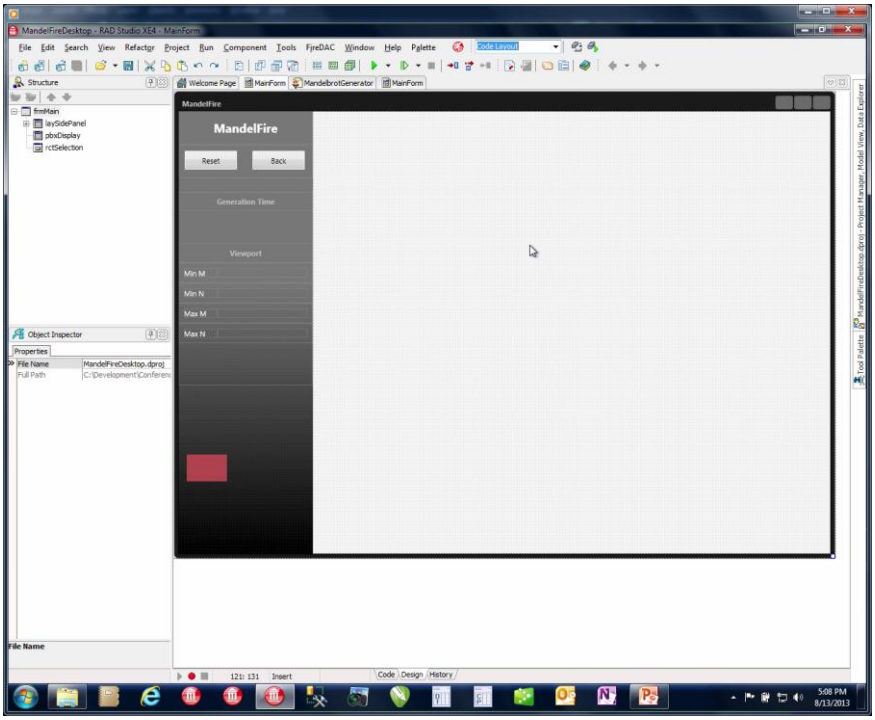


Figure 14: Main form for Desktop version of MandelFire

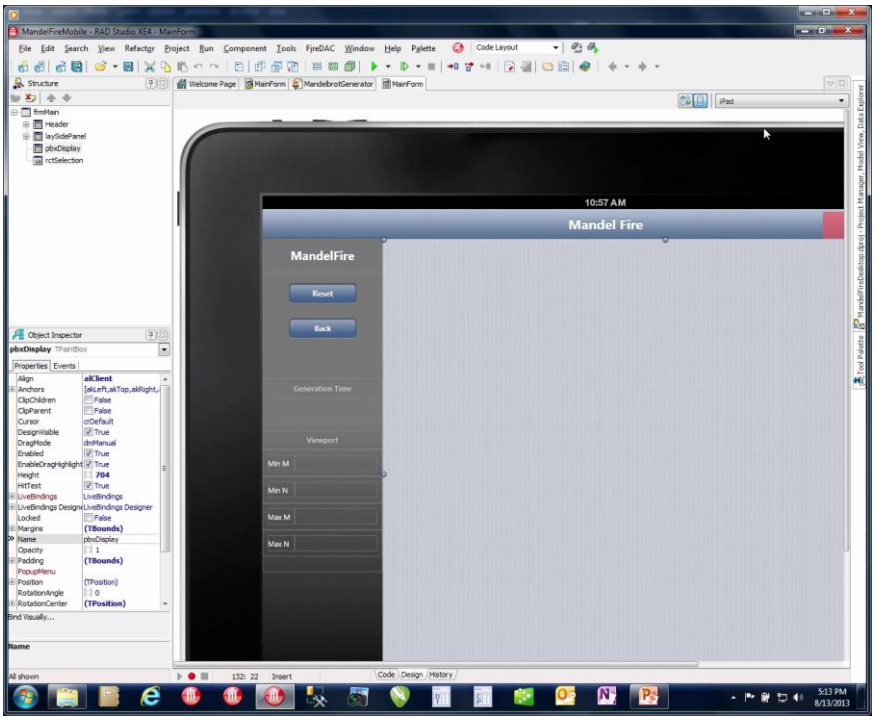


Figure 15: Main form for Mobile version of MandelFire

Listing 5 shows the source code for the main form for the mobile version. The main form creates an instance of the TMandelbrotGenerator, a reusable class that will take care of all the math and generation of the image for the specified viewport. A viewport represents a window in the complex plane that defines the bounds of the calculations. The viewport is adjusted to zoom into specific regions of the set.

The key method in the main form is the pbxDisplayPaint event, which is where the bitmap image created by the FMandelbrotGenerator is displayed on the screen. The Canvas parameter to the OnPaint event handler is what we will use to display the image. However, even though the Canvas parameter is of type TCanvas, this is the FM TCanvas and not the VCL TCanvas. They are quite different under the hood.

Whenever we wish to display something on a canvas in FM, we need to call the BeginScene method first, and when we are finished, EndScene. In MandelFire, we simply call Canvas.DrawBitmap to draw the bitmap that was created by the FMandelbrotGenerator.

As an aside, it is nice to display the time it takes to generate the Mandelbrot Set. Generating the set is computationally intensive and back in the 80s it was not uncommon to take more than 20 minutes to generate a display. With today's machines, even the mobile ones, it takes only a few seconds. Since this application targets multiple platforms, we need a way to perform timings that are platform independent. This is where the TStopWatch class comes in. It is part of the RTL and can be used to record time durations. What is nice is that the implementation details vary depending on the platform, so for on Windows machines, the QueryPerformanceCounter function is used, but on Mac, the mach_absolute_time value is used.

**Listing 5:** MainForm.pas

```
unit MainForm;

interface

uses
  System.SysUtils,
  System.Types,
  System.UITypes,
  System.Classes,
  System.Variants,
  System.Diagnostics,
  FMX.Types,
  FMX.Controls,
  FMX.Forms,
  FMX.StdCtrls,
  FMX.Objects,
  FMX.Layouts,
  MandelbrotGenerator;

type
  TfrmMain = class(TForm)
    Header: TToolBar;
    HeaderLabel: TLabel;
    rctSelection: TRectangle;
    pbxDisplay: TPaintBox;
```

```
    laySidePanel: TLayout;
    Rectangle1: TRectangle;
    txtTitle: TText;
    Layout1: TLayout;
    Text1: TText;
    Layout2: TLayout;
    lblMinM: TText;
    txtMinM: TText;
    Layout3: TLayout;
    Text2: TText;
    txtMaxN: TText;
    Layout4: TLayout;
    Text4: TText;
    txtMaxM: TText;
    Layout5: TLayout;
    Text6: TText;
    txtMinN: TText;
    Text3: TText;
    lblGenerationTime: TLabel;
    Layout6: TLayout;
    btnBack: TButton;
    btnReset: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure pbxDisplayPaint(Sender: TObject; Canvas: TCanvas);
    procedure pbxDisplayMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Single);
    procedure pbxDisplayMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Single);
    procedure pbxDisplayMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Single);
    procedure btnResetClick(Sender: TObject);
  private
    FGenerator: TMandelbrotGenerator;

    FGeneratingSet: Boolean;
    FZoomStartX: Single;
    FZoomStartY: Single;

    FStopWatch: TStopWatch;

    procedure GenerateMandelbrotSet;
    procedure UpdateViewportStatus;
  public
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.fmx}


{======================}
{== TfrmMain Methods ==}
{======================}


procedure TfrmMain.FormCreate(Sender: TObject);
begin
  FStopWatch := TStopWatch.Create;

  FGenerator := TMandelbrotGenerator.Create( Trunc( pbxDisplay.Width ), Trunc( pbxDisplay.Height ) );

  GenerateMandelbrotSet;
end;
```

```
procedure TfrmMain.FormDestroy(Sender: TObject);
begin
  FGenerator.Free;
end;


procedure TfrmMain.GenerateMandelbrotSet;
begin
  UpdateViewportStatus;
  FGeneratingSet := True;
  try
    FStopWatch.Reset;
    FStopWatch.Start;

    FGenerator.Generate;

    FStopWatch.Stop;
  finally
    FGeneratingSet := False;
  end;

  lblGenerationTime.Text := Format( '%d ms', [ FStopWatch.ElapsedMilliseconds ] );

  pbxDisplay.Repaint;
end;



procedure TfrmMain.pbxDisplayPaint(Sender: TObject; Canvas: TCanvas);
var
  R: TRectF;
begin
  if FGeneratingSet then
    Exit;

  R := RectF( 0, 0, FGenerator.Width, FGenerator.Height );

  if Canvas.BeginScene then
  try
    Canvas.DrawBitmap( FGenerator.Bitmap, R, R, 1.0 );
  finally
    Canvas.EndScene;
  end;
end;



{== Zooming ==}

procedure TfrmMain.pbxDisplayMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Single);
begin
  FZoomStartX := X;
  FZoomStartY := Y;
end;


procedure TfrmMain.pbxDisplayMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Single);
var
  DeltaX, DeltaY: Single;
begin
  if ssLeft in Shift then
  begin
    DeltaX := X - FZoomStartX;
    DeltaY := 5 * DeltaX / 6;

    if ( DeltaX > 0 ) and ( DeltaY > 0 ) then
    begin
```

```
      rctSelection.SetBounds( pbxDisplay.Position.X + FZoomStartX, pbxDisplay.Position.Y + FZoomStartY,
DeltaX, DeltaY );
      rctSelection.Visible := True;
    end;
  end;
end;


procedure TfrmMain.pbxDisplayMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Single);
begin
  rctSelection.Visible := False;

  if FGenerator.Zoom( FZoomStartX, FZoomStartY, X, Y ) then
    GenerateMandelbrotSet;
end;


procedure TfrmMain.btnResetClick(Sender: TObject);
begin
  FGenerator.ResetViewport;
  GenerateMandelbrotSet;
end;


procedure TfrmMain.UpdateViewportStatus;
begin
  txtMinM.Text := Format( '%.18f', [ FGenerator.Viewport.MinM ] );
  txtMinN.Text := Format( '%.18f', [ FGenerator.Viewport.MinM ] );
  txtMaxM.Text := Format( '%.18f', [ FGenerator.Viewport.MaxM ] );
  txtMaxN.Text := Format( '%.18f', [ FGenerator.Viewport.MaxN ] );
end;

end.
```

Listing 6 shows the source code for the MandelbrotGenerator unit. The unit contains two classes: TMandelbrotViewport and TMandelbrotGenerator. The TMandelbrotViewport is a simple class to manage the boundaries used for calculations in the complex plane. The TMandelbrotGenerator class is responsible for performing the necessary calculations and generating a hi-resolution image representing the current viewport.
The Mandelbrot Set is generated by taking each point in the current viewport and plugging it into a calculation and monitoring the results and then repeating if necessary. The number of iterations that are used for a given point is used to colorize the pixel representing that point.

This leads to two important tasks that the TMandelbrotGenerator must handle. The first is the generation of a color palette. With any hi-resolution imaging visualization, the choice of color palette is crucial. The second task is the ability to manipulate the individual pixels of the bitmap to be generated.

**Listing 6:** MandelbrotGenerator.pas

```
unit MandelbrotGenerator;

interface

uses
  System.Types,
```

```
  System.UITypes,
  FMX.Types,
  FMX.Colors;

type
  TMandelbrotViewport = class
  private
    FMinM: Double;
    FMaxM: Double;
    FMinN: Double;
    FMaxN: Double;
  public
    constructor Create;
    procedure Initialize;

    procedure SetBounds( MinM, MinN, MaxM, MaxN: Double );

    property MinM: Double
      read FMinM
      write FMinM;

    property MaxM: Double
      read FMaxM
      write FMaxM;

    property MinN: Double
      read FMinN
      write FMinN;

    property MaxN: Double
      read FMaxN
      write FMaxN;
  end;


  TMandelbrotGenerator = class
  private
    FWidth: Integer;
    FHeight: Integer;

    FBitmap: TBitmap;

    FViewport: TMandelbrotViewport;
    FMaxIterations: Integer;
    FMaxColors: Integer;

    FColorPalette: array of TAlphaColor;

    procedure InitializeColorPalette;
    procedure InitializeGradient( G: TGradient );

  public
    constructor Create( Width, Height: Integer );
    destructor Destroy; override;

    procedure Generate;

    procedure ResetViewport;
    function Zoom( OrigX, OrigY, X, Y: Single ): Boolean;

    property Bitmap: TBitmap
      read FBitmap;

    property Width: Integer
      read FWidth;

    property Height: Integer
      read FHeight;
```

```
    property Viewport: TMandelbrotViewport
      read FViewport;
  end;



implementation

uses
  System.SysUtils,
  System.UIConsts,
  FMX.PixelFormats;


{================================}
{== TMandelbrotViewport Methods ==}
{================================}

constructor TMandelbrotViewport.Create;
begin
  inherited;
  Initialize;
end;


procedure TMandelbrotViewport.Initialize;
begin
  FMinM := -2.25;
  FMaxM := 0.75;

  FMinN := -1.25;
  FMaxN := 1.25;
end;


procedure TMandelbrotViewport.SetBounds( MinM, MinN, MaxM, MaxN: Double );
begin
  FMinM := MinM;
  FMaxM := MaxM;

  FMinN := MinN;
  FMaxN := MaxN;
end;



{================================}
{== TMandelbrotGenerator Methods ==}
{================================}

constructor TMandelbrotGenerator.Create( Width, Height: Integer );
begin
  FWidth := Width;
  FHeight := Height;

  FBitmap := TBitmap.Create( FWidth, FHeight );

  FMaxIterations := 512;

  InitializeColorPalette;
  FViewport := TMandelbrotViewport.Create;
end;


destructor TMandelbrotGenerator.Destroy;
begin
```

```
  FViewport.Free;
  FBitmap.Free;
  inherited;
end;


procedure TMandelbrotGenerator.ResetViewport;
begin
  FViewport.Initialize;
end;


function TMandelbrotGenerator.Zoom( OrigX, OrigY, X, Y: Single ): Boolean;
var
  DeltaX, DeltaY: Single;
  W, H: Double;
begin
  DeltaX := X - OrigX;
  DeltaY := 5 * DeltaX / 6;

  if DeltaX > 3 then
  begin
    X := OrigX + DeltaX;
    Y := OrigY + DeltaY;

    W := FViewport.MaxM - FViewport.MinM;
    FViewport.MinM := FViewport.MinM + OrigX * W / FBitmap.Width;
    FViewport.MaxM := FViewport.MaxM - ( FBitmap.Width - X ) * W / FBitmap.Width;

    H := FViewport.MaxN - FViewport.MinN;
    FViewport.MinN := FViewport.MinN + ( FBitmap.Height - Y ) * H / FBitmap.Height;
    FViewport.MaxN := FViewport.MaxN - OrigY * H / FBitmap.Height;

    Result := True;
  end
  else
    Result := False;
end;


procedure TMandelbrotGenerator.Generate;
var
  X, Y, I, K: Integer;
  DeltaX, DeltaY, NewM, NewN: Double;
  M, N, A, B: Double;
  BitmapData: TBitmapData;
begin
  DeltaX := ( FViewport.MaxM - FViewport.MinM ) / FBitmap.Width;
  DeltaY := ( FViewport.MaxN - FViewport.MinN ) / FBitmap.Height;

  if FBitmap.Map( TMapAccess.maReadWrite, BitmapData ) then
  try
    for X := 0 to FBitmap.Width - 1 do
    begin
      for Y := 0 to FBitmap.Height - 1 do
      begin

        A := FViewport.MinM + DeltaX * X;
        B := FViewport.MaxN - DeltaY * Y;
        M := 0;
        N := 0;
        K := 0;

        for I := 1 to FMaxIterations do
        begin
          if ( M * M + N * N ) > 4 then
            Break;
```

```
          NewM := M * M - N * N + A;
          NewN := 2 * M * N + B;
          M := NewM;
          N := NewN;
          Inc( K );
        end;

        if K < FMaxIterations then
          K := K mod FMaxColors
        else
          K := 0;

        BitmapData.SetPixel( X, Y, FColorPalette[ K ] );
      end;
    end;
  finally
    FBitmap.Unmap( BitmapData );
  end;
end;



procedure TMandelbrotGenerator.InitializeColorPalette;
var
  G: TGradient;
  Bmp: TBitmap;
  BmpData: TBitmapData;
  R: TRectF;
  H: Integer;
begin
  FMaxColors := 128;
  H := 10;                            // H must be > 8 otherwise, in iOS, GetScanline does not work
correctly

  Bmp := TBitmap.Create( FMaxColors, H );
  R := RectF( 0, 0, FMaxColors, H );

  G := TGradient.Create;
  try
    InitializeGradient( G );


    // Fill the TBitmap Bmp using the TGradient G defined above

    Bmp.Canvas.BeginScene;
    try
      Bmp.Canvas.Fill.Gradient := G;
      Bmp.Canvas.Fill.Kind := TBrushKind.bkGradient;
      Bmp.Canvas.FillRect( R, 0.0, 0.0, AllCorners, 1.0 );
    finally
      Bmp.Canvas.EndScene;
    end;

    // Use the Scanline from the Bmp to populate the FColorPalette

    SetLength( FColorPalette, FMaxColors );

    if Bmp.Map( TMapAccess.maRead, BmpData ) then
    try
      ScanlineToAlphaColor( BmpData.GetScanline( 0 ), FColorPalette, FMaxColors, Bmp.PixelFormat );
      FColorPalette[ 0 ] := TAlphaColors.Black;
    finally
      Bmp.Unmap( BmpData );
    end;


  finally
    G.Free;
```

```
    Bmp.Free;
  end;
end;


procedure TMandelbrotGenerator.InitializeGradient( G: TGradient );
var
  GP: TGradientPoint;
begin
  G.Points.Clear;

  // Add new points for each color change in the gradient

  {$IFNDEF IOS}

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 5, 5, 80 );
  GP.Offset := 0.0;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 85, 167, 249 );
  GP.Offset := 0.08;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 255, 255, 215 );
  GP.Offset := 0.25;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 255, 235, 80 );
  GP.Offset := 0.38;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 255, 155, 0 );
  GP.Offset := 0.53;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 150, 0, 0 );
  GP.Offset := 0.75;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 5, 5, 80 );
  GP.Offset := 1.0;

  {$ELSE}

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 5, 5, 80 );
  GP.Offset := 0.0;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 85, 167, 249 );
  GP.Offset := 0.33;


  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 255, 235, 80 );
  GP.Offset := 0.66;

  GP := TGradientPoint( G.Points.Add );
  GP.Color := MakeColor( 150, 0, 0 );
  GP.Offset := 1.0;

  {$ENDIF}

  // Use StartPosition and StopPosition to specify the gradient angle

  G.StartPosition.Point := PointF( 0, 0.5 );
  G.StopPosition.Point := PointF( 1, 0.5 );
```

```
end;

end.
```

The color palette is generated in the **InitializeColorPalette** method and uses the built-in gradient support of FM to create a visually stunning color palette. The technique that is used is to create a temporary bitmap and then use the **TGradient** to fill that bitmap. Then we can take the individual colors of the pixels in the bitmap. There are a couple notes regarding the **TGradient** class in FM. First, the height of the temporary bitmap must be greater than 8. If not, on an iOS device, the extracting of the pixel color values will not work. The second is that on iOS devices, TGradient only supports 4 color points. This is illustrated in the **InitializeGradient** method.

Both the **InitializeColorPalette** method and the Generate method need to access the individual pixels of their respective **TBitmap** objects. However, unlike the VCL, the **TBitmap** class in FM does not provide access to the pixels. In order to access the individual pixels, we need to use a **TBitmapData** object. To obtain a **TBitmapData** object for a given bitmap, we use the **TBitmap.Map** function. With the **TBitmapData** object, when then have access to methods such as **SetPixel** and **GetScanline**. When we are done manipulating the bitmap, we need to call **TBitmap.Unmap**.

Figures 16 through 19 show the MandelFire application running on desktop and mobile devices. They also illustrate the effects of zooming into a particular region of the complex plane to generate a new fractal image.

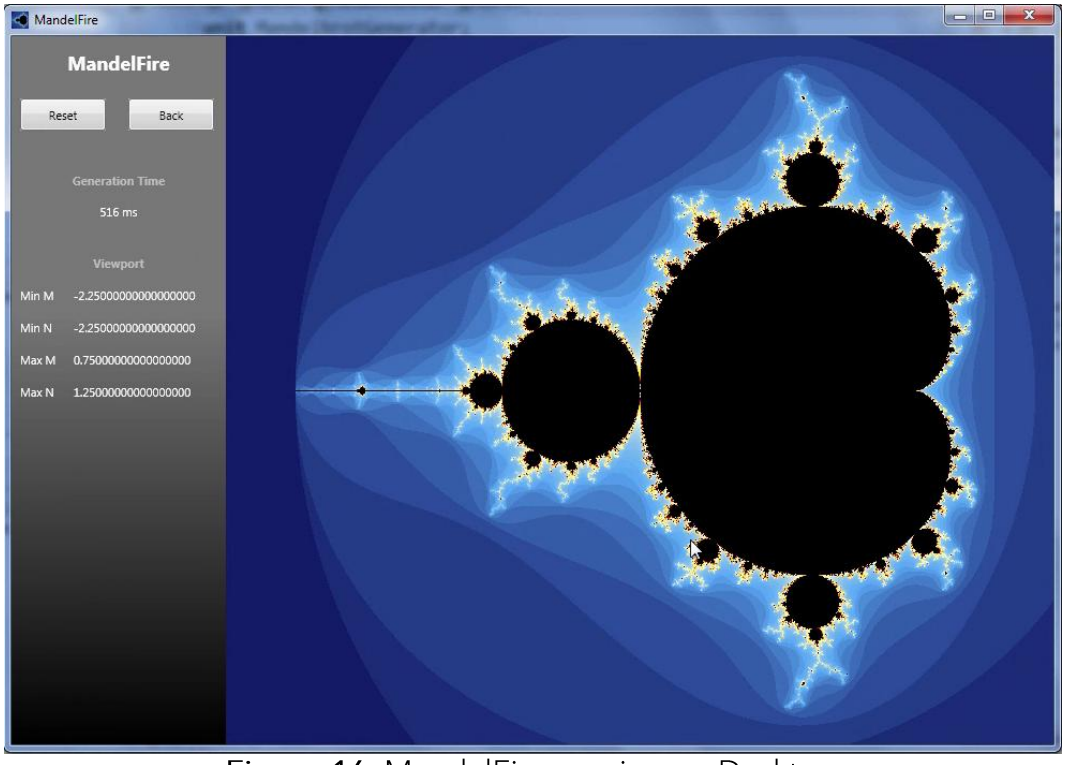


Figure 16: MandelFire running on Desktop

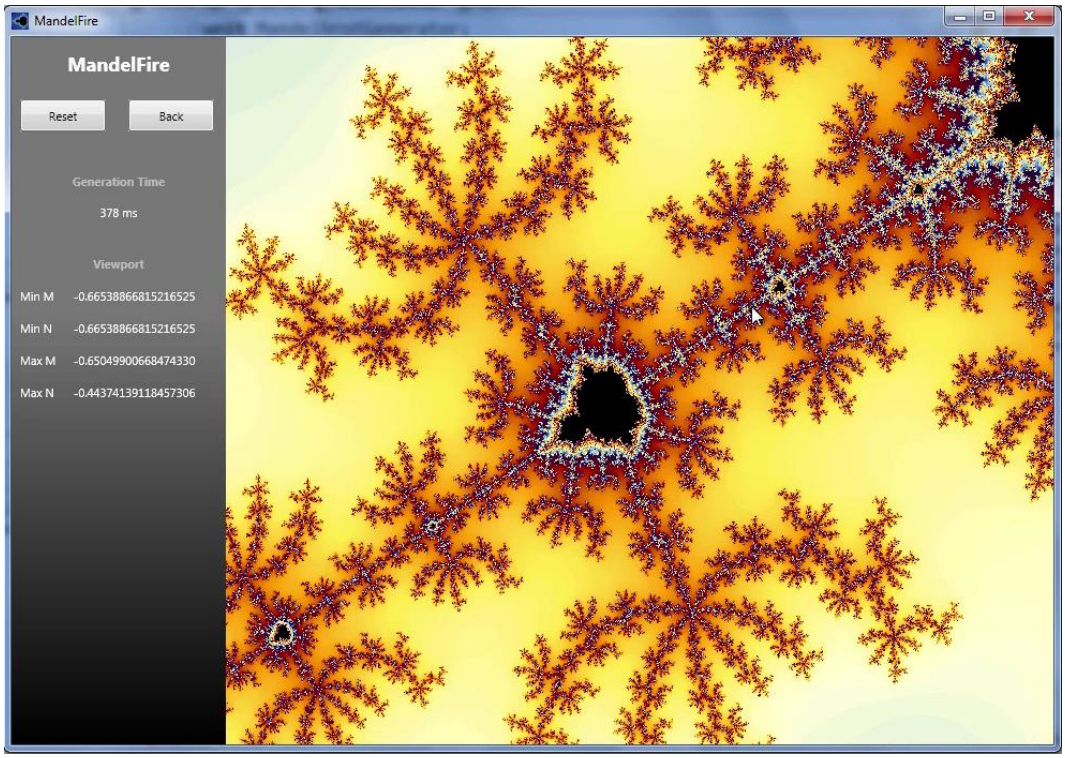


Figure 17: Zooming in on Desktop MandelFire

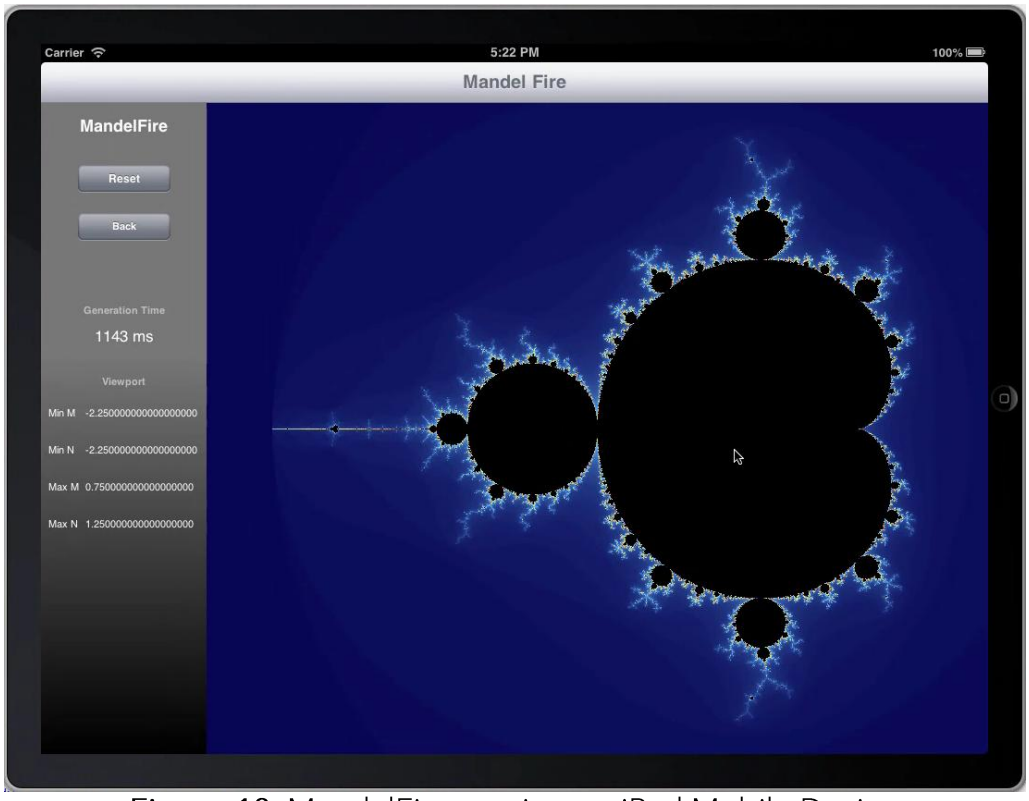


Figure 18: MandelFire running on iPad Mobile Device

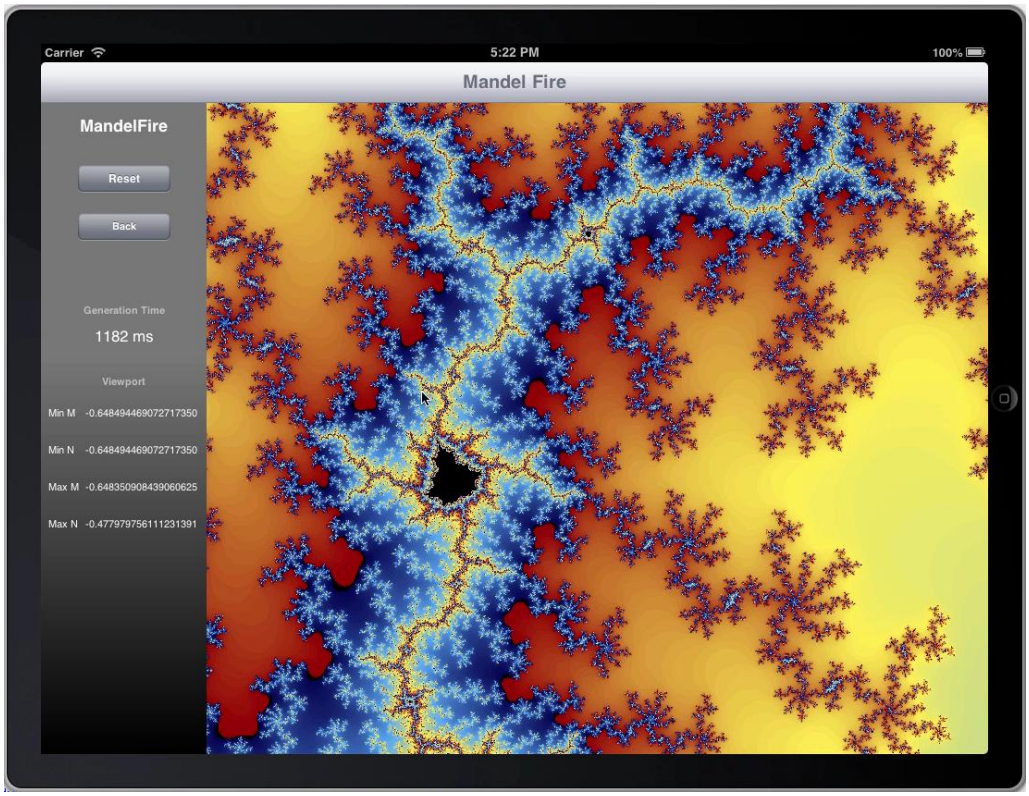


Figure 19: Zooming in on mobile MandelFire.

# RESOURCES

## SOURCE CODE

Source code for some of the components and projects mentioned in this paper are available on CodeCentral.

FM Controls Source Code http://cc.embarcadero.com/item/29517

Data Visualization Sample Apps Source Code http://cc.embarcadero.com/item/29516

## WEBINAR

The companion webinar with video demonstrations of the content included in this paper is available to view on demand at http://forms.embarcadero.com/DataVisualizationWebinar8-14.

## Books

There are many books available that cover various aspects of data visualization. The following list is notable in that the collection covers a wide variety of topics. *Designing Data Visualizations* is a great book for those starting out. It is a concise book that gets right to the point and provided a solid foundation of terminology and concepts that are involved with visualizing data. Visual Explanations is a classic read by a world-renowned data visualization expert. The book covers a wide variety of subjects and is not limited to just computer generated visualizations. Stephen Few's books are loaded with many practical examples.

Designing Data Visualizations
Noah Iliinsky & Julie Steele

Visual Explanations
Edward R. Tufte

Show Me the Numbers (Designing Tables and Graphs to Enlighten)
Stephen Few

Information Dashboard Design: Displaying Data for at-a-glance Monitoring
Stephen Few

## Websites

A simple web search shows that there are many sites out there dedicated to data visualization and infographics. The following list is a short sample of some of the most interesting.

http://visualization.geblogs.com

http://infosthetics.com

http://visual.ly/learn

http://flowingdata.com

http://www.informationisbeautiful.net

http://visualisingdata.com

http://www.census.gov/dataviz

# About The Author

Ray Konopka is the creator of Raize Components and CodeSite, the award winning products from Raize Software, which he founded in 1995. Ray is also the author of the highly acclaimed Developing Custom Delphi Components books and has published numerous articles on software development. Ray specializes in user interface design and custom component development. An engaging presenter, Ray is an Embarcadero MVP and a frequent speaker at developer conferences.

# About Embarcadero Technologies

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for application developers and database professionals so they can design systems right, build them faster and run them better, regardless of their platform or programming language. Ninety of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero products to increase productivity, reduce costs, simplify change management and compliance, and accelerate innovation. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices located around the world. Embarcadero is online at www.embarcadero.com.