# Mobile Tutorials

RAD Studio 10.2 Tokyo

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. The company's flagship tools include: Embarcadero® Change Manager™, CodeGear™ RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and Rapid SQL®. Founded in 1993, Embarcadero is headquartered in Austin, with offices located around the world. Embarcadero is online at www.embarcadero.com.

March, 2017

# CONTENTS

# Mobile Tutorials: Mobile Application Development (iOS and Android)

This integrated set of tutorials walks you through development of a Delphi or C++ multi-device application for iOS and Android:

- After the three initial setup tutorials, the first tutorial shows you how to construct an iOS or Android application using FireMonkey tools.
- The remaining tutorials demonstrate the recommended FireMonkey components to use in order to achieve a native look-and-feel in your iOS and Android applications.

These mobile tutorials are also available in PDF format here:
http://docs.embarcadero.com/products/rad_studio/radstudioTokyo/Mobile_Tutorials_en.pdf

# Setup

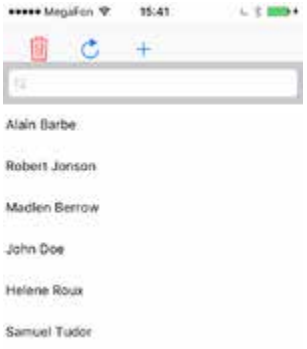

- Set Up Your Development Environment on the Mac (iOS)


- Set Up Your Development Environment on Windows PC (iOS)

- Set Up Your Development Environment on Windows PC (Android)

# Using Basic User Interface Elements


- Creating a Multi-Device Application (iOS and Android)




- Using the Address Book Component (iOS and Android)


- Using a Button Component with Different Styles (iOS and Android)


- Using a Calendar Component to Pick a Date (iOS and Android)


- Using Combo Box Components to Pick Items from a List (iOS and Android)

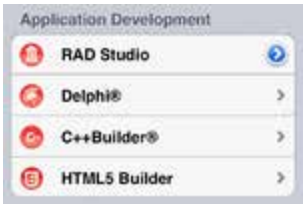

- Using a Map Component to Work with Maps (iOS and Android)

- Using the MultiView Component (iOS and Android)


- Using the Web Browser Component (iOS and Android)

- Using Tab Components to Display Pages (iOS and Android)


- Using LiveBindings to Populate a ListView (iOS and Android)




- Using ListBox Components to Display a Table View (iOS and Android)

- Using LiveBindings to Populate a ListBox (iOS and Android)

- Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android)

# Using Device Functionality


- Taking and Sharing a Picture, and Sharing Text (iOS and Android)


- Using Location Sensors (iOS and Android)

- Using Notifications (iOS and Android)


- Mobile Tutorial: Using the Phone Dialer on Mobile Devices (iOS and Android)

# Using Backend as a Service

- Mobile Tutorial: Using Remote Notifications (iOS and Android)


- Mobile Tutorial: Using BaaS for Backend Storage (iOS and Android)

# Accessing a Database


- Using InterBase ToGo with FireDAC (iOS and Android)
- Using InterBase ToGo with dbExpress (iOS and Android)


- Using SQLite and FireDAC (iOS and Android)
- Using SQLite and dbExpress (iOS and Android)


- Connecting to an Enterprise Database from a Mobile Client (iOS and Android)
- Using FireDAC in Mobile Applications (iOS and Android)

## See Also

- RAD Studio Mobile Tutorials (pdf)
- Mobile Tutorials: Table of Components Used
- FireMonkey Quick Start
- Creating an iOS App
- Creating an Android App
- FireMonkey Application Design
- Mobile Code Snippets
- iOS Mobile Application Development
- Android Mobile Application Development
- Supported Target Platforms
- Multi-Device Preview

# Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS)

A FireMonkey Delphi application destined for the iOS target platform can be optionally tested on the **iOS Simulator** available on the Mac. FireMonkey C++ and Delphi applications can be tested using the **iOS Device** target platform; this testing requires a test iOS device connected to the Mac.

- The first half of this tutorial describes the steps that you need to perform in order to run your iOS application (Delphi only) on the **iOS Simulator** on the Mac.
- The second half of this tutorial describes additional steps required in order to run your iOS application (Delphi or C++) on **your iOS Device**.

**Note:** The **iOS Simulator** is not supported by BCCIOSARM, the C++ Compiler for the iOS Device. Only iOS devices are supported by BCCIOSARM.

## Requirements on the Mac and iOS

- 10.9 Mavericks
- 10.10 Yosemite
- 10.11 El Capitan

(Neither OS is supported on legacy PowerPC- and 680x0-based Macintosh systems. All Macs since 2007 are Intel-based; all Macs since 2008 are 64-bit.)

- For OS X development, the latest version of Xcode
- For iOS development, the latest version of the iOS SDK and **Xcode** installed, along with **the Xcode command line tools**.

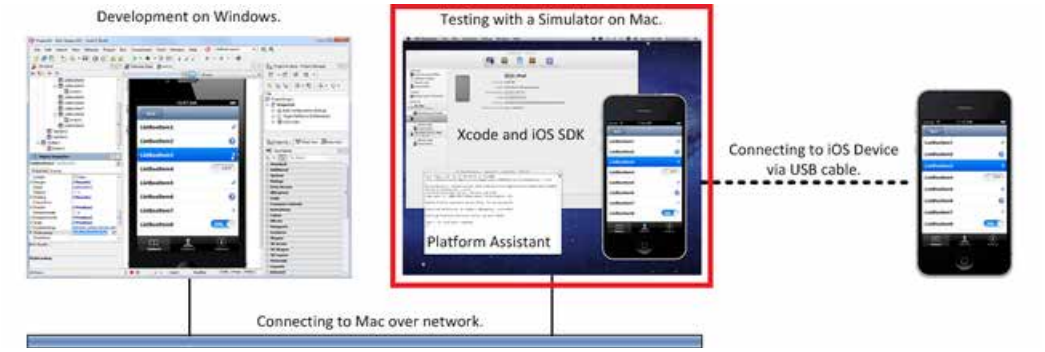
**Note:** RAD Studio does not support versions of the iOS SDK lower than 8.0.

- Membership in the Apple Developer Program.
- An iOS device connected to the Mac by USB port (required for testing or running your iOS app on the device)

## Steps to Configure Your Mac to Run Your iOS Application

To deploy an iOS application to your device for debugging and testing purposes, your system should have the configuration shown in the following figure. RAD Studio runs on a Windows PC computer. This PC computer is connected to a Mac running the **Platform Assistant** and having

installed the appropriate versions of **Xcode** and iOS SDK (for iOS development). To run iOS apps on an iOS device, the iOS device should be connected via USB cable to the Mac.




To deploy an iOS application to the **iOS Simulator** on the Mac or to an iOS device, you need to install the following tools on your Mac:

- **Platform Assistant (PAServer)**
  - RAD Studio uses the **Platform Assistant** to run and debug multi-device applications and to deploy multi-device applications on OS X and iOS devices.
  - Install and run the Platform Assistant on your Mac.
- **Xcode**

Xcode is the development and debug environment on the Mac and provides the required development files for OS X and iOS applications.

## Step 1: Install the Platform Assistant

The Platform Assistant must be running on the Mac when you deploy an iOS app from your PC to either the iOS simulator or an iOS device.

The OS X installer for the Platform Assistant is named `PAServer19.0.pkg` and it is available in two places:

- Inside the RAD Studio installation directory on your PC:

```
C:\Program Files (x86)\Embarcadero\Studio\19.0\PAServer\PAServer19.0.pkg
```

- On the Web, for download to the Mac:

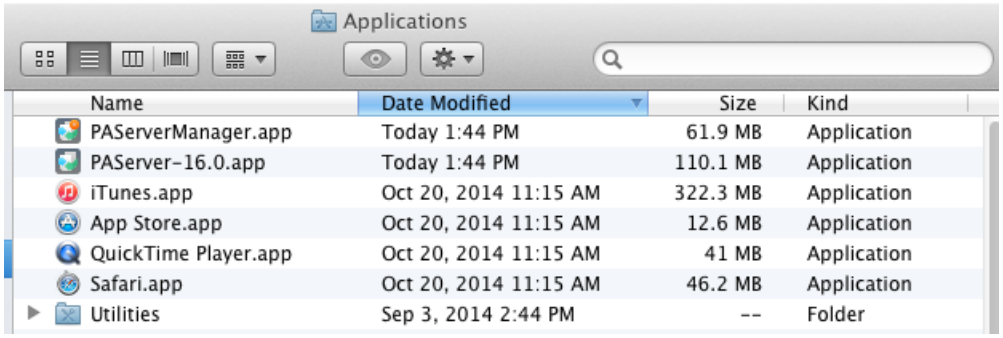
`http://altd.embarcadero.com/releases/studio/19.0/PAServer/PAServer19.0.pkg`


For further details, see [Installing the Platform Assistant on a Mac](#).

## Step 2: Run the Platform Assistant

In the Finder on the Mac, activate the .app file (`PAServer-19.0.app`) as follows:

1. Navigate to the top-level **Applications** folder.
2. Double-click `PAServer-19.0.app` to start the Platform Assistant:

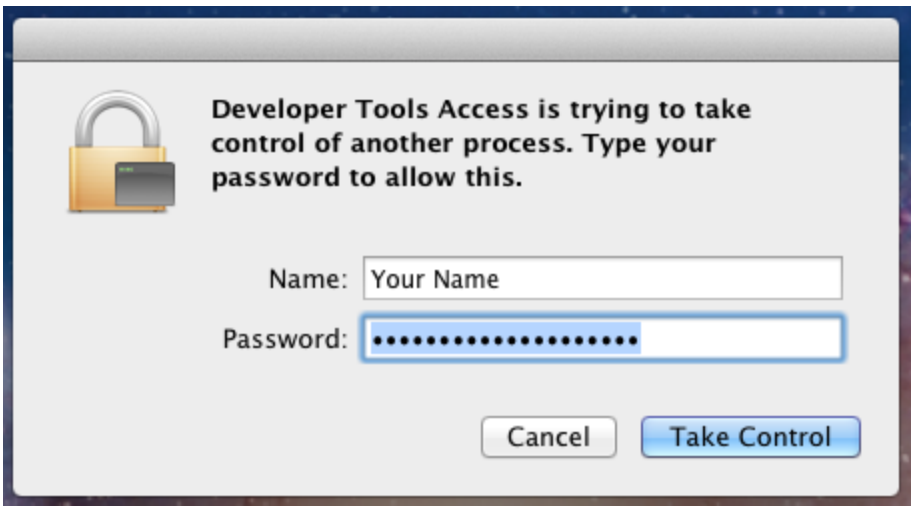



The Terminal window appears, displaying the Platform Assistant banner and the password prompt:

```
Connection Profile password <press Enter for no password>
```

Either press **Return**, or enter a password for PAServer and then press Return.

3. Next you are prompted to enter your Mac user password to allow the Platform Assistant to debug (take control of another process) your application.




Enter your password, and select **Take Control**:

```
Platform Assistant Server  Version 6.0.2.15
Copyright (c) 2009-2014 Embarcadero Technologies, Inc.

Connection Profile password <press Enter for no password>:

Acquiring permission to support debugging...succeeded

Starting Platform Assistant Server on port 64211

Type ? for available commands
>
```

For more details about running the Platform Assistant, see Running the Platform Assistant on a Mac.

## Step 3: Install Xcode on the Mac

**Xcode** is the development and debug environment on the Mac, and provides the required development files for OS X and iOS applications.

You can install Xcode from any of the following sources:

- On your "OS X Install" DVD, under **Optional Installs**, double-click **Xcode.mpkg** to install Xcode on your system.
- At the Mac App Store, download Xcode for free.
- As a registered Apple Developer, you can download the latest version of Xcode as a bundle (`.dmg`). To register and then download Xcode:
    1. Register (free of charge) as an Apple Developer at http://developer.apple.com/programs/register/.
    2. Download Xcode as a bundle from https://developer.apple.com/downloads.

### Next Steps

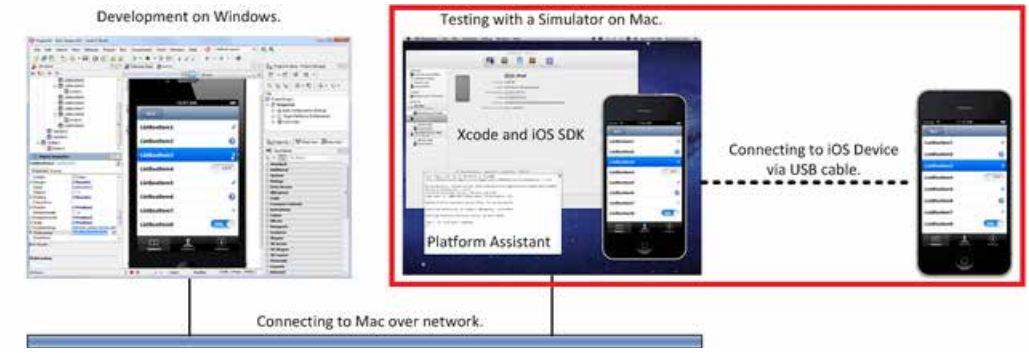
You have configured your Mac to run an iOS application on the **iOS Simulator**.

> **Note:** Only Delphi applications can be run on the **iOS Simulator**. C++Builder does not support the iOS Simulator.

- **iOS Simulator:** To run an iOS application (Delphi only) on the **iOS Simulator** on the Mac, you do not have to complete the second half of this tutorial. Instead, you can now go on to the next tutorial (Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS)) to complete the configuration of your RAD Studio IDE.
- **iOS Device:** To run your iOS application (either Delphi or C++Builder) on your **iOS Device**, please use the following steps in this tutorial to complete the configuration of your Mac. Then go on to next tutorial (Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS)) to complete the configuration of your RAD Studio IDE.

# Additional Steps to Configure Your Mac to Run Your iOS Application on Your iOS Device

The following additional steps enable you to run your iOS application on your **iOS Device**.



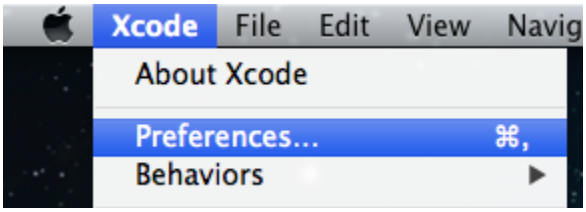

Remember that your iOS device should be connected to your Mac via USB cable.

## Step 1: Make Sure that the Xcode Command Line Tools Are Installed on Your Mac

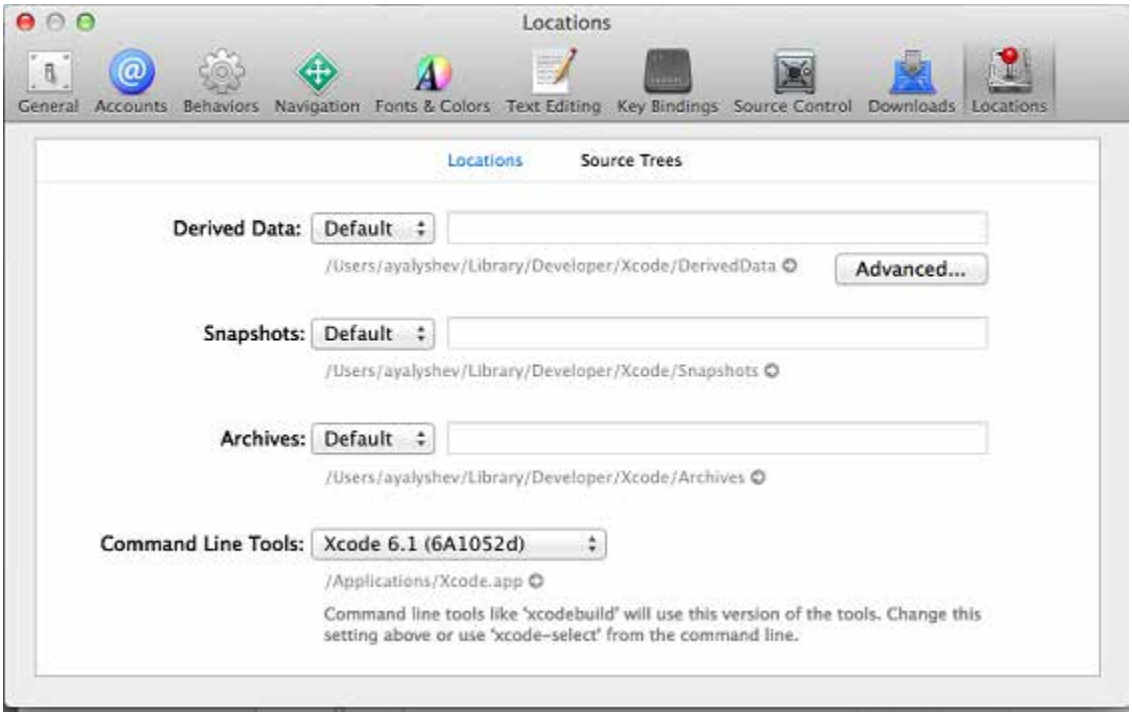
**Note:** Beginning with Xcode version 6.1, the Xcode Command Line Tools are automatically installed during Xcode installation.

**To make sure that the Xcode command line tools are installed on your Mac:**

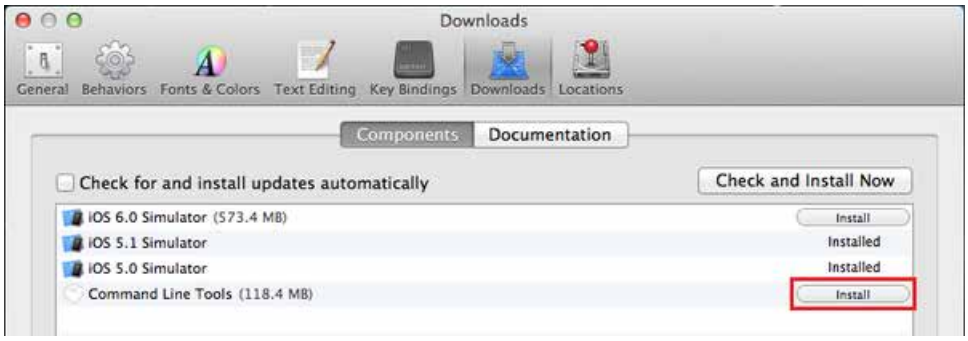
1. Start Xcode on the Mac.
2. Choose **Preferences** from the **Xcode** menu.




3. In the **General** window, click the **Locations** tab.
4. On the **Location** window, check that the **Command Line Tools** shows the Xcode version (with which the Command Line Tools were installed). This means that the Xcode Command Line Tools are already installed and you do not need to install them.




5. If the Xcode Command Line Tools are not installed, the **Command Line Tools** do not show the Xcode version.
    1. In this case, click the **Downloads** tab on the **Location** window.
    2. On the **Downloads** window, choose the **Components** tab.




3. Click the **Install** button next to **Command Line Tools**.

You are asked for your Apple Developer login during the install process.

For more details, see Installing the Xcode Command Line Tools on a Mac.

# Step 2: Sign Up for a Developer Account

Membership in the Apple Developer Program is a requirement for building, running, debugging, and deploying applications for iOS.

Follow the steps described at Joining the Apple Developer Program.

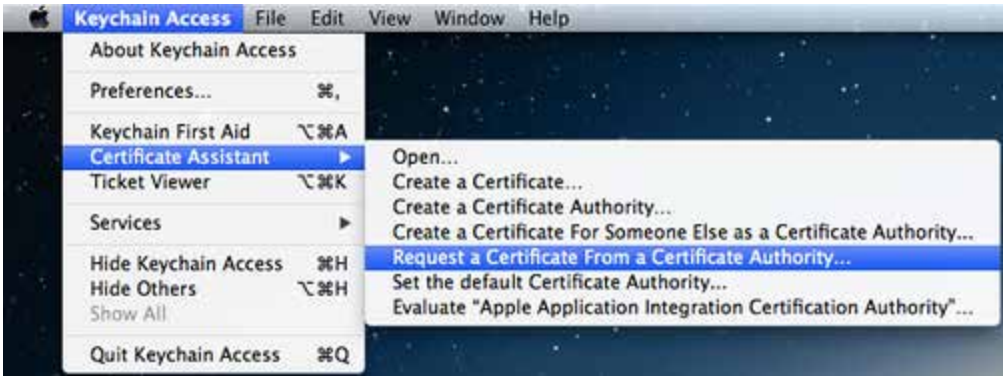
# Step 3: Request, Download and Install Your Development Certificate

Applications that are deployed on the device (or on the iOS Simulator) need to be cryptographically signed before they run. The **Development certificate** contains information that is needed for signing the applications. Each individual (an individual developer or a team member) must have a unique development certificate, which can be used for multiple applications.

For development teams, development certificates must be requested by each team member, and these requests must be approved by a team admin.

## Request, Download and Install Your Certificate

1. In the **Keychain Access** application on your Mac, select from the **Keychain Access** menu: **Certificate Assistant > Request a Certificate From a Certificate Authority**:




- If you are a development team member for a corporate/organization program, your team administrator needs to approve your request. After your team administrator approves it, you can download the certificate.
- If you are an individual developer, you should see a download option for your certificate shortly after you request it. See Apple documentation at: Code Signing Guide for details.

Save the certificate request as a CSR file, and then send it to your Certificate Authority through the iOS provisioning portal in the following way:

1. When prompted, enter your Apple ID and password, and then click **Sign In**.
2. Under **iOS Apps**, click **Certificates**.
3. On the page that opens, click the plus sign (+) icon.
   This opens the **Add iOS Certificate** wizard.
4. On the **Select Type** page, click **Continue** and follow the onscreen instructions to proceed with the wizard.
5. When prompted, upload the CSR file that you saved on your Mac.

2. Go to iOS Provisioning Portal. You can download the Development certificate clicking the **Download** button as shown below:


3. Launch the Development Certificate by double-clicking it. It automatically loads in the **Keychain Access** application.

# Step 4: Register Your Device for Deployment

Before a device can run user applications, it must be registered in the Apple Provisioning Portal. Devices are registered by their Unique Device ID (UDID). The UDID can be determined using Xcode, as follows:

1. Make sure your iOS device is connected to your Mac machine.
2. Open Xcode and go to **Devices** (**Window > Devices**).
3. Click on your device.
4. Next to the **Identifier** label is a string of characters:


The Identifier string represents your device's UDID.

- If you are an individual developer, register your device by adding the UDID in the Devices tab of the Apple Provisioning Portal.
- If you are part of a company/organization, ask your team admin to register your device.

# Step 5: Create and Install a Provisioning Profile

Provisioning profiles are used for linking a developer and devices to a development team. This provisioning profile is required for running applications on an iOS device.

- If you are an individual developer, you must create a provisioning profile. For specific information, see: Creating and Downloading a Distribution Provisioning Profile.
- If you are part of a company/organization, your team admins must create a provisioning profile that you can use.

After your provisioning profile is created, you must install it into Xcode, as follows:

1. Open Xcode on the Mac and go to the Devices (**Window > Devices**).
2. Right-click the device and select **Show Provisioning Profiles…**

3. Click the **plus sing button** to add a provisioning profile.


4. Find your provisioning profile, select it and click **Install**.
5. Click **Done** to close the window.

You have configured your Mac to run your iOS application on your **iOS Device**.

To run your iOS application, please see Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS) and complete the configuration of your RAD Studio IDE. (If you have configured your PC as part of running your application on the **iOS Simulator**, you can skip this step.)

# See Also

- Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS)
- Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android)
- paserver, the Platform Assistant Server Application
- Installing Xcode on a Mac
- Installing the Xcode Command Line Tools on a Mac
- Installing the Platform Assistant on a Mac
- Running the Platform Assistant on a Mac
- Acquiring an iOS Developer Certificate
- Troubleshooting: Cannot Deploy to the iOS Device
- Useful Apple Web Pages:
  - Creating and Configuring App IDs
  - Creating signing certificates (Code Signing Guide)
  - iOS Provisioning Portal (Requires your Apple Developer login)
  - Devices tab of the Apple Provisioning Portal
  - Create an Apple ID
  - Creating and Downloading a Distribution Provisioning Profile

# Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS)

Before starting this tutorial, you should read and perform the following tutorial session:

- Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS)

A FireMonkey application destined for the iOS target platform can be tested initially on the **iOS Simulator** available on the Mac. The second half of the testing process uses the **iOS Device** target platform and requires a test iOS device connected to the Mac.

**Note:** On iOS devices, you can run both Delphi and C++ applications. However, the **iOS Simulator** is not supported by BCCIOSARM, so only iOS devices are supported for C++.

To deploy an iOS Application to your iOS device or iOS Simulator for debugging and testing purposes, RAD Studio uses the Platform Assistant, which you must install and run on the Mac. Your hardware and software development environment should have the configuration demonstrated in the following figure. RAD Studio runs on a Windows PC computer. Your PC computer should be connected to a Mac running the Platform Assistant and having installed the appropriate versions of Xcode and iOS SDK (for iOS development). To run iOS apps on an iOS device, the iOS device must be connected via USB cable to the Mac.

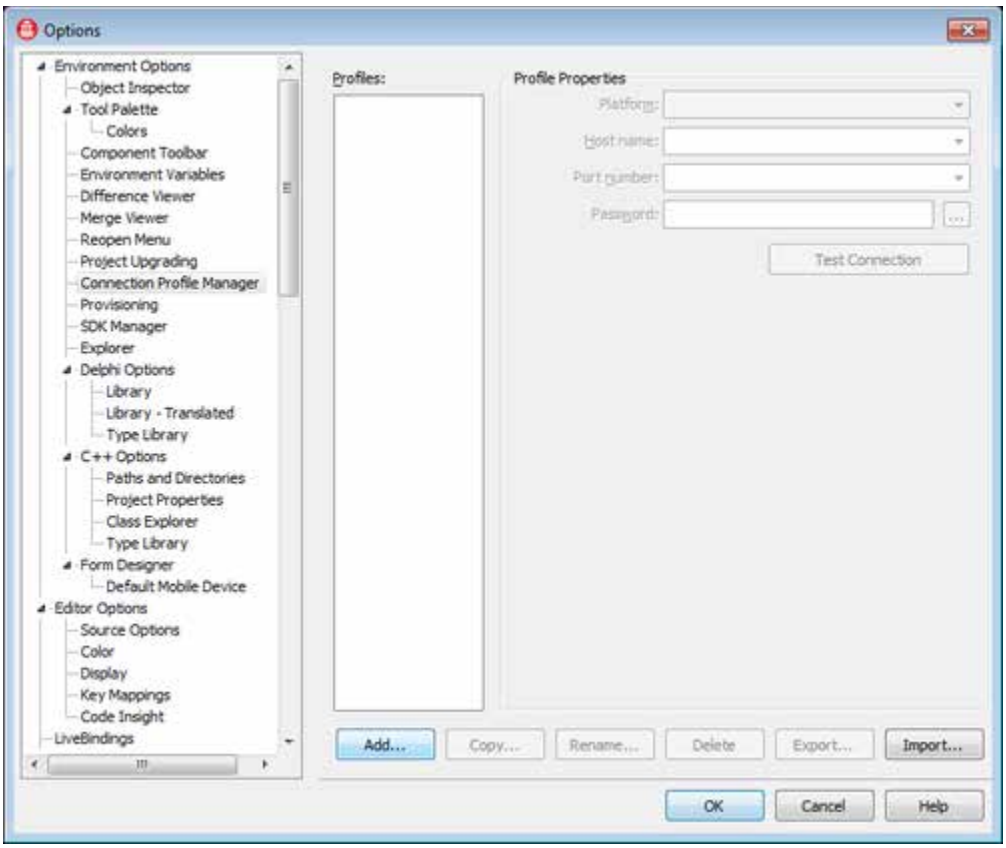

This section describes the steps to set up your development environment **after** you configure your environment on your Mac.

## Setting Up Your RAD Studio Environment

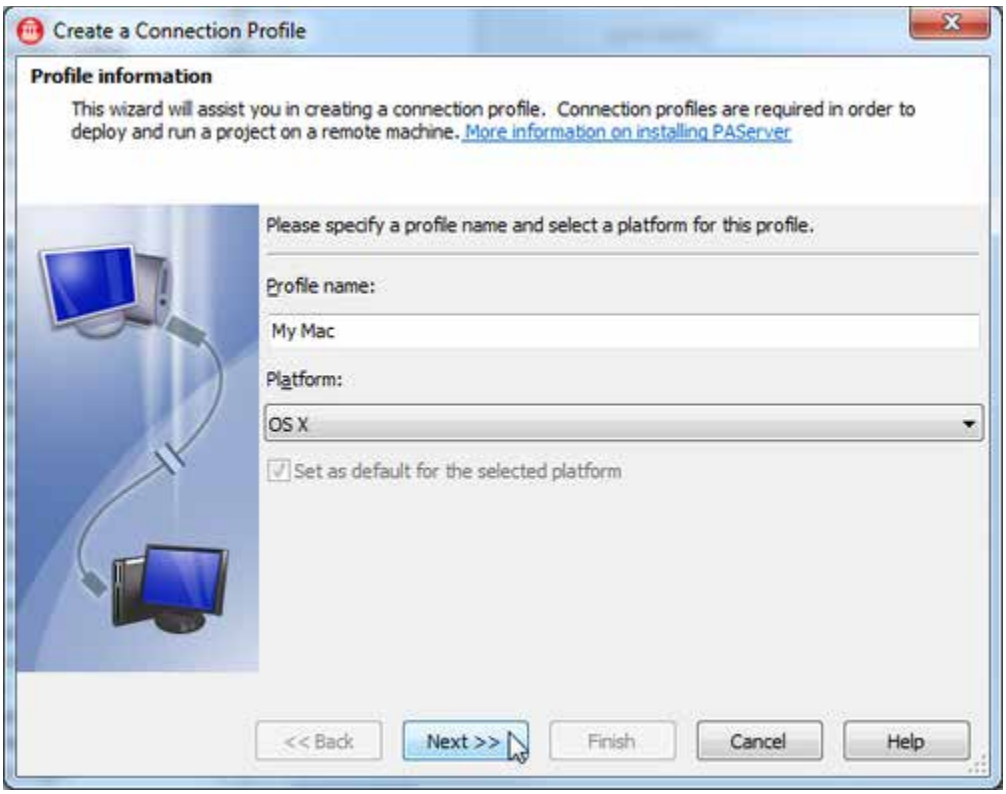
You need to execute the following configuration steps to prepare the iOS development with RAD Studio.

### Create a Connection Profile for the Mac

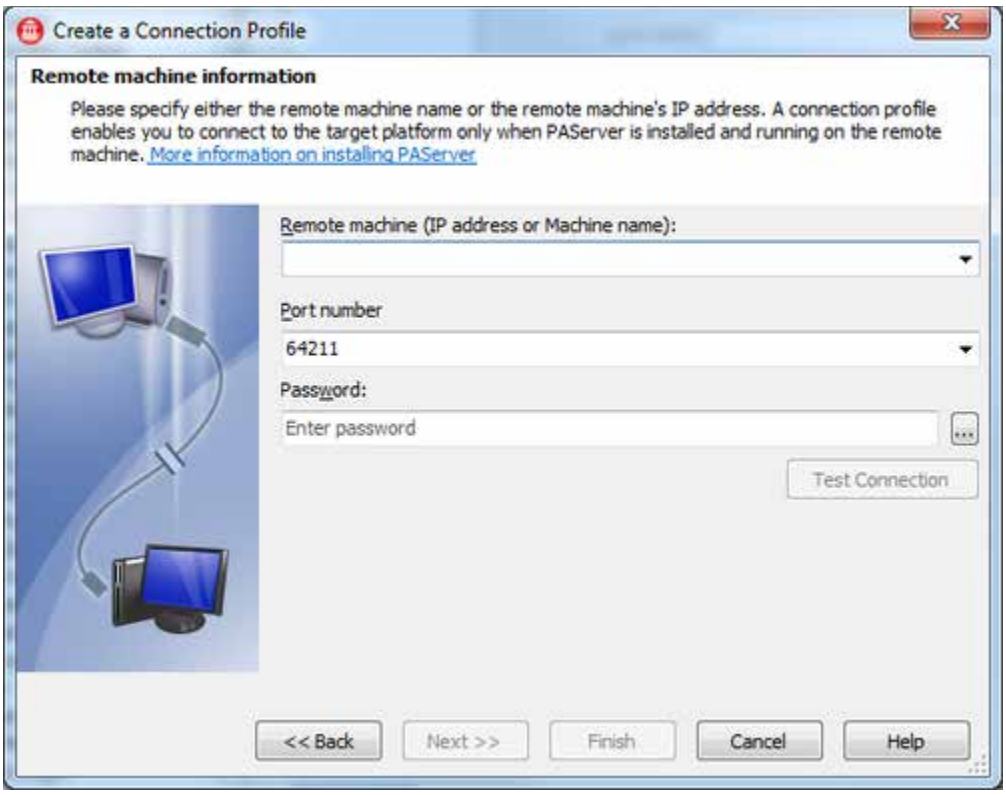
1. In the RAD Studio IDE, open **Tools > Options > Environment Options > Connection Profile Manager**.
2. Click **Add**:




3. Now you see the Create a Connection Profile wizard. Define a name for the connection profile, such as "My Mac".
   Make sure you select **OS X** as the platform, and then click **Next**:



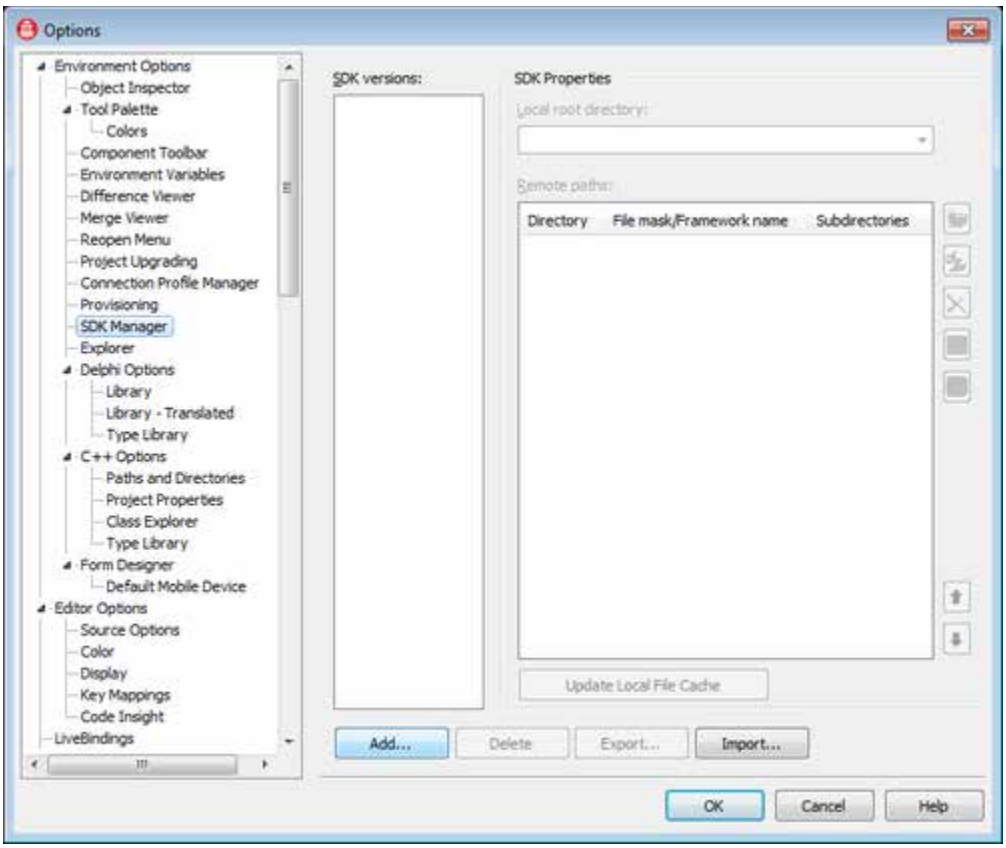

4. On the **Remote machine information** page, set the name or IP address of the host Mac, a port number to use (the default port 64211 typically works), and an optional password (if you want to use a password).



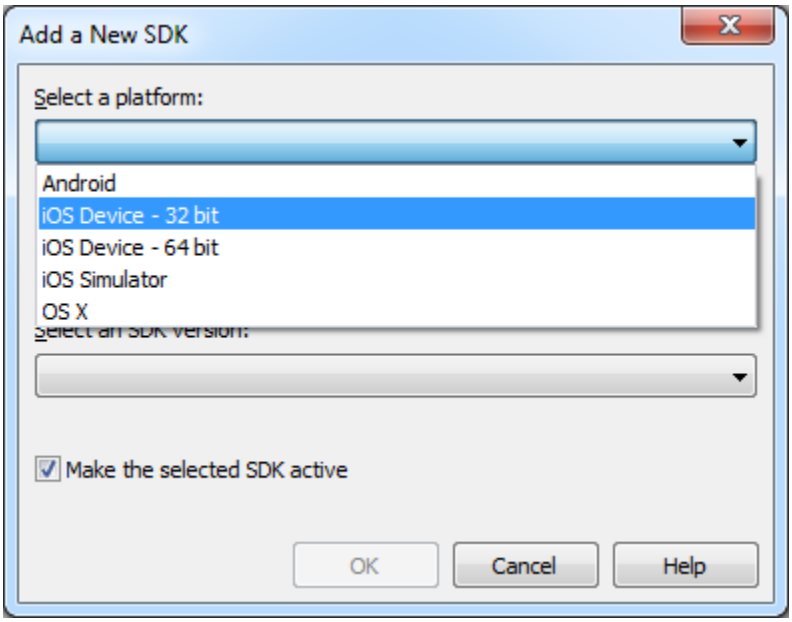

5. Click **Test Connection**, and make sure that the connection profile succeeds with no error (you should receive the message **"Connection to <hostname> on port <portnumber> succeeded"**).
6. If the **Test Connection** succeeds, click the **Finish** button, otherwise check the settings and try again.

## Add an SDK to the Development System for the iOS Device Connected to the Mac

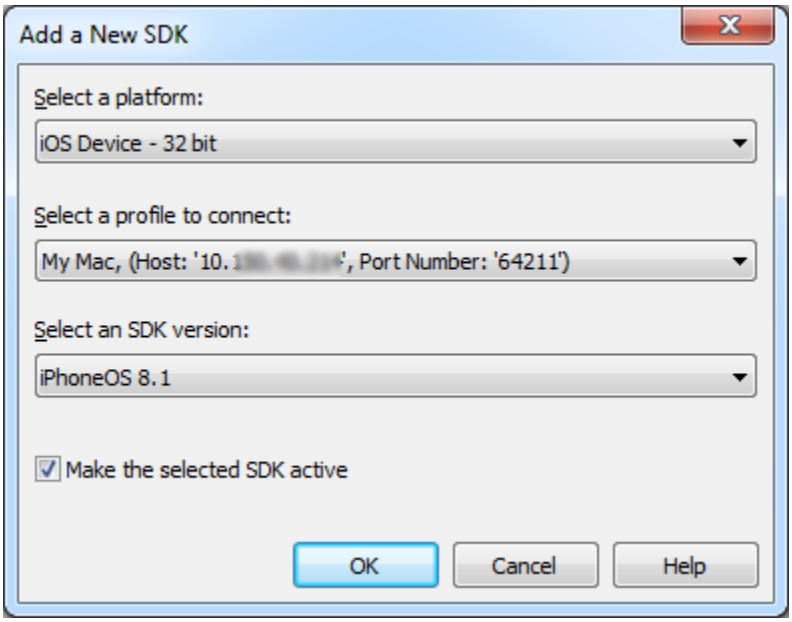
1. Open **Tools > Options > Environment Options > SDK Manager**:




2. Click **Add**.
3. On the **Add a New SDK** dialog box, select **iOS Device - 32 bit** or **iOS Device - 64 bit** as a platform.



4. After you select a platform, the IDE fills a Profile (such as "My Mac") and SDK version combo box with the list of SDK versions available on the machine where the Platform Assistant server is running:




5. Click **OK** to add the selected SDK version.
6. After the operation completes, click **Close** to close the dialog.

# See Also

- Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android)
- Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS)
- Working with a Mac and a PC
- Running Your iOS Application on an iOS Device
- Running Your iOS Application on the iOS Simulator
- FireMonkey Platform Prerequisites
- Creating an iOS App
- OS X Application Development
- Creating a FireMonkey Application
- Apple developer.apple.com pages
  - iOS Developer Library
  - iOS Developer Library: Getting Started
  - iOS Dev Center

- Launching Your App on Devices
- Preparing Your iOS App for Distribution in the App Store
- iAd Network

# Mobile Tutorial: Set Up Your Development Environment on Windows PC (Android)

During the RAD Studio installation, the required Android development tools are installed and configured on your system. Nothing else is required to start Android development.

In order to be able to run applications on an Android device, you also need to:

- Install the USB driver for your Android device
- Enable USB debugging on your Android device
- Configure your system to detect your Android device

## See Also

- Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android)

# Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android)

This topic describes how to create a "Hello World" multi-device application (C++ or Delphi) for either the iOS or Android target platform.

## Before You Start

To develop mobile (iOS and Android) applications using RAD Studio, you need to complete some important configuration steps. This tutorial assumes that you have completed all the necessary setup steps.

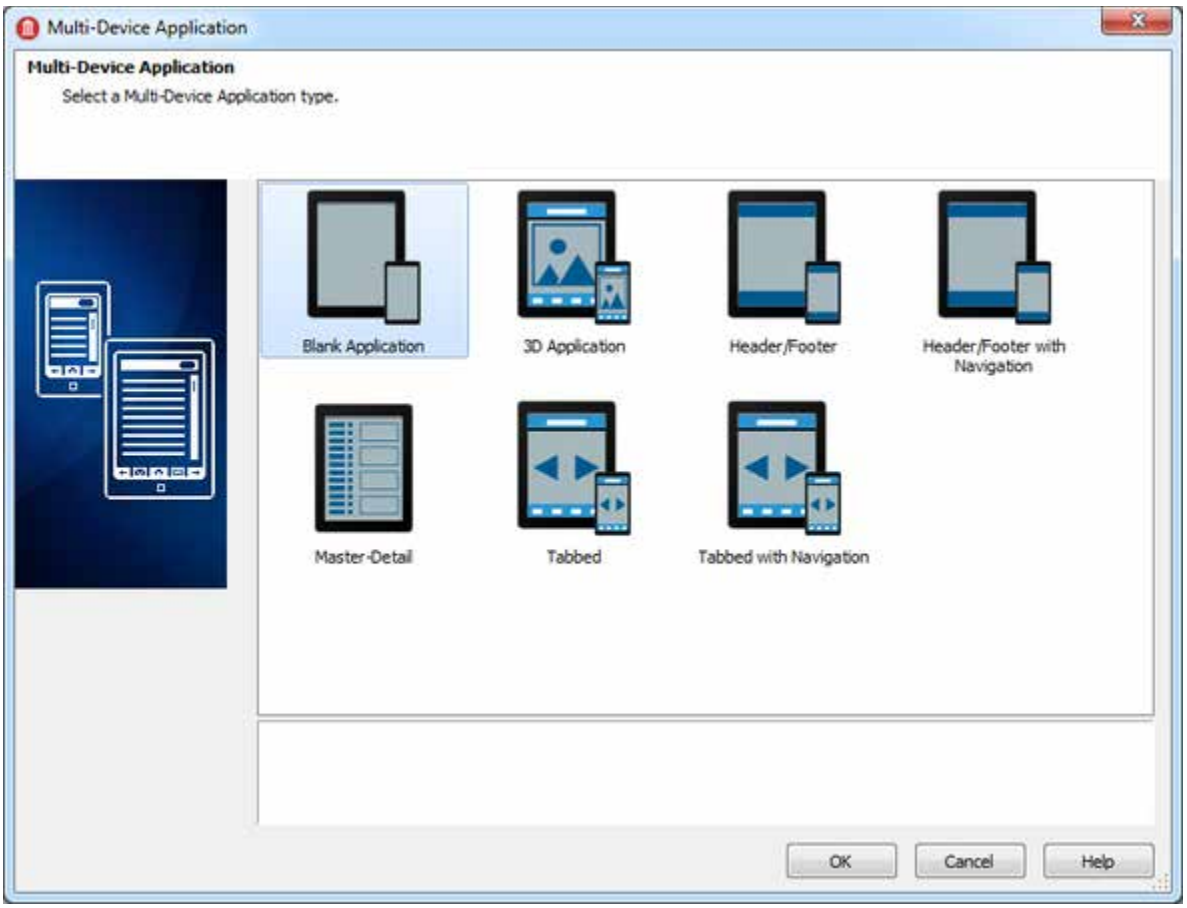
For details, see:

- Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS)
- Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS)
- Mobile Tutorial: Set Up Your Development Environment on Windows PC (Android)

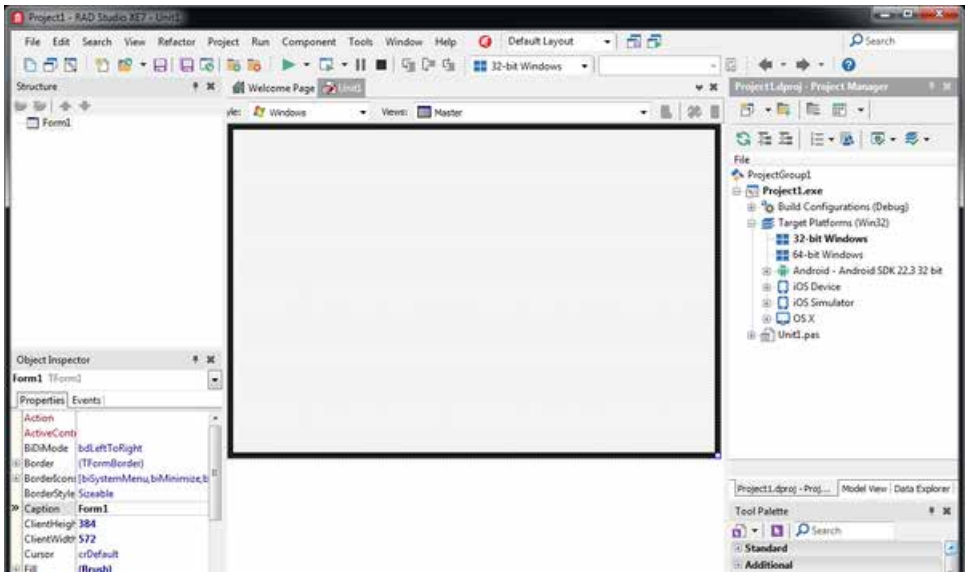
## Step 1: Create a New FireMonkey Application for Android or iOS

1. Select either:
    - File > New > Multi-Device Application - Delphi
    - File > New > Multi-Device Application - C++Builder

The Multi-Device Application wizard appears:




2. Select **Blank Application**. The Form Designer shows a new form:



3. Select the target platform from the Project Manager.
    1. **Android:** See **Configuring Your System to Detect Your Android Device** to use an Android device.

2. **iOS:** If you want to create an **iOS** app, open the **Target Platform** node in the Project Manager and double-click **iOS Simulator** (only for Delphi) or a connected iOS device (for either Delphi or C++):

**Note:** When you select a platform, the components not available for this particular platform appear grayed.

# Step 2: Select a Style

1. Select either iOS or Android from the **Style** drop-down menu in order to define the Master view to show all the properties related with this style.

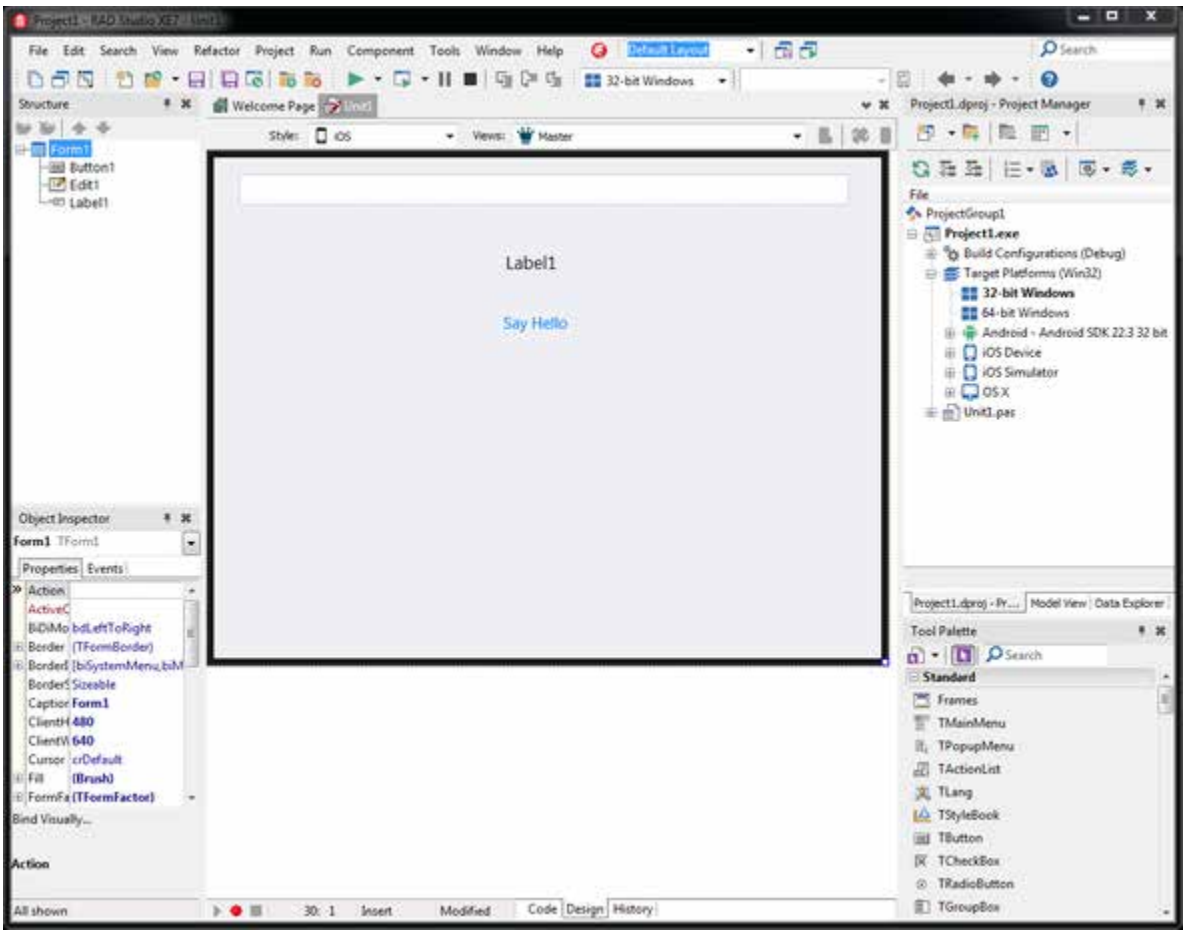
**Note:** See Style Selector for more information.

# Step 3: Place Components on the Multi-Device Form

We recommend that you read this tutorial before you start placing components: Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android).

The first step in creating a multi-device application is designing the user interface. There are many reusable components available in the IDE for creating user interfaces.

1. Move the mouse pointer over the Tool Palette, and expand the **Standard** category by clicking the plus (+) icon next to the category name.
2. Select the **TEdit** component and either double-click TEdit or drop it onto the Form Designer.
3. Repeat these steps, but now add a **TLabel** and a **TButton** component to the form.
4. Select the edit box and set the **KillFocusByReturn** property in the Object Inspector to `True`.
5. Select the button and change the **Text** property in the Object Inspector to "Say Hello".
6. Now you should see three components on the Form Designer. Here is an iOS app:

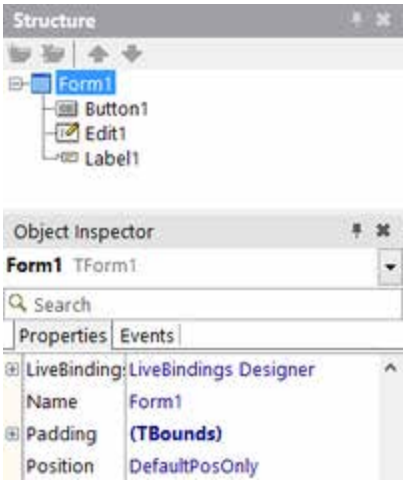



7. After you place these components on the Form Designer, the IDE automatically sets names for the components.

To see or to change the name of a component, click the component on the Form Designer, and then find its **Name** property in the Object Inspector and the Structure View:

For a **TButton** component, the component name is set by default to **Button1** (or Button2, Button3, depending on how many TButtons you have created in this application).

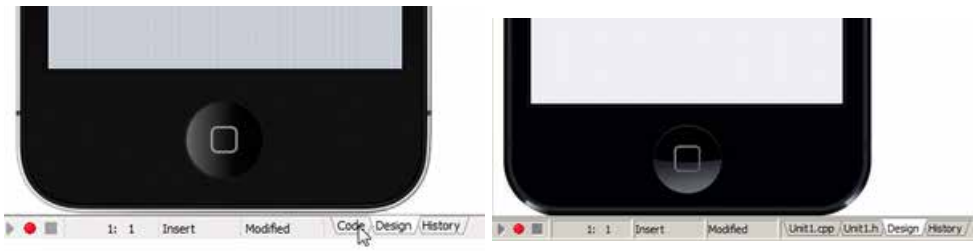
8. The form on which these components are located also has a name. Select the background of the Form Designer, and select the **Name** property in the Object Inspector. The name of the form **Form1** (or Form2, Form3,...) is displayed. You can also locate the name of the form in the Structure View:




**Note:** Form names set by default as Form1, Form2, Form3,... are for the Master views. Device views are named **FormName_ViewName** such as **Form1_iPhone** (iPhone 3.5" form) and **Form1_NmXhdpiPh** (Android 4" Phone form).

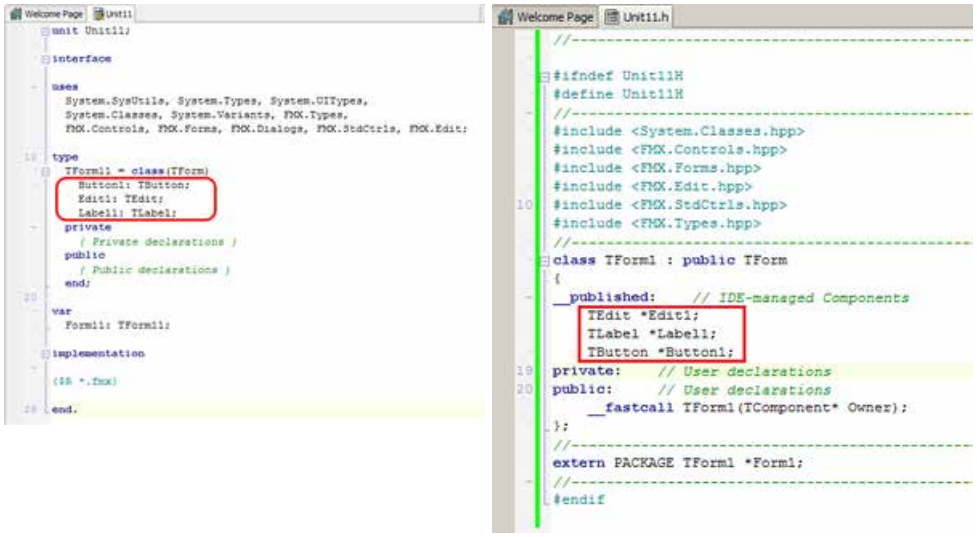
9. You can easily switch to source code by selecting the **Code** (for Delphi) or **<unit name>.cpp/<unit name>.h** (for C++) tab at the bottom of the Form Designer. You can also press the **F12** key to switch between the Form Designer and the Code Editor:

**Delphi** **C++**




The Code Editor displays the source code that the IDE has generated. You should find three components defined (Edit1, Label1, and Button1):

**Delphi** **C++**




**Note:** When you save or run your project, the **uses** and **include** clauses are updated (to add FMX.StdCtrls for TLabel and FMX.Edit for TEdit).

# Step 4: Adding Views to Your Project

If you want to customize you application for a particular type of device, you can do it using Views.

1. Go to the **Views** selector.
2. Select the available views you want to add just by clicking on them.
3. Go to the view to do the changes you want to include.

To add a customized view, see Adding a Customized View to the View Selector.

# Step 5: Write an Event Handler for a Button Click by the User

The next step is defining an event handler for the TButton component. You can define event handlers for your application in the same way you define event handlers for desktop platforms. For the TButton component, the most typical event is a button click.

Double-click the button on the Form Designer, and RAD Studio creates skeleton code that you can use to implement an event handler for the button click event:

**Delphi** **C++**


Now you can implement responses within the **Button1Click** method.

The following code snippets (Delphi and C++) implement a response that displays a small dialog box, which reads "Hello + <name entered into the edit box>":

Delphi code:

```
Label1.Text := 'Hello ' + Edit1.Text + ' !';
```

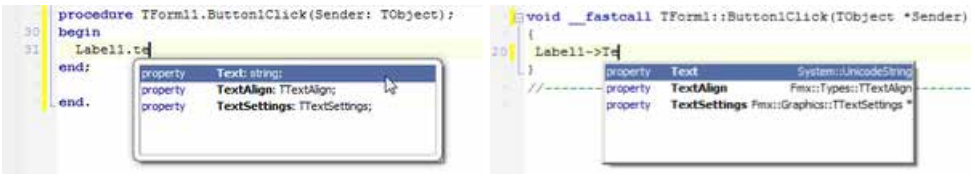
C++ code:

```
Label1->Text = "Hello " + Edit1->Text + " !";
```

In Delphi, the quotation marks that surround string literals must be straight single quotation marks (that is, `'string'`). You can use the plus (+) sign to concatenate strings. If you need a single quote inside a string, you can use two consecutive single quotes inside a string, which yields a single quote.

While you are typing code, some tooltip hints appear, indicating the kind of parameter you need to specify. The tooltip hints also display the kinds of members that are supported in a given class:

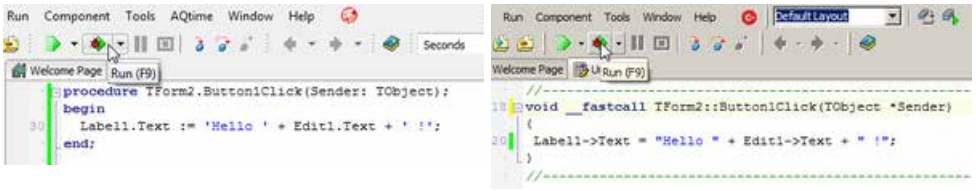
**Delphi** **C++**




# Step 6: Test Your Mobile Application

The implementation of this application is finished, so now you can run the application.
You can click the **Run** button ( ) in the IDE, press **F9**, or select Run > Run from the RAD Studio main menu:

**Delphi** **C++**



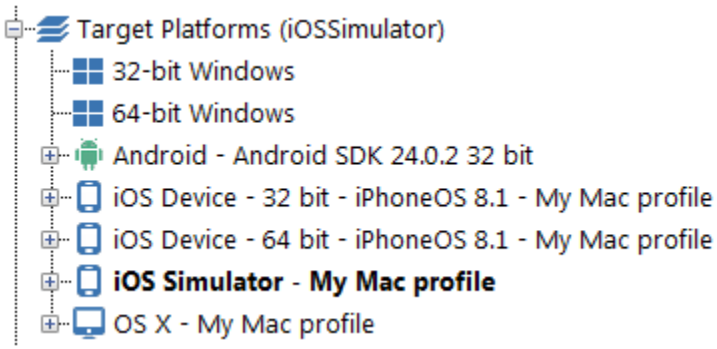

## Test Your Android Application on the Android Device

If you complete the steps described in [Mobile Tutorial: Set Up Your Development Environment on Windows PC (Android)] before creating your new project, you can now run your Android app on an Android device connected to your PC by USB cable.

## Test Your iOS Application

### Testing on the Mac (iOS Simulator)

By default, FireMonkey Delphi iOS applications run on the **iOS Simulator** target platform. You can confirm the target platform in the Project Manager:



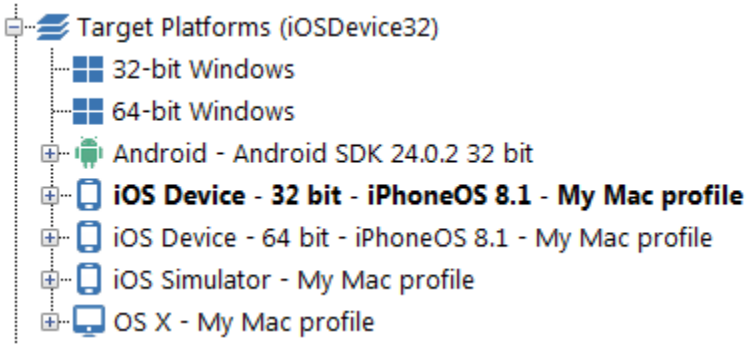

When you run your application, it is deployed to the Mac and then to the iOS Simulator on the Mac. For our app, a form with an edit box and a button is displayed. Enter text into the edit box, and click the **Say Hello** button:


**Note:** On the iOS simulators, you can test only your Delphi applications.

### Testing on a Connected iOS Device

If you complete the steps described in Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS) and Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS) before creating your new project, you can now run your iOS app on an iOS device connected to your Mac by USB cable.

To run your iOS app on a connected iOS device, first select the **iOS Device - 32 bit** or **iOS Device - 64 bit** target platform so that the Platform Assistant deploys the application to the connected iOS Device:




After you select the appropriate **iOS Device** target platform, run your iOS app by clicking the **Run** button in the IDE, pressing F9 or selecting Run > Run.

On your Mac, you might see a dialog asking your permission to code sign your iOS app. Select either "Always Allow" or "Allow" to sign your app.


Then go to your iOS device and wait for your FireMonkey iOS app to appear. Watch for the FireMonkey launch image (the icon is available in $(BDS)\bin\Artwork\iOS, and you can set the launch image in Application Options):

## See Also

- Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)

- Android Mobile Application Development
- iOS Mobile Application Development
- OS X Application Development
- Mobile Code Snippets

# Mobile Tutorial: Using an Address Book Component (iOS and Android)

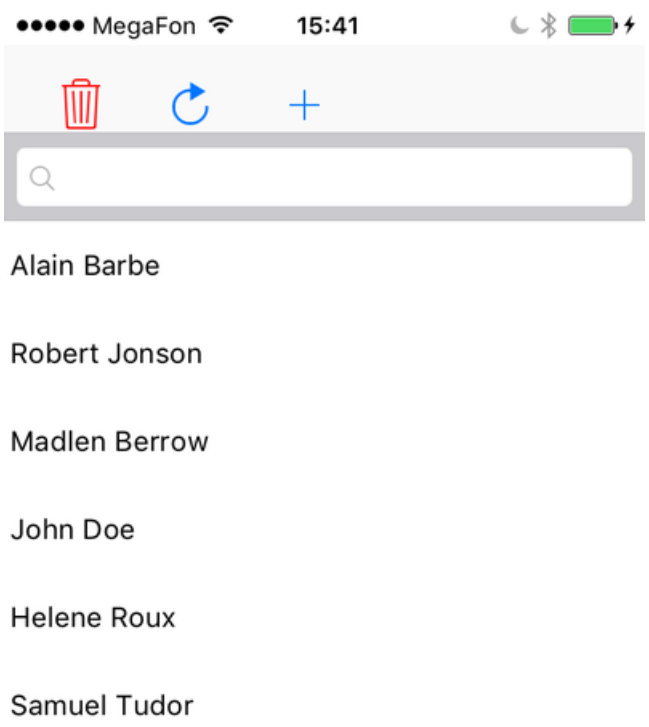
Before starting this tutorial, it is recommended that you read and perform the following tutorials:

- Mobile Tutorial: Using a MultiView Component to Display Alternate Views of Information (iOS and Android)
- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

FireMonkey provides the TAddressBook component to work with Address Book on iOS and Android devices.

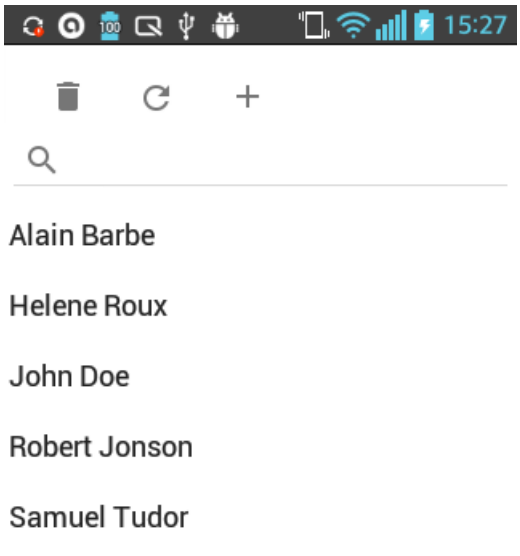
This tutorial describes how to create a simple FireMonkey application that uses the TAddressBook component.

**iOS**




**iPhone 5**

**Android**




**Android (LG - E612)**

# Basic Features of the TAddressBook Component

On mobile platforms (Android and iOS), FireMonkey supports the TAddressBook component that lets your applications access a device Address Book. The basic features of this component are as follows:

- Sending a request to access a device Address Book and getting the access status.
- Accessing all sources defined in the device Address Book.
- Fetching all or individual contacts from the specified source.
- Fetching all contacts from the specified group(s).
- Editing or removing existing contacts and groups.
- Creation of new contacts.
- Creation of new groups.

# Creating a Sample Application

This section helps you develop a sample application (for Android and iOS target platforms) that illustrates the use of the TAddressBook component. The application demonstrates the following techniques:

- Requesting a permission to access Address Book.
- Retrieving all contacts from the default source in the device Address Book.
- Adding a new contact to Address Book.
- Removing a selected contact from Address Book.

## Designing the User Interface

1. Create a blank Multi-Device Application, by selecting:
   - For **Delphi**: **File > New > Multi-Device Application - Delphi > Blank Application**
   - For **C++**: **File > New > Multi-Device Application - C++Builder > Blank Application**
2. In the Project Manager, set the target platform to Android or iOS.
3. In the Tool Palette, select the **TToolBar**, **TAddressBook**, and **TListBox** components and drop them on the Form Designer.
4. In the Object Inspector, set the **Align** property of **TListBox** to `Client`.
5. On the Form Designer, right-click **TListBox**, and then on the shortcut menu, select **Add Item > TSearchBox**.

6. In the Tool Palette, select the **TMultiView** component, and drop it on the Form Designer.

## Designing the Application Toolbar and Master Pane

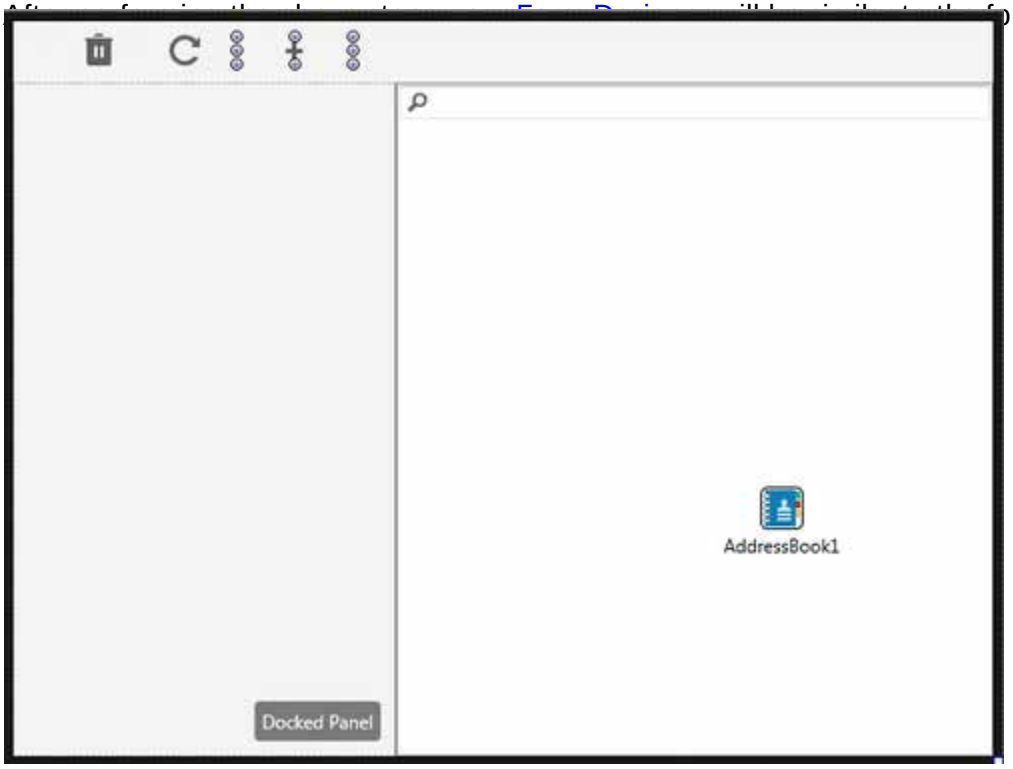
Place all control elements on the toolbar and Master Pane:

- The toolbar contains three speed buttons:
  -  : removes a selected contact from the Address Book.
  -  : refreshes the current list of contacts.
  -  : opens the Master Pane to add a new contact to the Address Book.
- The Master Pane provides a form to specify and add a new contact to the Address Book.

**To design the application toolbar:**

1. On to the top toolbar, drop three TSpeedButton components.
2. In the Form Designer, do the following:
   - Select **SpeedButton1**, and set the **Name** and **StyleLookup** properties to `btnRemove` and `trashtoolbuttonordered`, respectively.
   - Select **SpeedButton2**, and set the **Name** and **StyleLookup** properties to `btnRefresh` and `refreshtoolbuttonordered`, respectively.
   - Select **SpeedButton3**, and set the **Name** and **StyleLookup** properties to `btnAdd` and `addtoolbuttonordered`, respectively.

After performing these steps, Form Designer will look similar to the following screen:

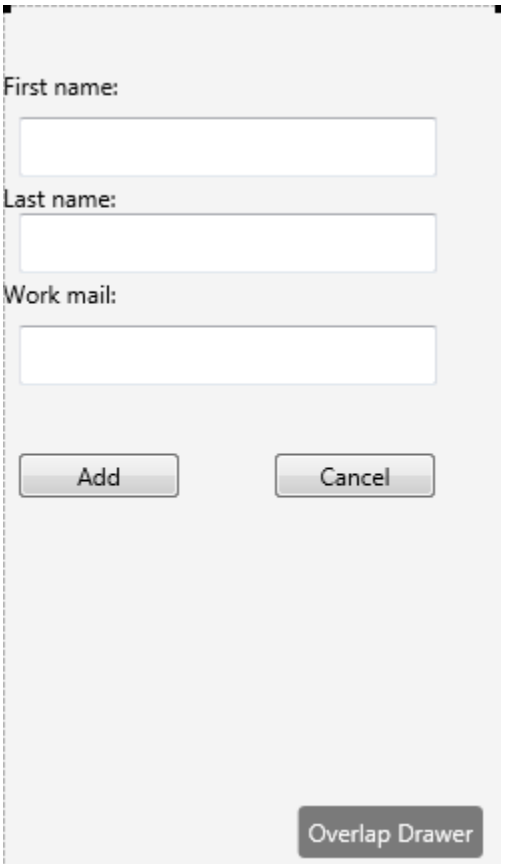



To design the Master Pane:

1. In the Structure View, select **MultiView1**, and then in the Object Inspector, specify the following properties:
   - Set the **MasterButton** property to `btnAdd`.
   - Set the **Mode** property to `Drawer`.
   - To make the **MultiView1** visible, set the **Visible** property to `True`.
2. In the Tool Palette, select three **TLabel** components, three **TEdit** components, and two **TButton** components, and drop them to **Multiview1**.
3. In the Object Inspector, specify the following properties of the control elements that you have placed onto **MultiView1**:
   - For **Label1**, **Label2**, and **Label3**: set the **Text** properties to `First Name`, `Last Name`, and `Work Mail`, respectively.
   - For **Edit1**, **Edit2**, and **Edit3**:
     - set the **Name** property to `edtFirstName`, `edtLastName`, and `edtWorkMail`, respectively.
     - set the KillFocusByReturn property to `true`
     - set the ReturnKeyType property to `Go`.

- For **Button1** and **Button2**:
  - set the **Text** property to `Add` and `Cancel`
  - set the **Name** property to `btnOK` and `btnCancel`, respectively.

We recommend that you arrange the control elements of the Master Pane in the following way:




## Retrieving the Contacts List

This section explains how to retrieve the Contacts list from the default source in a device Address Book.

In your application, define the following private methods:

- `FillContactsList`: gets the list of contacts of the default source in the device Address Book;
- `AddListBoxItem`: adds a new item to **ListBox**.

Make the following changes to the Private section of your application code:

**Delphi**:

```
private
    { Private declarations }
```

```
    procedure FillContactList;
    procedure AddListBoxItem(Contact: TAddressBookContact);
```

**C++Builder:**

```
private:  // User declarations
        void __fastcall FillContactsList();
        void __fastcall AddListBoxItem(TAddressBookContact *Contact);
```

Implement the `FillContactslist` and `AddListBoxItem` methods as follows:

**Delphi**:

```
// Fill the contact list
procedure TForm1.FillContactList;
var
  I: Integer;
  Contacts: TAddressBookContacts;
begin
  Contacts := TAddressBookContacts.Create;
  try
    AddressBook1.AllContacts(AddressBook1.DefaultSource, Contacts);
    ListBox1.BeginUpdate;
    try
      ListBox1.Clear;
      for I := 0 to Contacts.Count - 1 do
        AddListBoxItem(Contacts.Items[I]);
    finally
      ListBox1.EndUpdate;
    end;
  finally
    Contacts.Free;
  end;
end;
// ---------------------------------------------------------------------
// Add a new contact to List box
procedure TForm1.AddListBoxItem(Contact: TAddressBookContact);
var
  ListBoxItem: TListBoxItem;
begin
  try
    ListBoxItem := TListBoxItem.Create(nil);
    ListBoxItem.Text := Contact.DisplayName;
    ListBoxItem.Tag := Contact.ID;
    ListBox1.AddObject(ListBoxItem);
  finally
    ListBoxItem.Free;
  end;
end;
```

**C++Builder:**

```
// Fill the List box with existed contact display names.
void __fastcall TForm1::FillContactsList() {
```

```
        int i;
        TAddressBookContacts *Contacts = new TAddressBookContacts();
        __try {
                AddressBook1->AllContacts(AddressBook1->DefaultSource(), Contacts);
                __try {
                        ListBox1->BeginUpdate();
                        ListBox1->Clear();
                        for (i = 0; i < Contacts->Count; i++)
                                AddListBoxItem(Contacts->Items[i]);

                }
                __finally {
                        ListBox1->EndUpdate();
                }
        }

        __finally {
                Contacts->Free();
        }
}
// ---------------------------------------------------------------------------
// Add a new contact to List box
void __fastcall TForm1::AddListBoxItem(TAddressBookContact *Contact) {
    TListBoxItem *ListBoxItem = new TListBoxItem(ListBox1);

        __try {

                ListBoxItem->Text = Contact->DisplayName;
                ListBoxItem->Tag = Contact->ID;
                ListBox1->AddObject(ListBoxItem);
        }
        __finally {
                ListBoxItem->Free();
        }
}
```

## Implementing the Control Elements Functionality

To complete the application development, you should implement event handlers for all control elements that you have dropped onto the application form.

### To implement the OnClick event handlers for speed buttons:

1. On the Form Designer, double-click a speed button (**btnRefresh** or **btnRemove**).
2. In the Code Editor, specify the following event handlers for each button:

**Delphi**:

```
//------------------For Remove button ----------------------------------------
procedure TForm1.btnRemoveClick(Sender: TObject);
var
  ContactIndex, ContactID: Integer;
  Contact: TAddressBookContact;
begin
  ContactIndex := ListBox1.ItemIndex;
  if (ContactIndex > -1) Then
  begin
```

```
    ContactID := ListBox1.ListItems[ContactIndex].Tag;
    Contact := AddressBook1.ContactByID(ContactID);
    if Contact <> nil then
      try
        AddressBook1.RemoveContact(Contact);
        ListBox1.BeginUpdate;
        ListBox1.Items.Delete(ContactIndex);
      finally
        ListBox1.EndUpdate;
        Contact.Free;
      end;
  end;
end;
// -------------------For Refresh button-------------------------------------
procedure TForm1.btnRefreshClick(Sender: TObject);
begin
  FillContactsList;
end;
```

**C++Builder:**

```
//-------------------For Remove button -----------------------------------------
// Remove the contact currently selected from the List box
void __fastcall TForm1::btnRemoveClick(TObject *Sender) {
        int ContactIndex = ListBox1->ItemIndex;
        if (ContactIndex > -1) {
                int ContactID = ListBox1->ListItems[ContactIndex]->Tag;
                TAddressBookContact *Contact = AddressBook1->ContactByID(ContactID);
                if (Contact != NULL) {
                        __try {
                                AddressBook1->RemoveContact(Contact);
                                ListBox1->BeginUpdate();
                                ListBox1->Items->Delete(ContactIndex);

                        }
                        __finally {
                                ListBox1->EndUpdate();
                                Contact->Free();

                        }
                }
        }
}
// --------------------------For Refresh button-------------------------------------
void __fastcall TForm1::btnRefreshClick(TObject *Sender) {
        FillContactsList();
}
```

**To implement the onClick event handlers for the Master Pane buttons:**

1. In the Structure View, select **MultiView1**.
2. In the Object Inspector, set the **Visible** property to `True`.
   *This makes the Master Pane visible.*
3. On the Master Pane, double-click the **Add** button, and then implement the following event handler:

- For **Delphi**:

```
// ---------------------------------------------------------------------
// Add a newly created contact to Address Book
procedure TForm1.btnOKClick(Sender: TObject);
var
  Contact: TAddressBookContact;
  eMails: TContactEmails;
  Addresses: TContactAddresses;

begin
  Contact := AddressBook1.CreateContact(AddressBook1.DefaultSource);
  try
    Contact.FirstName := edtFirstName.Text;
    Contact.LastName := edtLastName.Text;
    // Add the work mail
    eMails := TContactEmails.Create;
    try
      eMails.AddEmail(TContactEmail.TLabelKind.Work, edtWorkMail.Text);
      Contact.eMails := eMails;
    finally
      eMails.Free;
    end;
    AddressBook1.SaveContact(Contact);
    try
      ListBox1.BeginUpdate;
      AddListBoxItem(Contact);
    finally
      ListBox1.EndUpdate;
    end;
  finally
    Contact.Free;
  end;
  MultiView1.HideMaster;

end;
```

- For **C++**:

```
// Add a newly created contact to Address Book
void __fastcall TForm1::btnOKClick(TObject *Sender) {

        TContactEmails *eMails;
        TAddressBookContact *Contact =
                AddressBook1->CreateContact(AddressBook1->DefaultSource());

        __try {

                Contact->FirstName = edtFirstName->Text;
                Contact->LastName = edtLastName->Text;
                // Add the work mail
                eMails = new TContactEmails();
                __try {
                        eMails->AddEmail(TContactEmail::TLabelKind::Work,
                        edtWorkMail->Text);
                        Contact->EMails = eMails;
                }
```

```
                __finally {
                        eMails->Free();
                }
                AddressBook1->SaveContact(Contact);
                __try {
                        ListBox1->BeginUpdate();
                        AddListBoxItem(Contact);
                }
                __finally {
                        ListBox1->EndUpdate();
                }

        }
        __finally {
                Contact->Free();
        }
        MultiView1->HideMaster();
}
```

4. On the Master Pane, double-click the **Cancel** button, and then implement the following event handler:

    - For **Delphi**:

```
// Clear and close the Add New Contact form
procedure TForm1.btnCancelClick(Sender: TObject);
begin
  edtFirstName.Text := '';
  edtLastName.Text := '';
  edtWorkMail.Text := '';
  MultiView1.HideMaster;
end;
```

    - For **C++**:

```
// Clear and close the Add New Contact form
void __fastcall TForm1::btnCancelClick(TObject *Sender)
{
        edtFirstName->Text = "";
        edtLastName->Text = "";
        edtWorkMail->Text = "";
        MultiView1->HideMaster();
}
```

## Keeping Address Book in Sync

Perhaps some third-party tools make changes to a device Address Book when your application is running. In this scenario, it is important to keep information synchronized between the device Address Book and the instance of the Address Book with which your application is directly working. For this purpose, FireMonkey provides the TAddressBook.RevertCurrentChangesAndUpdate method.

### To keep Address Book in Sync

1. In the Object Inspector, select **AddressBook1**, and then open the **Events** tab.
2. Double-click next to **OnExternalChange**, and then implement the following event handler:
   - For **Delphi**:

```
procedure TForm1.AddressBook1ExternalChange(ASender: TObject);
begin
  AddressBook1.RevertCurrentChangesAndUpdate;
  FillContactsList;
end;
```

   - For **C++:**

```
void __fastcall TForm1::AddressBook1ExternalChange(TObject *ASender) {
        AddressBook1->RevertCurrentChangesAndUpdate();
        FillContactsList();
}
```

## Configuring Access to Address Book

Before using TAddressBook, your application should request a permission to access the Address Book on a target mobile device.

This section provides the steps to configure your Android and iOS applications to access the Address Book on the target mobile devices.

### To request a permission to access the Address Book:

1. In the Object Inspector, select **Form1**, and then open the **Events** tab.
2. Double-click next to **onShow**, and then implement the following event handler:
   - For **Delphi**:

```
procedure TForm1.FormShow(Sender: TObject);
begin
  // Display this information box while loading the contacts
  if AddressBook1.Supported then
  begin
    TDialogService.ShowMessage('Loading contacts...');
    AddressBook1.RequestPermission;
  end
  else
    ShowMessage('This platform does not support the Address Book service');

end;
```

- For **C++**:

```
void __fastcall TForm1::FormShow(TObject *Sender) {
        if (AddressBook1->Supported()) {
                // Display this information box while loading the contacts
                TDialogService::ShowMessage("Loading contacts...");
                AddressBook1->RequestPermission();
        }
        else {
                TDialogService::ShowMessage
                        ("This platform does not support the Address Book
service.");
        }
}
```

**Note** To use the TDialogService.ShowMessage method, ensure that your code includes the following instructions:

**Delphi**:

```
uses
  FMX.DialogService;
```

**C++**:

```
#include <FMX.DialogService.hpp>
```

3. In the Object Inspector, select **AddressBook1**, and then open the **Events** tab.
4. Double-click next to **OnPermissionRequest**, and then implement the following event handler:

- For **Delphi**:

```
procedure TForm1.AddressBook1PermissionRequest(ASender: TObject;
  const AMessage: string; const AAccessGranted: Boolean);
begin
 if AAccessGranted then
   Begin
      FillContactlist;
   End
                  else
      ShowMessage('You cannot access Address Book. Reason: ' + AMessage);


end;
```

- For **C++**:

```
void __fastcall TForm1::AddressBook1PermissionRequest(TObject *ASender,
        const UnicodeString AMessage, const bool AAccessGranted) {

        if (AAccessGranted) {
                FillContactsList();

        }
        else {
                ShowMessage("You cannot access Address Book. Reason: " +
AMessage);
        }
}
```

### Configuring Android Applications to Access the Address Book

Before running Android applications that use the TAddressBook component, you need to configure some Uses Permissions for your project.

**To configure Uses Permissions:**

1. In the RAD Studio IDE, open **Project > Options > Uses Permissions**.
2. Enable the following permissions:
   - **Get accounts**
   - **Read contacts**
   - **Write contacts**

## Running the Sample Application

To run this application, do the following:

1. In the Project Manager, select the target platform (supported platforms: Android or iOS).

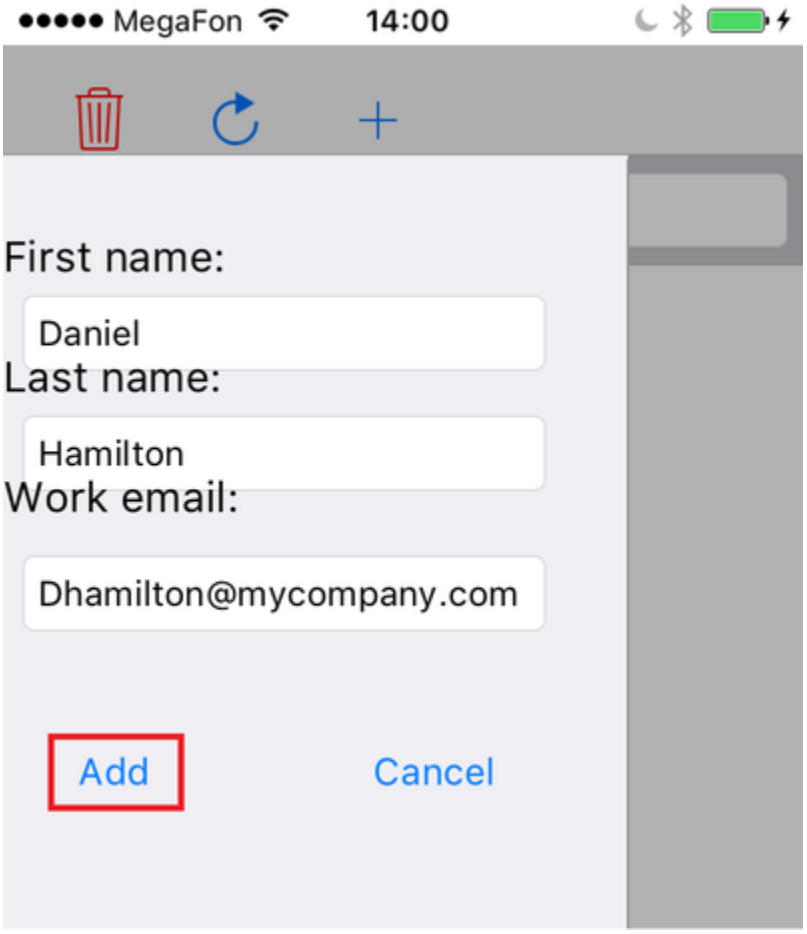
   **Important**: Before running this application on Android devices, ensure that you have completed the steps from Configuring Android Applications to Access the Address Book.

2. Press `Shift+Ctrl+F9` to run the application without debugging.

To test your application, you can use the following scenarios.

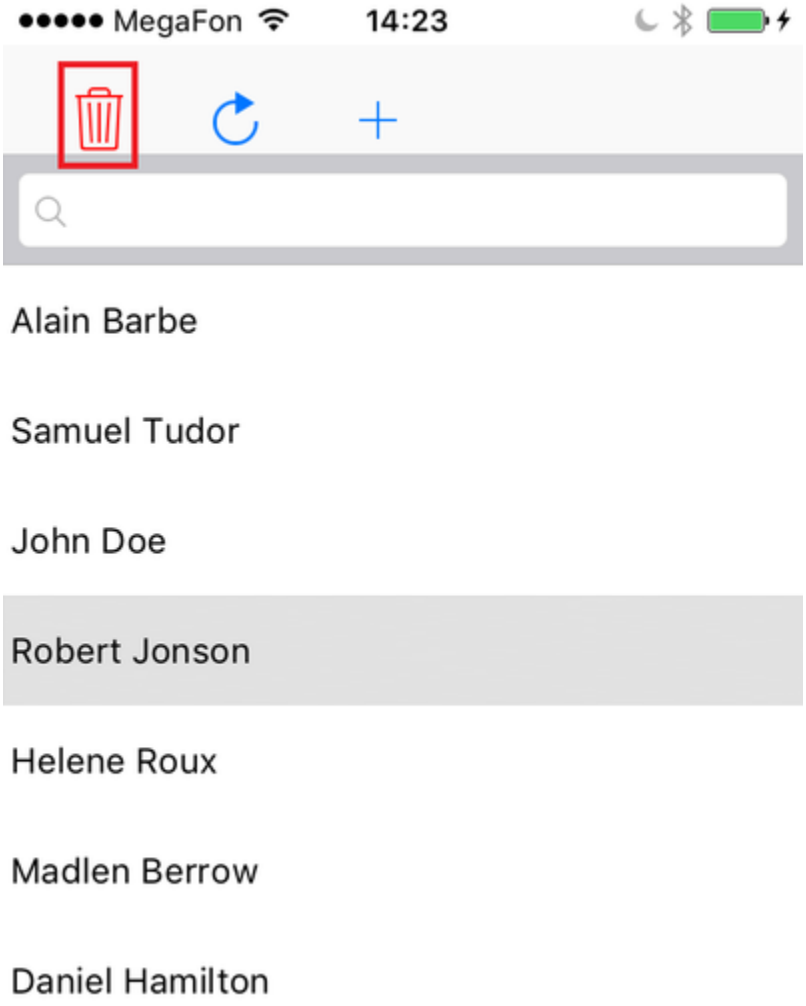
**To add a new contact to Address Book**

1. On the application toolbar, tap  .
   *This opens the Master Pane.*
2. In the Master Pane, enter appropriate information in the **First name**, **Last name**, and **Work mail** text boxes. When finished, tap **Add** to add the newly created contact to Address Book.




### To delete a contact

1. In the contacts list, tap a contact to delete.
2. On the application toolbar, tap 
   **Attention!** *This removes the contact from the device Address Book without displaying a confirmation prompt!*




## See Also

- TAddressBook
- Mobile Tutorial: Using a MultiView Component to Display Alternate Views of Information (iOS and Android)
- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

### Code Samples

- Birthday Reminder Sample
- Address Book Sample

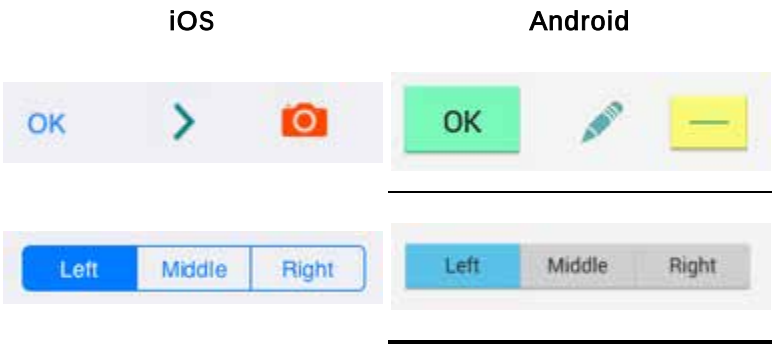
# Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)

## Buttons in Mobile Platforms

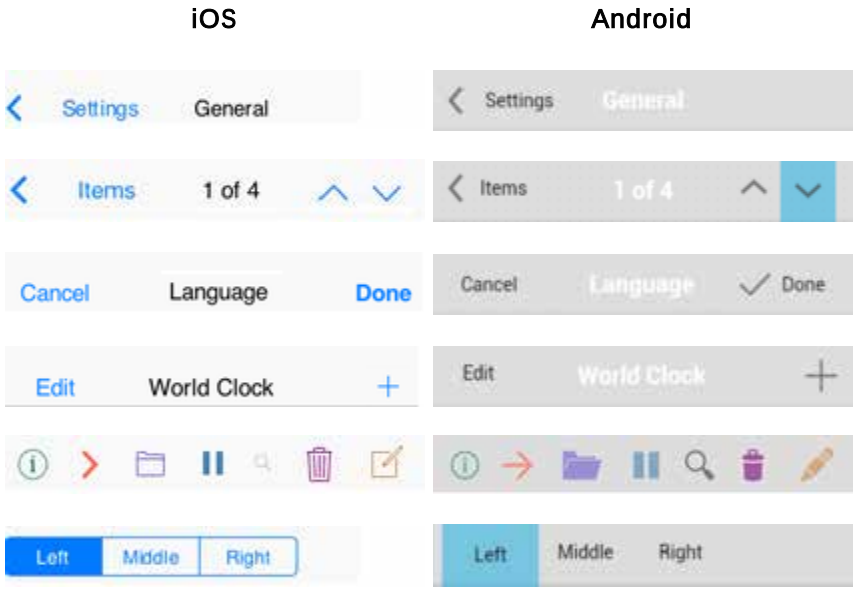
FireMonkey defines various types of buttons, and you can use these different types of buttons with the same steps described here. The FireMonkey buttons include TButton and TSpeedButton.

Following are some examples of different styles with Button components available for you to use in different parts of the user interface of your application:

- **Buttons on the Form:**




- **Buttons on the Navigation Bar** (also known as Toolbar):



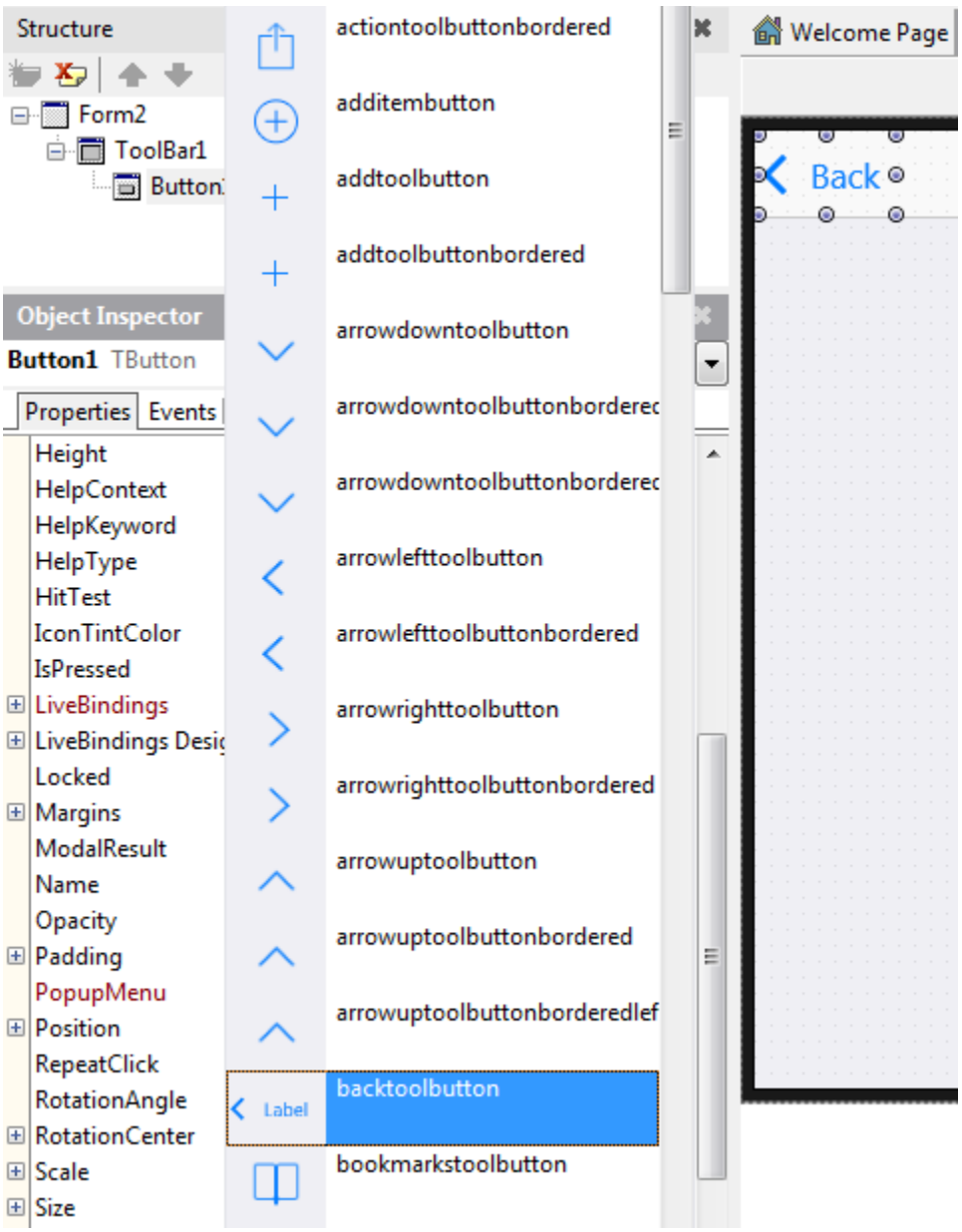

# Define the Look and Feel for a Button Component

After you place a new button on the Form Designer, you can specify some important properties for a selected component by using the Object Inspector.

Select a component (in this case, a button), and then browse and change the value of some properties as follows:

- Change the text displayed on the button surface by updating the value of the **Text** property in the Object Inspector.
- Change the value of the **Position.X** and **Position.Y** properties (or drag the component using your mouse.)
- Change the value of the **Height** and/or **Width** properties (or drag the edge of the component using your mouse.)
- Click the down-arrow in the **StyleLookup** property.

In the **StyleLookup** drop-down list, you can select a predefined Style based on how your component is to be used:




- To create a colored button, change the values of the **TintColor** and **IconTintColor** properties. The latter property is available only for styled buttons with icons. The next section gives more details about using **TintColor** and **IconTintColor**.

## Using TintColor and IconTintColor on Buttons

For TButton and TSpeedButton, FireMonkey provides two properties that determine how to tint or color the button:

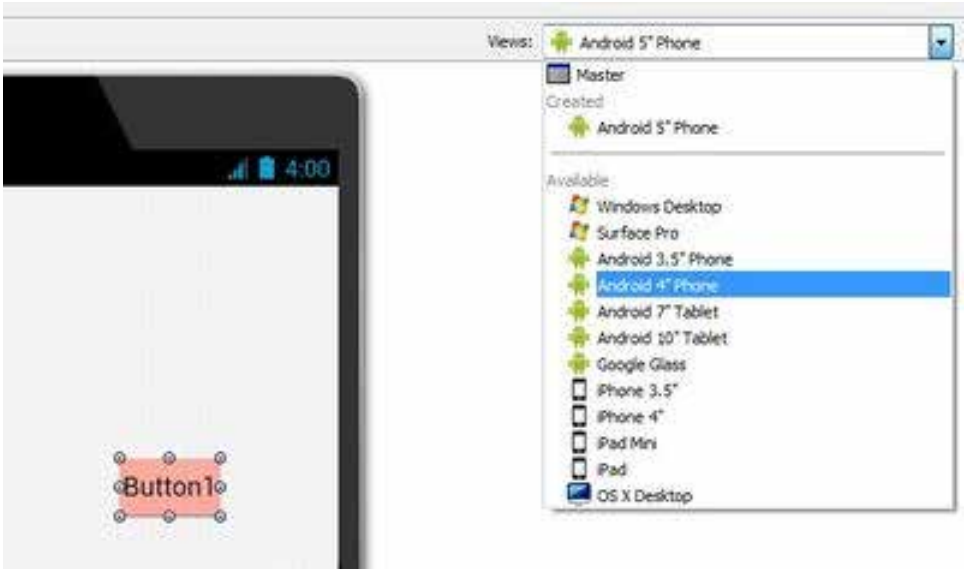
- TintColor specifies the button background color.
- IconTintColor specifies the color of the icon on styled buttons.

**Note**: The **TintColor** and **IconTintColor** properties are only available in the Object Inspector if you select a proper Style for the button and select a proper View in the Form Designer (these properties are not visible in all **Views**).

**For the Android target platform:**

You can apply a tint to most buttons of any style:

- For speed buttons, you need to select a proper StyleLookup value in order to change the **TintColor** value in the Object Inspector.
- To modify **TintColor** and **IconTintColor**, choose an Android device from the **Views** list in the upper right of the Form Designer.




**For the iOS target platforms:**

- FireMonkey provides buttons that correspond to the Apple Style Guide, and some buttons might not support the tint feature.
- For example, on iOS, segmented buttons have the **TintColor** property.

When you change the **StyleLookup** property of a button, the Object Inspector automatically displays or hides the **TintColor** and **IconTintColor** properties as appropriate for the **StyleLookup** property value. The following image shows three TSpeedButtons on an Android app:

## Using Styled and Colored Buttons on Target Platforms

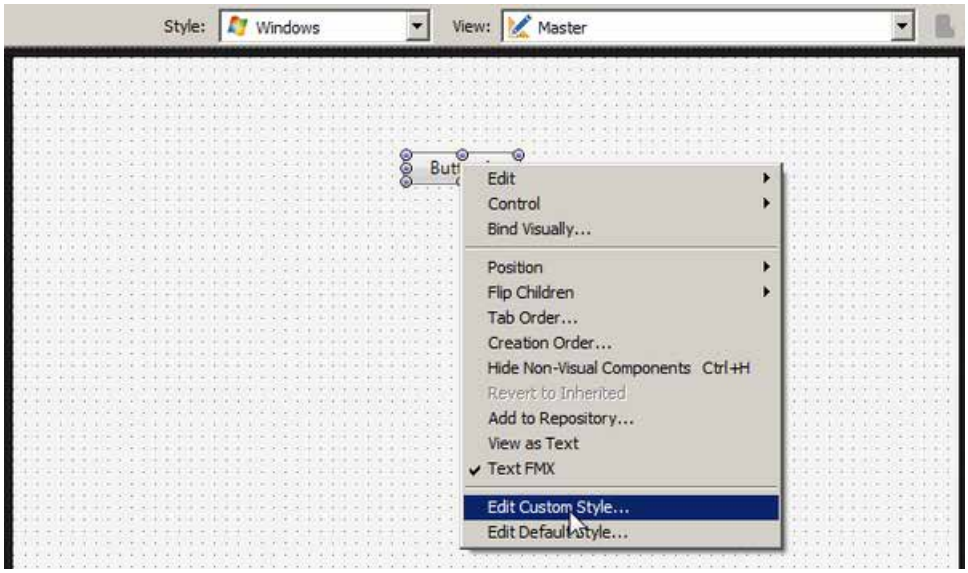
For information on the availability of the **StyleLookup**, **TintColor**, and **IconTintColor** properties for buttons on all target platforms, see this summary table: **Using Styled and Colored Buttons on Target Platforms**.

## Customizing Buttons with Styles

Now in FireMonkey, a TStyleBook component is a collection of styles for various target platforms. You can create your custom styles for individual buttons or for a whole control class (TButton or TSpeedButton).

### To start working with custom styles for buttons

1. In the Form Designer, ensure that you have selected **Master** from the View selector.
2. On the Form Designer, right-click a button or speed button, then choose on the shortcut menu one of the following items:
   - **Edit Custom Style**: opens the FireMonkey Style Designer to edit the styles for selected button.
   - **Edit Default Style**: opens the FireMonkey Style Designer to edit the styles for the selected control class (such as TButton or TSpeedButton).




For more information about working with custom styles, see the following topics:

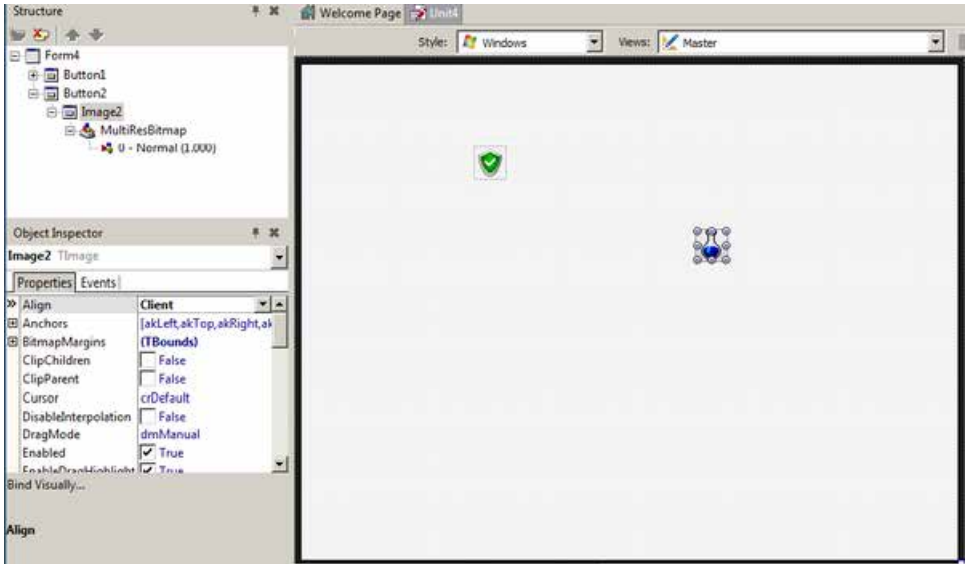
- Customizing FireMonkey Applications with Styles
- FireMonkey Style Designer
- BitmapLinks Editor

## Placing an Image over a Button

RAD Studio allows you to easily put custom images on button components at design time.

**To place an image over a button:**

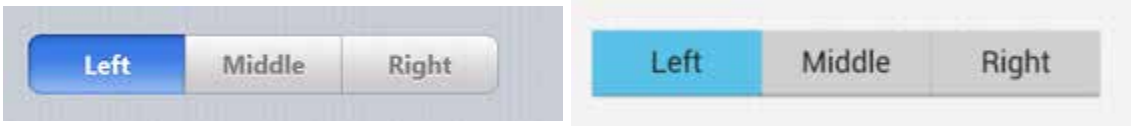
1. With a TButton and TImage component on the Form Designer, make TImage a child component of TButton. (Use the Structure View.)
2. In the Object Inspector, select **TImage** and set its Align property to `Client`.
3. In the Structure View, select the button, expand the **Image** node, and then click **0 - Empty (1.000)**.
4. In the Object Inspector, click the ellipsis button (**...**) next to **Bitmap**.
5. Add your custom image in the MultiResBitmap Editor.
6. In the Object Inspector, select **TButton**, and do the following:
   - Clear the **Text** property.
   - Set the Height and Width properties of **TButton** to the actual height and width of your image.




# Create a Segmented Control Using Button Components

FireMonkey uses a SpeedButton component to define the **Segmented Control**, which gives users the ability to select one value from several options.

**iOS** **Android**




**To define a Segmented Control, use the following steps:**

1. Place three **TSpeedButton** components from the Tool Palette. Place the TSpeedButton components next to each other using your mouse:

iOS


Android

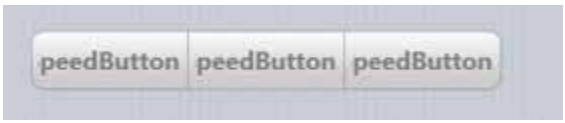

2. Select the first component, and change its **StyleLookup** property to **segmentedbuttonleft**:

iOS

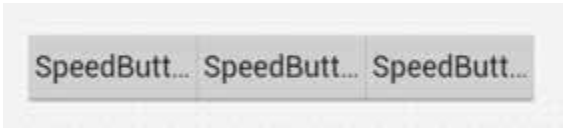

Android


3. Select the second component, and change its StyleLookup property to **segmentedbuttonmiddle**.
4. Select the third component, and change its StyleLookup property to **segmentedbuttonright**. Now all three buttons look like a Segmented Control:

iOS

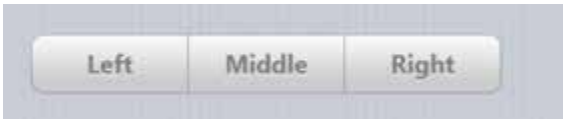



Android




5. Select each component, and change the **Text** property as you like:

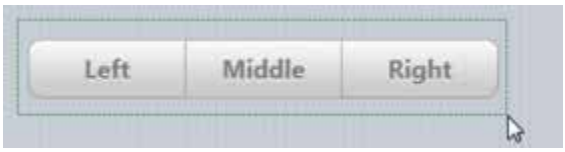
iOS




Android


6. Use the mouse to select these three buttons:

iOS




Android


7. Set the GroupName property to a unique name such as **LocationSegments**:


8. To specify that one of these components is to appear as **Pressed** by default, set the IsPressed property for one component to **True**:

iOS


Android


# Create a Scope Bar on a Toolbar Component

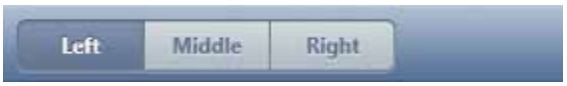
You can define a Segmented Control on a toolbar; this is also known as a **Scope Bar**, a segmented control that can be used to control the scope of a search.

Use the same TSpeedButton controls as in the previous steps, but with the following values for the StyleLookup property (only available on iOS target platform):

- **toolbuttonleft**
- **toolbuttonmiddle**
- **toolbuttonright**

(on the Android target platform set StyleLookup as **toolbutton** for each of the buttons.)

iOS




Android


# Important Differences Between a TButton and TSpeedButton

- TSpeedButton cannot receive TAB focus. That also means that pressing a TSpeedButton does not take away focus from other elements.
- TSpeedButton is primarily meant to be a toolbar button.
- The style of TSpeedButton can be different than the style of TButton. For example, on the Android platform, the default style of the TSpeedButton is similar to the **toolbutton** style of the TButton. If you want a TSpeedButton with a similar style to the TButton, choose the **buttonstyle** style.

# See Also

- Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android)
- Mobile Tutorial: Using a Calendar Component to Pick a Date (iOS and Android)
- Tutorial: Using the BitmapLinks Editor
- FireMonkey Style Designer
- Android Mobile Application Development
- FMX.Controls Sample
- FMX.StdCtrls.TButton
- FMX.StdCtrls.TSpeedButton
- FMX.Controls.TStyledControl.StyleLookup
- FMX.StdCtrls.TToolBar
- FMX.StdCtrls.TCustomButton.IconTintColor
- FMX.StdCtrls.TCustomButton.TintColor
- FMX.StdCtrls.TToolBar.TintColor

# Mobile Tutorial: Using a Calendar Component to Pick a Date (iOS and Android)

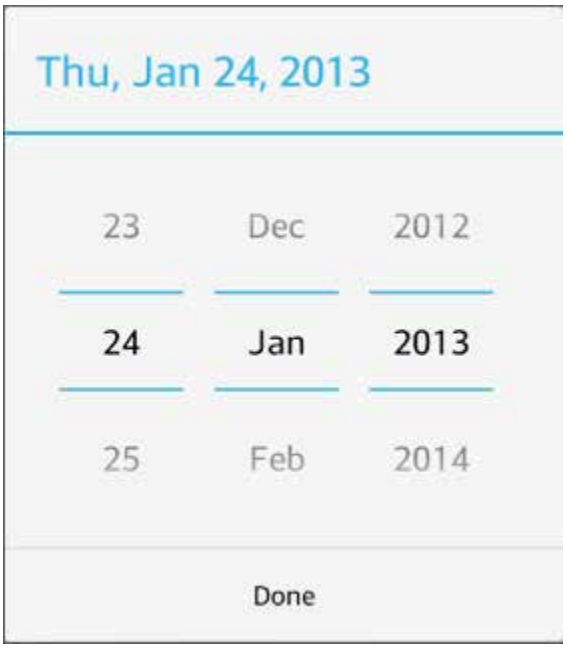
## Calendar in Mobile Platforms

FireMonkey uses the **TDateEdit** component to wrap a calendar component or datepicker for the mobile target platform:

**iOS7**


**iPad2**

**Android**

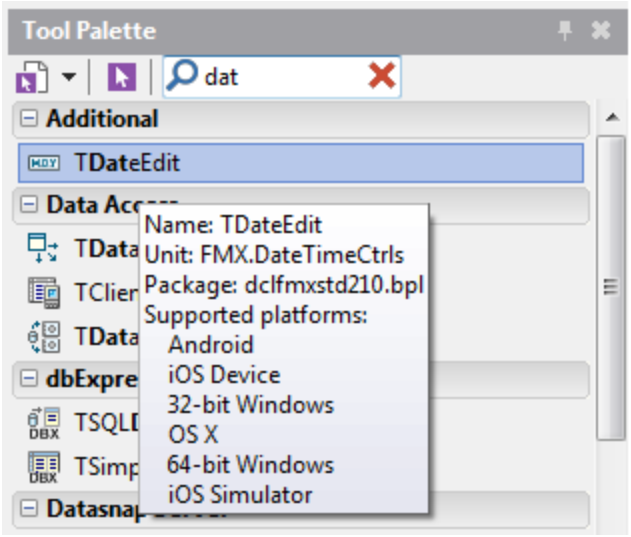



**LG-E612**

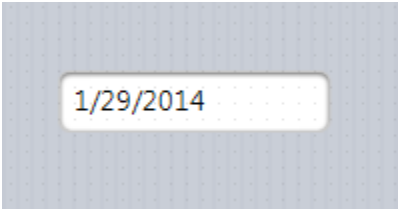
> **Note:** The TCalendarEdit component used in RAD Studio XE5 or earlier is deprecated. Use the TDateEdit component instead.

To use the TDateEdit component, perform the following simple steps:

1. Select the TDateEdit component in the Tool Palette, and drop the component onto the Form Designer. To find the component in the Tool Palette, enter the first few characters (such as "dat") in the search box ( ):

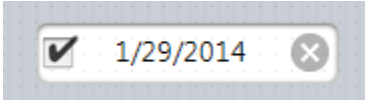



After you drop the component, you can see the **TDateEdit** component on the Form Designer:



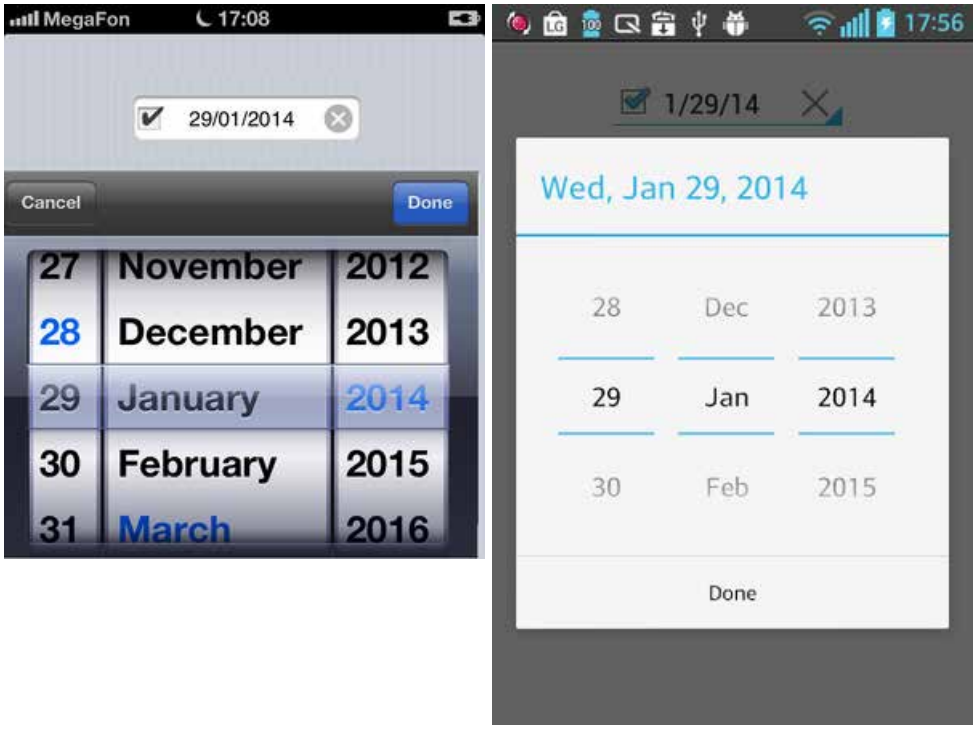

Optionally, in the Object Inspector, you can set the following properties of **TDateEdit**:

- ShowCheckBox: when `true`, displays a checkbox on the **TDateEdit** control. This checkbox allows you to enable/disable the **TDateEdit** control at run time.
- ShowClearButton: when `true`, displays a button on the **TDateEdit** control. Click this button to clear values in this control at run time.




2. Basically, that's it. Run your application on either a simulator or your connected mobile device. After you tap **TDateEdit**, the calendar control appears, and you can select a date.

iOS6 (iPhone5) Android (LG-E612)




# Implementing an Event Handler for User Changes to the Date

After the user changes the date, the OnChange event is fired. You can implement an event handler for the **OnChange** event to react to the user's action.

**To implement the OnChange event handler'**

1. Select the **TDateEdit** component.
2. In the Object Inspector, open the **Events** page, and double-click the empty space next to **OnChange**.
3. Write code as follows:

**Delphi:**

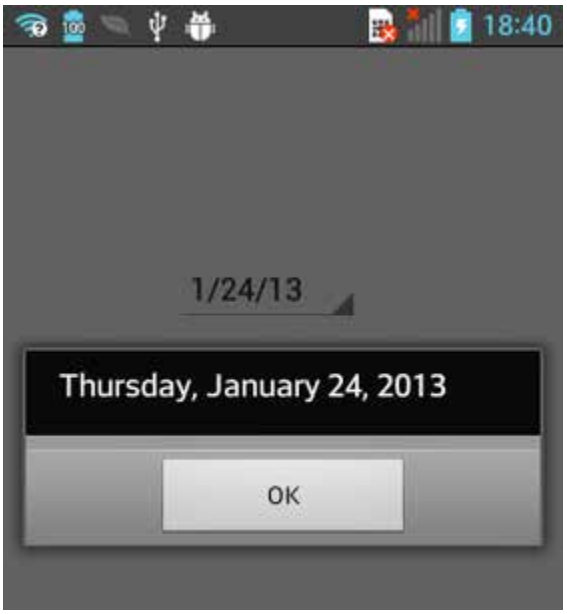
```
procedure TForm25.DateEdit1Change(Sender: TObject);
begin
     ShowMessage(FormatDateTime('dddddd', DateEdit1.Date));
end;
```

**C++Builder:**

```
void __fastcall TForm25::DateEdit1Change(TObject *Sender)
{
     ShowMessage(FormatDateTime("dddddd", DateEdit1->Date));
}
```

This code shows a message dialog with a date selected. The FormatDateTime function converts the selected date to a specified format (in this case *dddddd* gives long-style date format):

**iOS (iPad)** **Android (LG-E612)**


## See Also

- Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)
- Mobile Tutorial: Using Combo Box Components to Pick Items from a List (iOS and Android)
- Date and Time Support
- Type conversion routines
- FMX.DateTimeCtrls.TDateEdit

## Samples

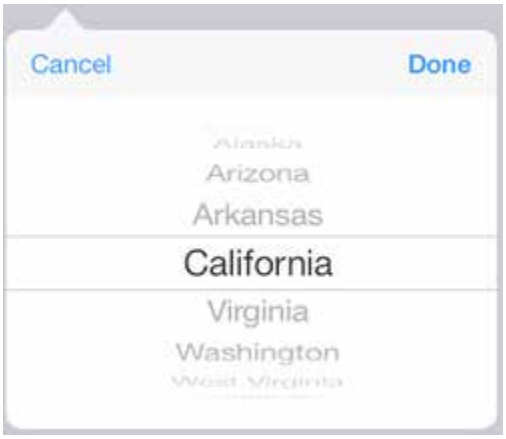
- Date Picker sample

# Mobile Tutorial: Using Combo Box Components to Pick Items from a List (iOS and Android)

## Implementing a Picker in Multi-Device Applications

For mobile platforms, FireMonkey wraps the Picker component with the TComboBox component:

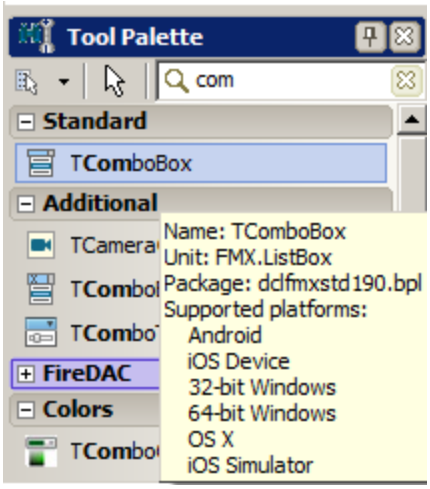
iOS



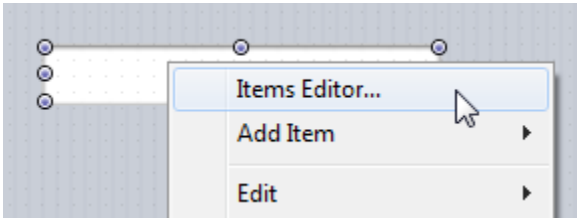

Android


**To define a picker and the associated list items:**

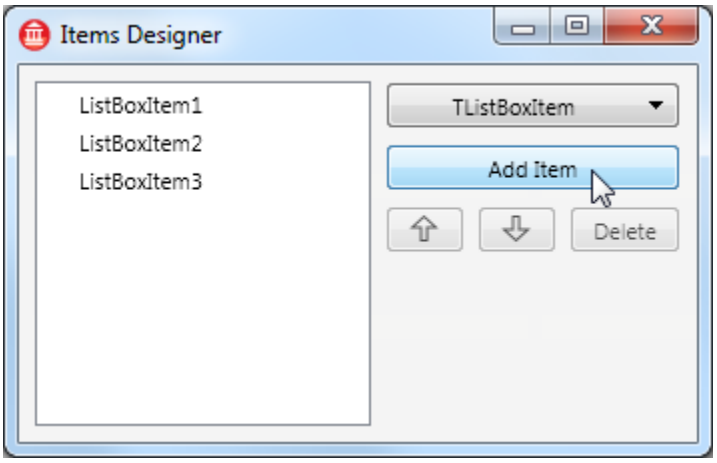
1. Select either of the following:
   - File > New > Multi-Device Application - Delphi > Blank Application
   - File > New > Multi-Device Application - C++Builder > Blank Application
2. Select the TComboBox component in the Tool Palette, and drop it on the Form Designer.
   To find TComboBox, enter the first few characters ("Com") in the Search box of the Tool Palette:



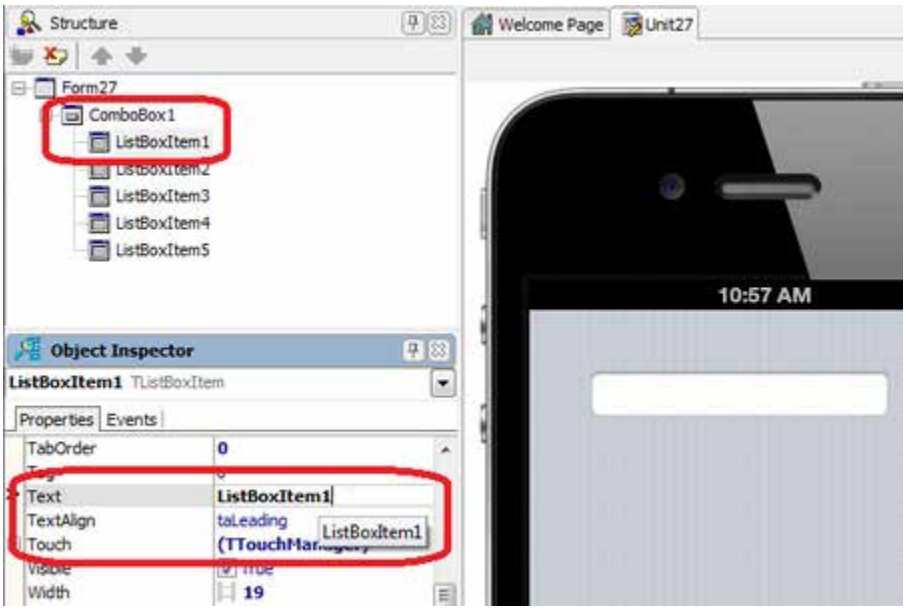

3. After you drop the component, you can see the TComboBox component on the Form Designer.
   Right-click the TComboBox component and select **Items Editor...**:




4. To define items, click **Add Item** several times.




5. In the Structure View, select **ListBoxItem1** (the first item in the list).
6. In the Object Inspector, edit the **Text** property for ListBoxItem1.
   In this example (the fifty states in the USA), enter "Alabama" as the first item in the list:




7. Edit other items as well, such as Alaska, Arizona, Arkansas, California, Colorado, and so forth.
8. Select the TComboBox component and in the Object Inspector, set the TComboBox.Align property to **Top**.
9. Add a second TComboBox (**ComboBox2**) to the form. Select the TComboBox component in the Tool Palette, and drop it again on the Form Designer.
10. Select **ComboBox2** and in the Object Inspector, set the TComboBox.Align property to **Bottom**.
11. Run the application on your chosen mobile target platform (iOS Simulator (for Delphi only), iOS Device, or Android Device).
    After you tap a TComboBox, the Picker control appears, and you can select an item.

# Building a List of Items Using Code

To build a list of items using code, you should implement the **onFormCreate** event handler in the following way:

**Delphi:**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox2.Items.Add('Tiger');
  ComboBox2.Items.Add('Cat');
  ComboBox2.Items.Add('Penguin');
  ComboBox2.Items.Add('Bee');
  // Other animals can be listed here
  ComboBox2.Items.Add('Elephant');
  ComboBox2.Items.Add('Lion');
end;
```

**C++Builder:**

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox2->Items->Add("Tiger");
  ComboBox2->Items->Add("Cat");
  ComboBox2->Items->Add("Penguin");
  ComboBox2->Items->Add("Bee");
  // Other animals can be listed here
  ComboBox2->Items->Add("Elephant");
  ComboBox2->Items->Add("Lion");
}
```

# Displaying a Specific Item

The currently selected item is specified by the ItemIndex property. ItemIndex is an integer value that is specified using a zero-based index (that is, the first item is zero).

To display the list with the fifth item selected ("California"), specify ItemIndex for ComboBox1 as follows:

**Delphi:**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Index of 5th item is "4"
  ComboBox1.ItemIndex := 4;

  ComboBox2.Items.Add("Tiger");
  ComboBox2.Items.Add("Cat");
  ComboBox2.Items.Add("Penguin");
  ComboBox2.Items.Add("Bee");
  // Other animals can be listed here
  ComboBox2.Items.Add("Elephant");
  ComboBox2.Items.Add("Lion");
end;
```

**C++Builder:**

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Index of 5th item is "4"
  ComboBox1->ItemIndex = 4;

  ComboBox2->Items->Add("Tiger");
  ComboBox2->Items->Add("Cat");
  ComboBox2->Items->Add("Penguin");
  ComboBox2->Items->Add("Bee");
  // Other animals can be listed here
  ComboBox2->Items->Add("Elephant");
  ComboBox2->Items->Add("Lion");
}
```

If you do not know the index value, you can find the value by using the **IndexOf** method. To display the **ComboBox2** with the item whose text is 'Penguin' selected, add the following line to the previous code:

**Delphi:**

```
ComboBox2.ItemIndex := ComboBox2.Items.IndexOf('Penguin');
```

**C++Builder:**

```
ComboBox2->ItemIndex = ComboBox2->Items->IndexOf("Penguin");
```

# Implementing an Event Handler for the User's Selection

After the user selects an item, the **OnChange** event is fired. To respond to the user's action, you can implement an event handler for the OnChange event.

**Note:** Before proceeding with this scenario, perform the following steps:

1. Select the TMemo component in the Tool Palette, and drop it on the Form Designer.
2. In the Object Inspector, set the TMemo.Align property to `Client`.

**To implement an OnChange event handler:**

1. Select the **ComboBox1** component.
2. In the Object Inspector, open the **Events** page, and double-click the empty space next to **OnChange**.
3. The Code Editor opens. Write code as follows:

**Delphi:**

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
Memo1.Lines.Insert(0, (Format('%s: Item %s at Index %d was selected. ',
[ComboBox1.Name,ComboBox1.Selected.Text, ComboBox1.ItemIndex])));
end;
```

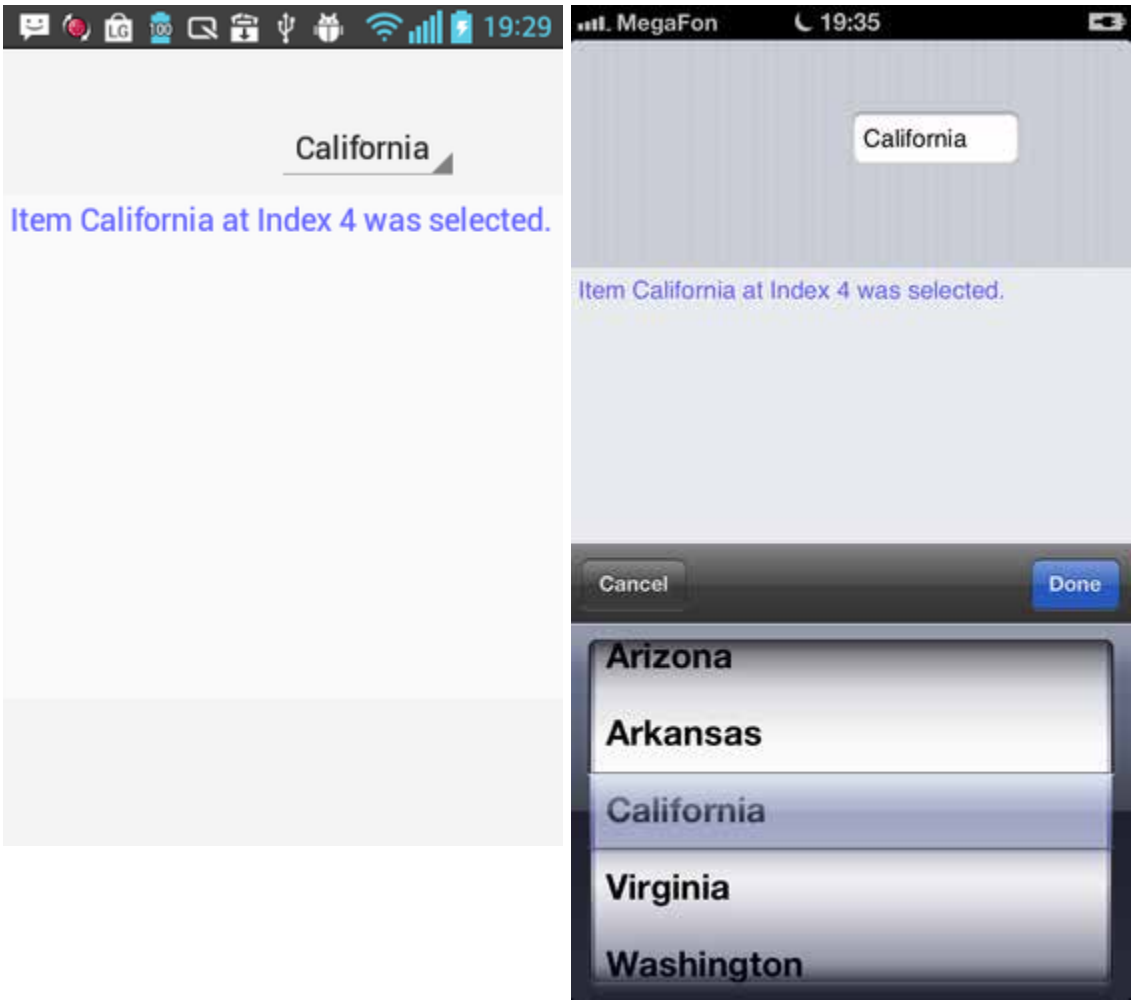
**C++Builder:**

```
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  Memo1->Lines->Insert(0, ComboBox1->Name + ": Item " + ComboBox1->Selected->Text + "
at Index " + IntToStr(ComboBox1->ItemIndex) + " was selected.");
}
```

This event handler displays a message dialog that indicates the item that was selected.

In the Delphi code, the Format function returns a formatted string assembled from a format string and an array of arguments:

**Android (LG - E612)** **iOS6 (iPad)**




## See Also

- Mobile Tutorial: Using a Calendar Component to Pick a Date (iOS and Android)
- Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)
- Creating a Metropolis UI ComboBox
- iOS Mobile Application Development
- OS X Application Development

## Samples

- FireMonkey Custom Picker sample

# Mobile Tutorial: Using a Map Component to Work with Maps (iOS and Android)

FireMonkey wraps a map component as **TMapView**. This component provides access to map APIs that depend on the target platform in the following way:

- On Android devices: Google Maps Android API
- On iOS devices: Map Kit Framework

This tutorial describes how to create a simple FireMonkey application that uses the **TMapView** component.

**iOS**


**iPad**

**Android**


**Android (LG - E612)**

## Basic Features of the TMapView Component

The TMapView component adds interactive maps to your mobile applications. The basic features of this component are as follows:

- **Four Types of Maps**: Normal, Satellite, Hybrid, and (for Android only) Terrain
- **Gesture Control**: Intuitive tilt, rotate, and zoom gesture controls
- **Control the Map View**: Ability to control the map properties, such as the map center coordinates, the map orientation, and so on

# Creating a Sample Application

This section helps you to develop a sample application (for Android and iOS target platforms) that illustrates the use of the TMapView component. The application demonstrates the following techniques:

- Selecting a map type
- Rotating the map
- Specifying the map center coordinates
- Adding markers to the map

## Configuring Android Applications to Use the TMapView component

Before using Google Maps, ensure that you have a **Google Maps Android API** key (freely available). Without this key in place, your map application will generate a run-time error.

You also need to configure some permissions and project options for your application.

For detailed instructions on how to configure your application, see **Configuring Android Applications to Use Google Maps**.

## Designing the User Interface

1. Create a blank Multi-Device Application, by selecting:
   - For **Delphi**: **File > New > Multi-Device Application - Delphi > Blank Application**
   - For **C++**: **File > New > Multi-Device Application - C++Builder > Blank Application**
2. Select two **TToolBar** components in the Tool Palette, and drop them on the Form Designer.
3. Select the **TMapView** component in the Tool Palette, and drop it on the Form Designer.
4. In the Object Inspector, set the **Align** property of **TMapView** to `Client`.
5. In the Object Inspector, set the **Align** properties of the toolbars to `Top` and `Bottom`, respectively.

## Designing the Application Toolbars

Place all control elements on the toolbars. The application uses two toolbars (a top toolbar, and a bottom toolbar).

### To design the top toolbar

- In the Tool Palette, select the following components and drop them onto the top toolbar:
  - TTrackBar: Rotates the map.
  - Two TEdit components: Allow you to set the map center coordinates (latitude and longitude).
  - TButton: Updates the map using the current map center coordinates.

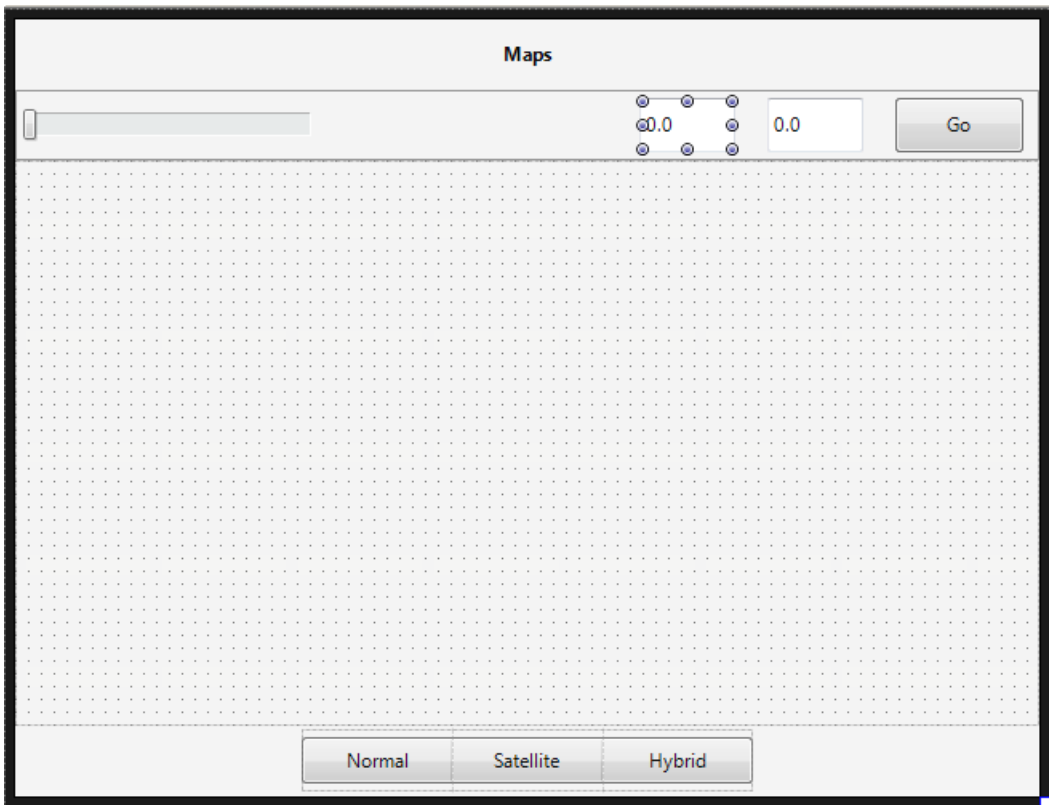
To specify the appropriate properties for control elements, do the following:

1. In the **Form Designer**, select **TrackBar1** and set the Max property to `359`.
2. Select **Edit1**, and set the **Name** and **Text** properties to `edLat` and `0.0`, respectively.
3. Select **Edit2**, and set the **Name** and **Text** properties to `edLong` and `0.0`, respectively.
4. Select **Button1**, and set the **Text** property to `Go`.

### To design the bottom toolbar

1. In the Tool Palette, select the **TLayout** component and drop it onto the bottom toolbar.
2. In the Object Inspector, specify the following properties of **Layout1**:
   - Set the **Align** property to `Center`.
   - Set the **Width** property to 241.
3. In the Tool Palette, select three **TSpeedButton** components, and add them as child elements of **Layout1**.
4. In the Object Inspector, specify the following properties of the speed buttons:
   - Set the **Text** property of buttons to `Normal`, `Satellite`, and `Hybrid`, respectively.
   - Set the **GroupName** property of each button to `Selector`.
   - Set the **StyleLookup** property to `segmentedbuttonleft`, `segmentedbuttonmiddle`, and `segmentedbuttonright`, respectively.

After performing the above steps, your Form Designer will be similar to the following screen:




## Implementing the Control Elements Functionality

To complete the application development, you should implement event handlers for all control elements that you have dropped onto the toolbars.

### To implement the OnClick event handlers for speed buttons

1. On the Form Designer, double-click a speed button (**Normal**, **Satellite**, and **Hybrid**).
2. In the Code Editor, specify the following event handlers for each button:

   **Delphi**:

```
//------------------For Normal button ----------------------------------------
-
procedure TForm1.SpeedButton1Click(Sender:TObject) ;
begin
        MapView1.MapType := TMapType.Normal;
        TrackBar1.Value := 0.0;
end;
// ------------------For Satellite button-------------------------------------
--

procedure TForm1.SpeedButton2Click(Sender:TObject) ;
begin
        MapView1.MapType := TMapType.Satellite;
        TrackBar1.Value := 0.0;
end;
// --------------------For Hybrid button-----------------------------------
--
```

```
procedure TForm1.SpeedButton3Click(Sender:TObject) ;
begin
        MapView1.MapType := TMapType.Hybrid;
        TrackBar1.Value := 0.0;
end;
```

C++Builder:

```
//-------------------For Normal button ----------------------------------------
-
void __fastcall TForm1::SpeedButton1Click(TObject *Sender) {
        MapView1->MapType = TMapType::Normal;
       TrackBar1->Value = 0.0;
}
// -------------------For Satellite button-----------------------------------
--

void __fastcall TForm1::SpeedButton2Click(TObject *Sender) {
        MapView1->MapType = TMapType::Satellite;
        TrackBar1->Value = 0.0;
}
// --------------------For Hybrid button-----------------------------------
--

void __fastcall TForm1::SpeedButton3Click(TObject *Sender) {
        MapView1->MapType = TMapType::Hybrid;
        TrackBar1->Value = 0.0;
}
```

### To implement the OnChange event handler for the track bar element

1. On the Form Designer, select TrackBarl1.
2. In the Object Inspector, open the **Events** tab, and then double-click next to **onChange**. Specify the following code:

   **Delphi**:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  MapView1.Bearing := TrackBar1.Value;
end;
```

   **C++Builder:**

```
void __fastcall TForm1::TrackBar1Change(TObject *Sender)
{
   MapView1->Bearing = TrackBar1->Value;
}
```

### To implement the OnClick event handler for the Go Button

1. On the Form Designer, double-click the **Go** button.
2. In the Code Editor, specify the following code:

   **Delphi**:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  mapCenter: TMapCoordinate;
begin
  mapCenter := TMapCoordinate.Create(StrToFloat(edLat.Text),
    StrToFloat(edLong.Text));
  MapView1.Location := mapCenter;
end;
```

   **C++Builder:**

```
void __fastcall TForm1::Button1Click(TObject *Sender) {
        TMapCoordinate mapCenter = TMapCoordinate::Create(StrToFloat(edLat-
>Text),
                StrToFloat(edLong->Text));
        MapView1->Location = mapCenter;
}
```

Markers identify locations on the map. If you want to add markers to the map, you can implement the **OnMapClick** event handler for the map in the following way.

### To implement the OnMapClick event handler for the map

1. In the Structure View, select **MapView1**.
2. In the Object Inspector, open the **Events** tab, and double-click next to **OnMapClick**.
3. In the Code Editor, implement the following event handler:

   **Delphi**:

```
procedure TForm1.MapView1MapClick(const Position: TMapCoordinate);
var
  MyMarker: TMapMarkerDescriptor;
begin
  MyMarker := TMapMarkerDescriptor.Create(Position, 'MyMarker');
  // Make a marker draggable
  MyMarker.Draggable := True;
  // Make a marker visible
  MyMarker.Visible :=True;
  MapView1.AddMarker(MyMarker);
end;
```

   **C++Builder:**

```
```

```
void __fastcall TForm1::MapView1MapClick(const TMapCoordinate &Position) {
        TMapMarkerDescriptor myMarker = TMapMarkerDescriptor::Create(Position,
"MyMarker");
        // Make a marker draggable
        myMarker.Draggable = true;
        // Make a marker visible
        myMarker.Visible = true;
        MapView1->AddMarker(myMarker);
}
```

# Running the Sample Application

To run this application, do the following:

1. In the Project Manager, select the target platform (supported platforms: Android or iOS).

   **Important**: Before running this application on Android devices, ensure that you have completed the steps from Configuring Android Applications to Use Google Maps.

2. Press `Shift+Ctrl+F9` to run the application without debugging.

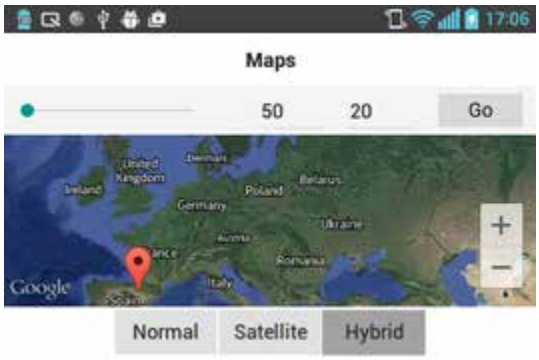
To test your application, you can use the following scenario:

1. Click the **Hybrid** button.
2. Specify the new map center (by default the map center is (0.0, 0.0)):
   - In the left text box, set the latitude value (such as 50 degrees).
   - In the right text box, set the longitude value (such as 20 degrees).
   - Click the **Go** button.
3. Click any point on the map to add a marker.
4. Tap the track bar element, and then move the slide indicator by dragging it to a particular location. [*This changes the map orientation (bearing). The map orientation is the direction in which a vertical line on the map points, measured in degrees clockwise from north.*]

iOS


iPad

Android




Android (LG - E612)

# See Also

- TMapView
- Configuring Android Applications to Use Google Maps
- Mobile Tutorial: Using Location Sensors (iOS and Android)

## Code Samples

- Tabbed Map Sample
- Map Sample

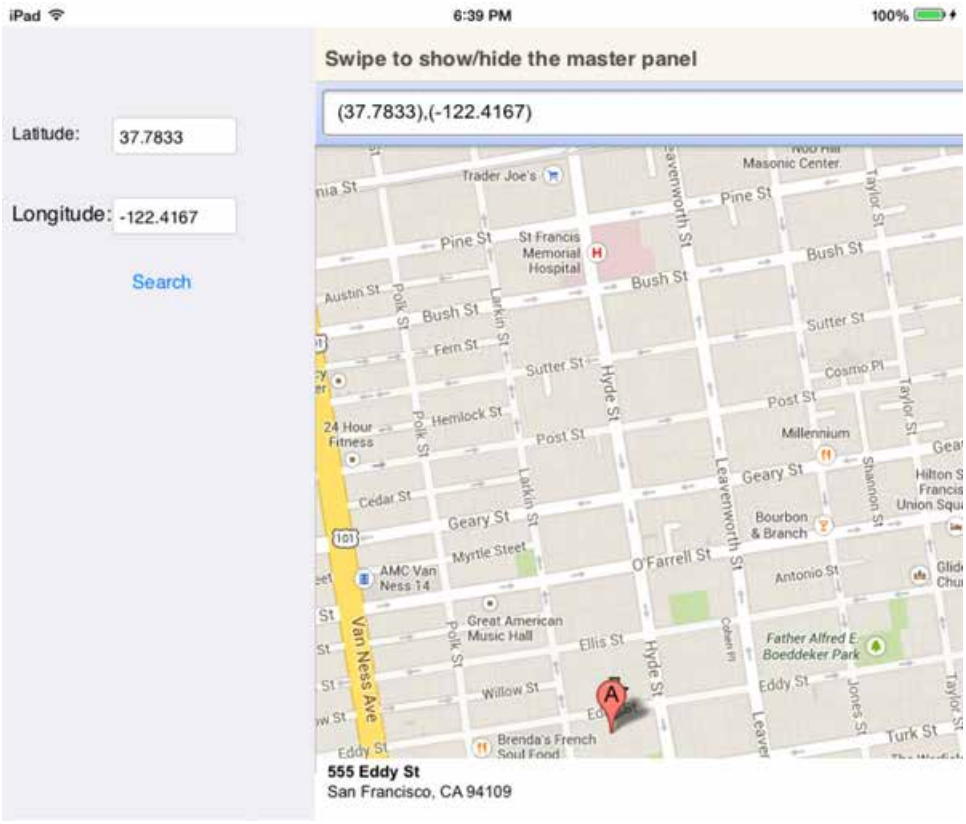
# Mobile Tutorial: Using a MultiView Component to Display Alternate Views of Information (iOS and Android)

## About the TMultiView Component

The **FMX.MultiView.TMultiView** component represents a container (the master pane) for other controls, and provides a way for you to easily present alternate views of information. The TMultiView component allows you to implement a master-detail interface, which can be used for any supported target platform.

- The **master pane** can display a collection of any visual controls, such as edit boxes, labels, lists, and so forth.
- The **detail pane** typically displays information based on the properties of the controls in the master pane.

The following screen illustrates an example master-detail interface. In the master pane (the left docked panel), you enter a geographical position, and then click **Search** to cause the detail pane (the right panel) to display the appropriate Google map.




## Master Pane Presentation Modes

The TMultiView class defines a set of properties that allow you to control the interface behavior. For example, the TMultiView.Mode property specifies the master pane presentation mode as described in the following table:

| TMultiView.Mode | Master Pane Presentation |
|---|---|
| Drawer | In the Drawer mode, the master pane can be whether hidden or can slide to overlap the detail pane. |
| Panel | Master and detail panels are always displayed, independently of a device type and orientation. The master panel is docked to the left or right of the MultiView component. |
| PlatformBehaviour | An application automatically selects the master pane presentation mode. See the Platform Dependent Behavior Mode subsection. |
| Popover | Pop-up menu. |
| NavigationPane | Navigation pane. |
| Custom | The user can customize the master pane presentation. The custom presentation class should be set in the CustomPresentationClass property. See the Custom Mode subsection. |

**Note**: At design time, after you change the TMultiView.Mode property value in the Object Inspector, the master pane might become invisible. To work around this issue, on the Form Designer, select the **TMultiView** component, and then in the Object Inspector, set the Visible property to `True`.

**Note**: At design time, you can show/hide TMultiView by two ways:

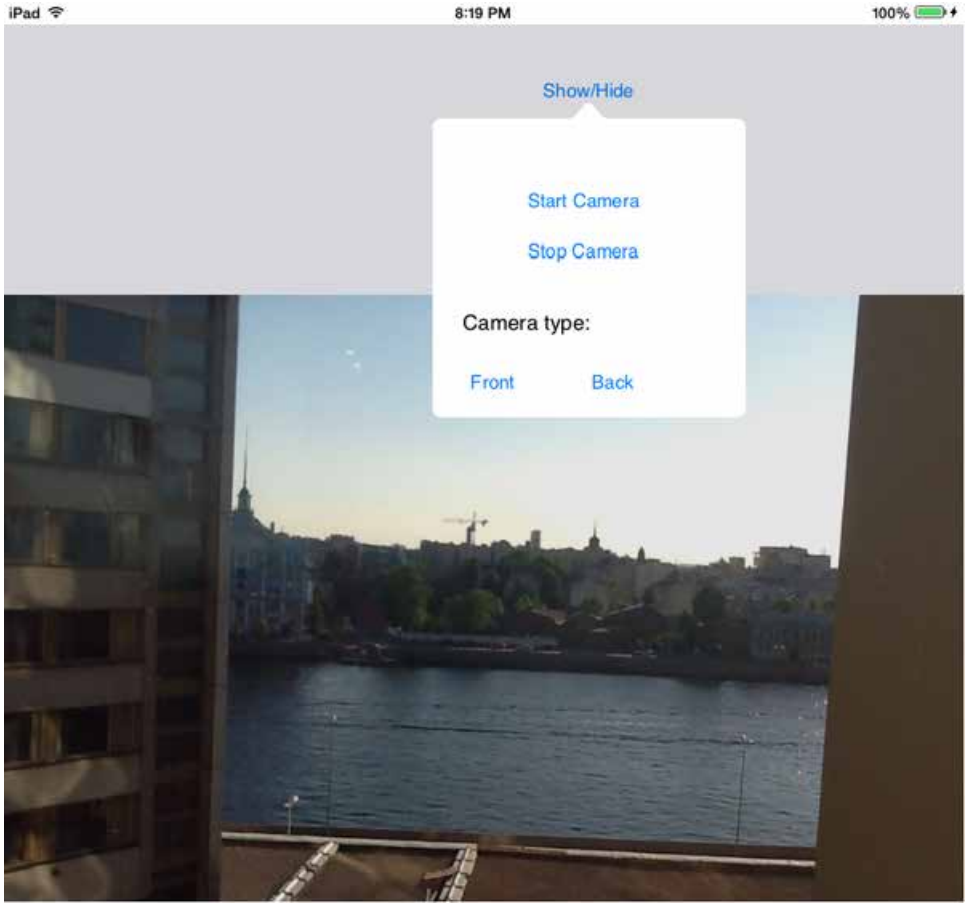
1. In **Structure View**, right-click a TMultiView node and select the **Show** or **Hide** commands.
2. In **Structure View**, double-click a TMultiView node.

### Drawer Mode

If you set the TMultiView.Mode property to `Drawer` (using TDrawerAppearance.Mode=`OverlapDetailView`), the master pane is initially hidden. To display the master pane, the user swipes right from the left edge of the screen, as shown in the following animated image:


## Popover Mode

You can also set the TMultiView.Mode property to `Popover` to make the master pane a pop-up menu that is displayed next to the Master button specified in the TMultiView.MasterButton property.

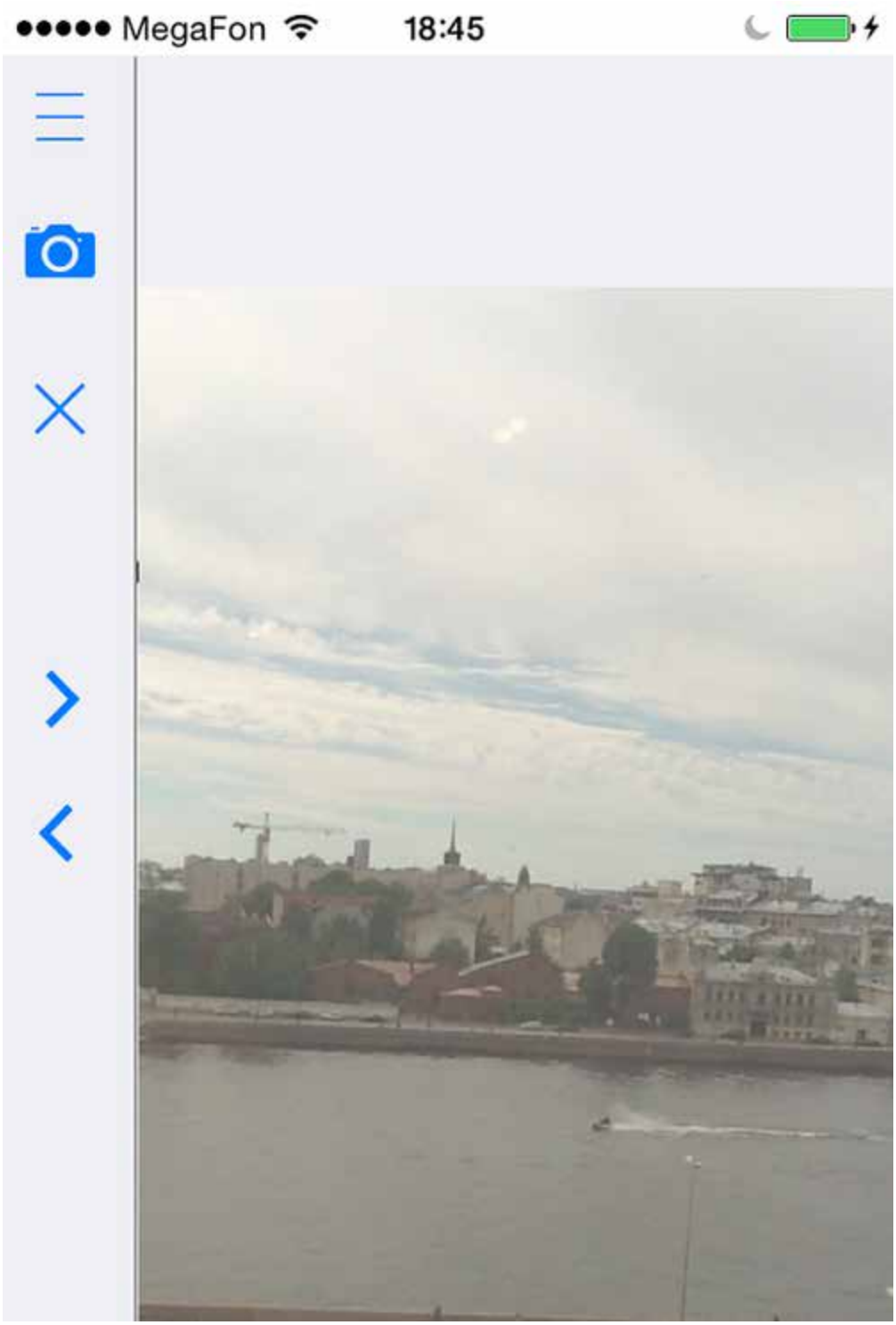



**Important:** In the Popover mode, you must set the TMultiView.MasterButton property. This property refers to a UI element that displays or hides the master panel. In the above screen, the Master button is the **Show/Hide** button.

## Navigation Pane Mode

If you set the TMultiView.Mode property to `NavigationPane`, the master pane is initially displayed as a minimized docked panel. You can control the initial width of this panel with the CollapsedWidth property (by default, **CollapsedWidth**=50).

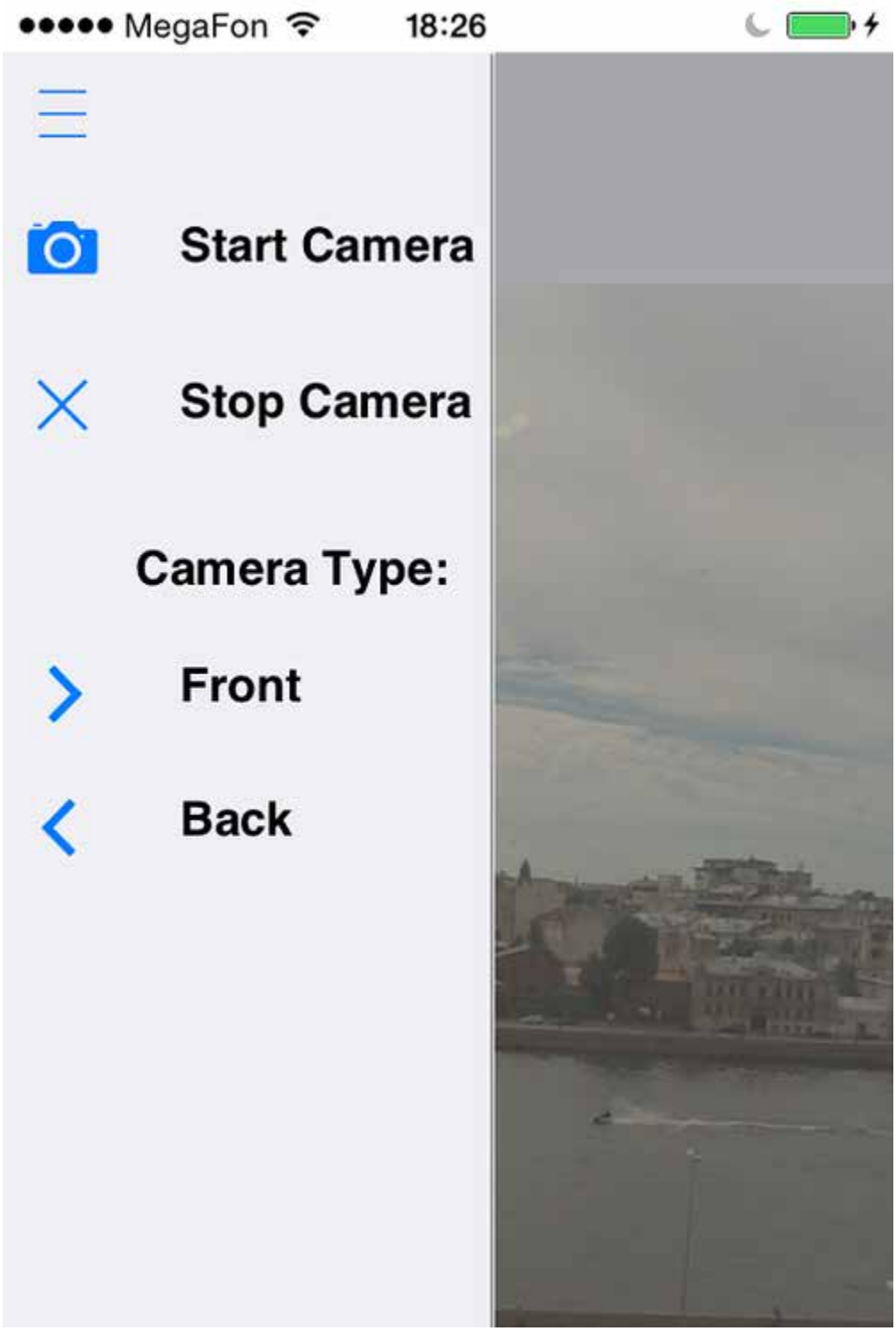
### Initially minimized Navigation Pane




**Tip** To clip any portion of child elements moved outside the minimized Navigation Pane, set the TMultiview.ClipChildren property to `True`.

To expand the navigation panel to display all child controls, tap the Master button specified in the TMultiView.MasterButton property.

Expanded Navigation Pane




**Important:** In the Navigation Pane mode, you must set the TMultiView.MasterButton property. This property refers to a UI element that

collapses or expands the master panel. In the above screen, the Master button is the 

### Platform Dependent Behavior Mode

You can let the application automatically select the master pane presentation mode, if the TMultiView.Mode property is set to `PlatformBehaviour`. For this setting, the application behavior depends on the device type and orientation, as described in the following table:

| Device Type | Device Orientation | Master Pane Presentation |
|---|---|---|
| Phone | Landscape, Portrait | Drawer (push/overlap) |
| Tablet | Landscape | Docked panel |
| Tablet | Portrait | Drawer (push/overlap) |

### Custom Mode

In Custom mode, you can customize the master pane presentation to conform to your tasks. To customize the master pane presentation, perform the following basic steps:

1. Declare your own class, such as `MyPresentationClass` that descends from TMultiViewPresentation or from other classes that were declared in the FMX.MultiView.Presentations unit.
2. In the `MyPresentationClass`, optionally, override the following virtual methods defined in the base class:
   - DoOpen
   - DoClose
   - GetDisplayName
   - DoInstall
   - DoUninstall

   These methods define the master pane behavior.
3. In the Form Designer, select the **TMultiView** component, and then in the Object Inspector, set its property **Mode** to `Custom`.
4. Implement the **onFormCreate** event handler as follows:
   - For **Delphi**:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  MultiView1.CustomPresentationClass := MyPresentationClass;
```

```
end;
```

- For **C++**:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MultiView1->CustomPresentationClass = __classid(MyPresentationClass);
}
```

This topic helps you develop a simple application that illustrates the use of the TMultiView component.

# Designing the User Interface

1. Create a blank Multi-Device Application, by selecting:
   - For **Delphi**: **File > New > Multi-Device Application - Delphi > Blank Application**
   - For **C++**: **File > New > Multi-Device Application - C++Builder > Blank Application**
2. Select the **TMultiView** component in the Tool Palette, and drop it on the Form Designer.
3. Drop other controls, such as buttons, edit boxes or any other controls you need onto the MultiView container.
4. In the Tool Palette, select a component you want to use as a detail pane (such as TPanel), and drop any controls onto this panel.
5. In the Object Inspector, specify the appropriate properties of the **TMultiView** component.

To clarify this procedure, the following sections consider a particular example: an application that controls the mobile device camera.

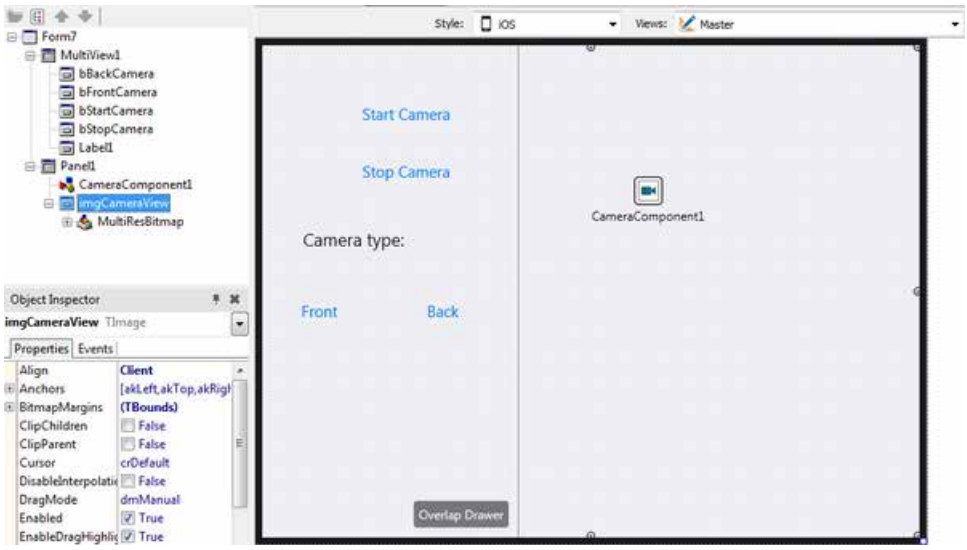
## Designing the Master Pane

1. Select the **TMultiView** component in the Tool Palette, and drop it on the Form Designer.
2. Drop two TButton components into the **TMultiview** container, and then in the Object Inspector specify the following properties of those buttons:
   - Set the `Name` property to **bStartCamera** and **bStopCamera**, respectively.
   - Set the `Text` property to **Start Camera** and **Stop Camera**, respectively.
3. Drop a TLabel component into the **TMultiview** container, and then in the Object Inspector, set its `Text` property to **Camera type:**.

4. Drop two TButton components into the **TMultiview** container, and then in the Object Inspector specify the following properties of those buttons:
    - Set the `Name` property to **bFrontCamera** and **bBackCamera**, respectively.
    - Set the `Text` property to **Front** and **Back**, respectively.

## Designing the Detail Pane

1. Select the TPanel component in the Tool Palette, and drop it on the Form Designer.
2. In the Object Inspector, set the **TPanel.Align** property to `Client`.
3. Drop the TCameraComponent into the **TPanel** container.
4. Drop the TImage into the **TPanel** container, and set the following properties:
    - **Name** = `imgCameraView`
    - **Align** = `Client`




> **Tip:** Put all elements of the details pane into a unique container (a TPanel component in our example). This container should be specified in the TMultiView.TargetControl property.

## Implementing the Camera Buttons Functionality

To complete the application development, you should implement event handlers for the application buttons and the `GetImage` private method that gets an image from the device camera.

### To implement the OnClick event handlers

1. On the Form Designer, double-click the **Start Camera** button, and insert the following code:
    - For **Delphi**:

```
procedure TForm1.bStartCameraClick(Sender: TObject);
begin
   CameraComponent1.Active := true;
end;
```

- For **C++**:

```
void __fastcall TForm1::bStartCameraClick(TObject *Sender)
{
CameraComponent1->Active = true;
}
```

2. Double-click the **Stop Camera** button, and insert the following code:

- For **Delphi**:

```
procedure TForm1.bStopCameraClick(Sender: TObject);
begin
   CameraComponent1.Active := false;
end;
```

- For **C++**:

```
void __fastcall TForm1::bStopCameraClick(TObject *Sender)
{
CameraComponent1->Active = false;
}
```

3. Double-click the **Front** button, and insert the following code:

- For **Delphi**:

```
procedure TForm1.bFrontCameraClick(Sender: TObject);
begin
  CameraComponent1.Active := False;
  CameraComponent1.Kind := FMX.Media.TCameraKind.FrontCamera;
  CameraComponent1.Active := True;
end;
```

- For **C++**:

```
void __fastcall TForm1::bFrontCameraClick(TObject *Sender) {
        // select Front Camera
        CameraComponent1->Active = false;
        CameraComponent1->Kind = TCameraKind::FrontCamera;
        CameraComponent1->Active = true;
```

```
}
```

4. Double-click the **Back** button, and insert the following code:

    - For **Delphi**:

```
procedure TForm1.bBackCameraClick(Sender: TObject);
begin
  CameraComponent1.Active := False;
  CameraComponent1.Kind := FMX.Media.TCameraKind.BackCamera;
  CameraComponent1.Active := True;
end;
```

    - For **C++**:

```
void __fastcall TForm1::bBackCameraClick(TObject *Sender) {
        // select Back Camera
        CameraComponent1->Active = false;
        CameraComponent1->Kind = TCameraKind::BackCamera;
        CameraComponent1->Active = true;
}
```

### To implement the onSampleBufferReady event handler

- In the Form Designer, double-click the **CameraComponent1** and implement the following code:

    - For **Delphi**:

```
procedure TForm1.CameraComponent1SampleBufferReady(Sender: TObject;
  const ATime: Int64);
begin
       TThread.Synchronize(TThread.CurrentThread, GetImage);
end;
```

    - For **C++**:

```
void __fastcall TForm1::CameraComponent1SampleBufferReady(TObject *Sender,
                  const __int64 ATime)
{
        GetImage();
}
```

For the **TForm1** class, you should implement the private method **GetImage**. The onSampleBufferReady event handler calls this method to get the image from the device camera.

Do the following:

1. In the private section of the **TForm1** class, declare the **GetImage** method:
    - For **Delphi**:

```
private
   { Private declarations }
   procedure GetImage;
```

    - For **C++**:
    - In the header file (.h file), add the following code:

```
private:  // User declarations
        void __fastcall GetImage();
```

2. Implement the **GetImage** method as follows:
    - For **Delphi**:

```
procedure TForm1.GetImage;
begin
  CameraComponent1.SampleBufferToBitmap(imgCameraView.Bitmap, True);
end;
```

    - For **C++**:

```
void __fastcall TForm1::GetImage()
{
        CameraComponent1->SampleBufferToBitmap(imgCameraView->Bitmap, true);
}
```

## Setting the TMultiView Component Properties

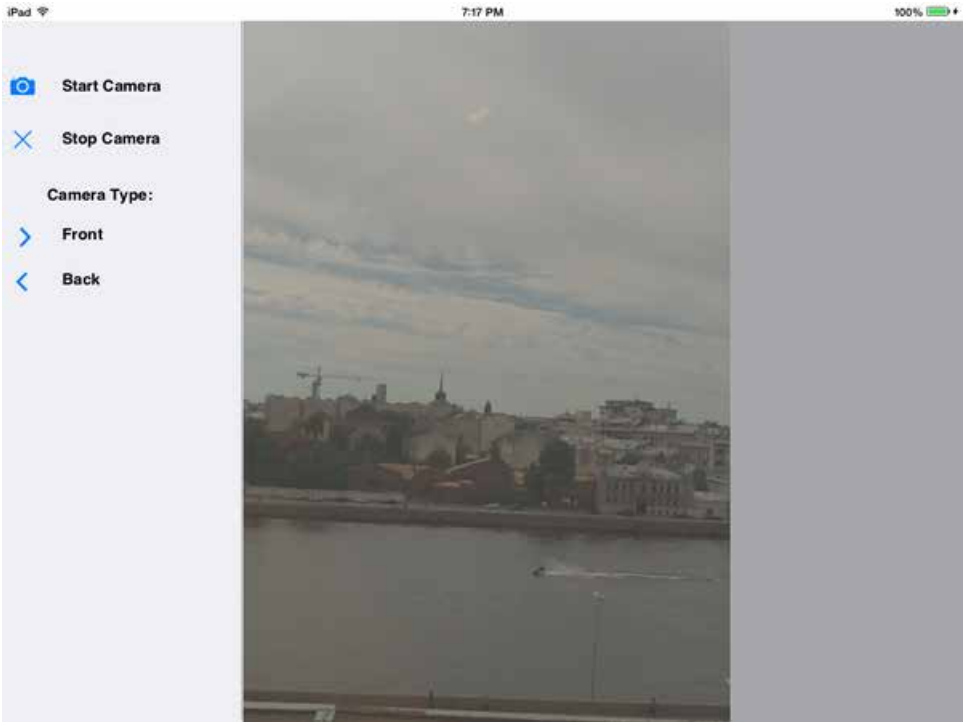
In the Form Designer, select the **TMultiView** component, and then in the Object Inspector, set the following properties:

- TargetControl = `Panel1`
- Mode = `Drawer`
- Visible = `True`
- Expand the DrawerOptions node, and set the **Mode** property to `OverlapDetailView`.
- Expand the ShadowOptions node, and set the **Color** property to `Beige`. (This property defines the color of the master panel shadow. You can use any available color.)

# Running the Example Application

To run this application, do the following:

1. In the Project Manager, select the target platform (supported platforms: Android or iOS).
2. Press `Shift+Ctrl+F9` to run the application without debugging.
3. To open the master panel, swipe right from the left edge of the device screen.
4. To activate the device camera, on the master panel, click **Start Camera**.
5. Optionally, you can select the front or back camera (if available) by using the **Front** or **Back** button, respectively.




To close the master panel, slide it left.

# Mobile Product Samples that Use TMultiView

Go to the Mobile Samples folder in `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples`.

- MultiView Demo sample
- Location Demo sample
- Music Player sample
- REST Surf Spot Finder sample (Delphi)

- REST Surf Spot Finder sample (C++)

## See Also

- FMX.MultiView.TMultiView
- FMX.MultiView.TMultiViewPresentation
- FireMonkey Native iOS Controls

# Mobile Tutorial: Using the Web Browser Component (iOS and Android)

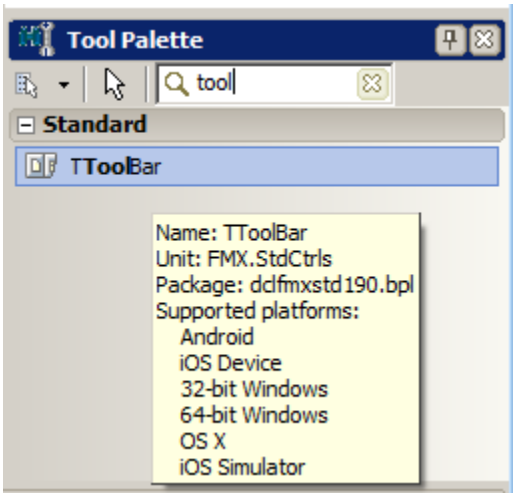
FireMonkey wraps the Web Browser component as the **TWebBrowser** component. You can use TWebBrowser in desktop apps as well as mobile apps, but this topic describes how to create a simple FireMonkey Web Browser application for iOS and Android platforms.


# Design the User Interface

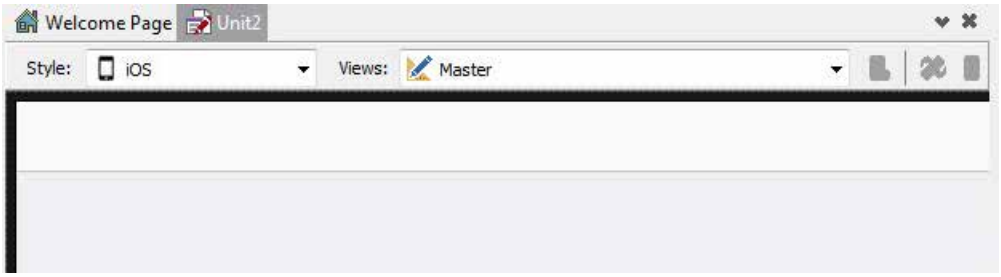
1. Select either:
   - § File > New > Multi-Device Application - Delphi > Blank Application
   - § File > New > Multi-Device Application - C++Builder > Blank Application

2. Select the **TToolBar** component in the Tool Palette, and drop it on the Form Designer.

To find **TToolBar**, enter a few characters (such as "tool") in the **Search** box of the Tool Palette:




3. After you drop the component, you can see the **TToolBar** component at the top of the Form Designer. Here is a screenshot after setting the iOS style in the Form Designer:




4. Select the **TButton** component in the Tool Palette and drop it on the **TToolBar**.
5. Select the **TButton** component on the Form Designer, and then in the Object Inspector, set the **StyleLookup** property to **priortoolbutton**.
    - The **priortoolbutton** StyleLookup value for **TButton** adds a Back button label.

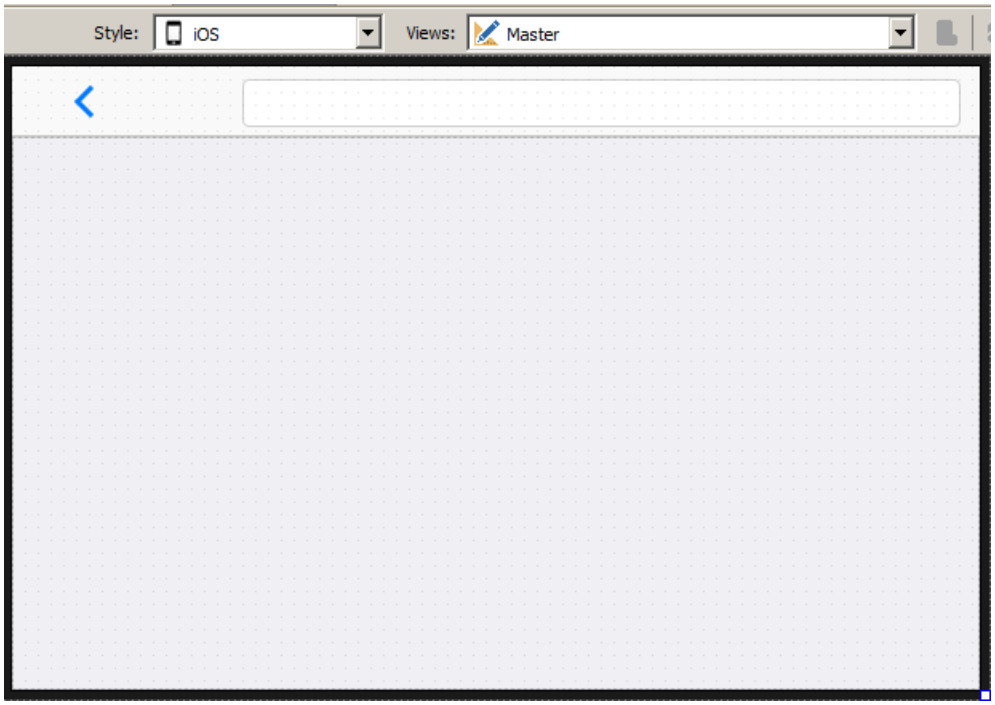
      On iOS devices, the label is similar to the following image: 
    - For more detail about selecting a style in multi-device applications, see Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android).
6. Select the **TEdit** component in the Tool Palette and drop it on the **TToolBar**. Make sure that the size of the Edit control is wide enough to fill the area of the **TToolBar**:

7. Select the Edit box on the Form Designer, and then in the Object Inspector, set the **ReturnKeyType** property to `Done`, the **KeyboardType** property to `URL`, and the **KillFocusByReturn** property to `True`.

For more information about selecting the most appropriate Virtual Keyboard type in mobile platforms, see Selecting the Proper Virtual Keyboard for the Web Browser Application.

8. Select the **TWebBrowser** component in the Tool Palette and drop it on the form.
9. Select the Web Browser component on the Form Designer, go to the Object Inspector and select **Client** for the **Align** property.

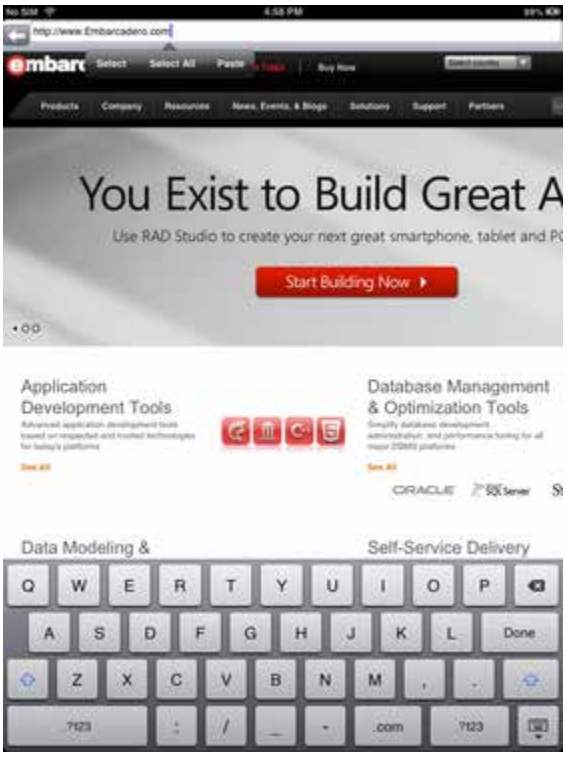
After you complete these steps, the form should be similar to the following picture:




# Write an Event Handler to Open a Web Page when the User Changes the URL in the Edit Control

Unlike desktop platforms, mobile platforms use the Virtual Keyboard to enter text as in the following images. The user can complete the action by clicking "Done".

iOS




iPad

Android


Android (LG - E612

FireMonkey provides many types of event handlers to cover most actions taken by users. After the "Done" button is selected, the FireMonkey framework sends an OnChange event to the TEdit control. On the other hand, there is no specific event for the "Back" button. In this section, you implement event handlers to support both scenarios.

## Implement a Common Method to Open a Web Page

Before implementing event handlers, first implement a common method to open a Web page based on the Text property of the Edit control.

1. In the private section of the TForm1 class, declare the **OpenURL** method:

   **Delphi**:

```
private
  { Private declarations }
  procedure OpenURL;
```

**C++Builder:**

```
private:        // User declarations
  void __fastcall openURL();
```

2. Implement the **openURL** method as follows:

**Delphi:**

```
procedure TForm1.OpenURL;
begin
  WebBrowser1.Navigate(Edit1.Text);
end;
```

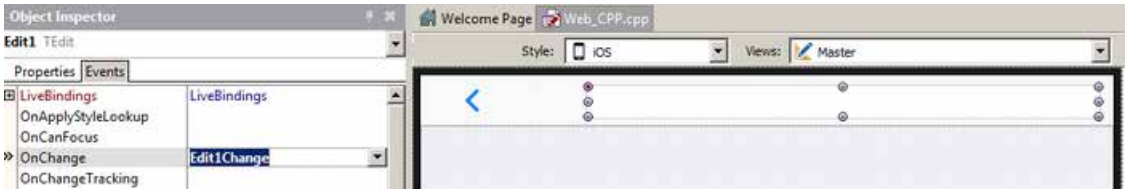
**C++Builder:**

```
void __fastcall TForm1::openURL()
 {
     WebBrowser1->Navigate(Edit1->Text);
 }
```

## Implement an Event Handler for the OnChange Event

1. Create the event handler by selecting the **Edit** component (in the Form Designer), and then double-clicking the white space next to the **OnChange** event (in the Object Inspector's **Events** tab).

The Object Inspector creates a new event handler called **Edit1Change**:




2. Complete the event handler by adding the following code:

**Delphi:**

```
procedure TForm1.Edit1Change(Sender: TObject):
begin
  OpenURL;
end;
```

**C++Builder:**

```
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
    openURL();
}
```

### Implement an Event Handler for the Back Button

To implement the **Back** button for your Web Browser, you can simply call the GoBack method on the Web Browser component:

**Delphi:**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  WebBrowser1.GoBack;
end;
```

**C++Builder:**

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   WebBrowser1->GoBack();
}
```

The basic behavior is now implemented for this Web Browser application. Try running your application on your Android device, the iOS Simulator, or your iOS device.

## Selecting the Proper Virtual Keyboard for the Web Browser Application

After you run your first Web Browser application, you might realize that the Virtual Keyboard is not optimized.

iOS provides several virtual keyboards as follows:

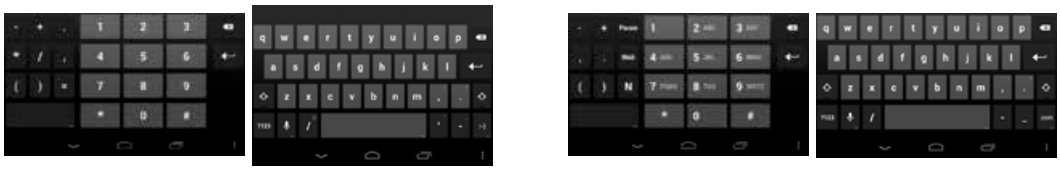
Alphabet: Default: EmailAddress: NamePhonePad:

NumberPad: NumbersAndPunctuation: PhonePad: URL:

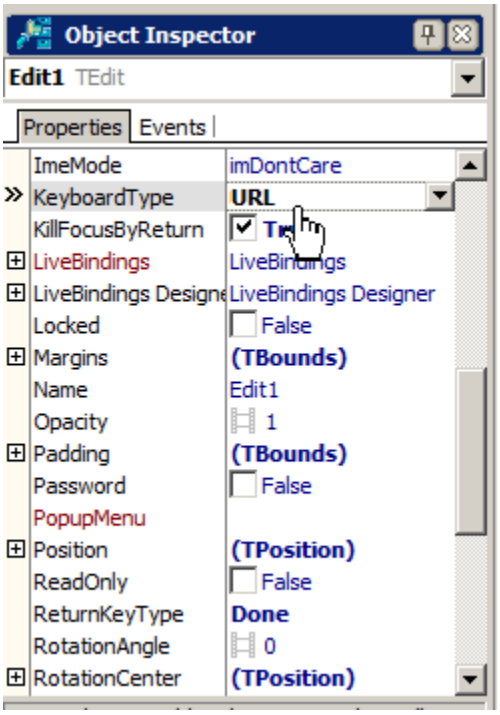
Android provides several virtual keyboards as follows:

Alphabet: Default: EmailAddress: NamePhonePad:

NumberPad: NumbersAndPunctuation: PhonePad: URL:



The most appropriate Virtual Keyboard type for Web Browser components is **URL**. As we have already discussed in Design the User Interface, the following steps set the **URL** as the Virtual Keyboard type for the Web Browser component in this example. Select the Edit box on the Form Designer, and then in the Object Inspector, set the **KeyboardType** property to URL.




# WebBrowser Mobile Code Snippet

The **WebBrowser** project in Mobile Code Snippets demonstrates the functionality described in this tutorial.

You can find the **WebBrowser** project at:

- Start | Programs | Embarcadero RAD Studio 10.2 Tokyo | Samples and navigate to **\Object Pascal\Mobile Snippets\WebBrowser**
- **Subversion Repository:**http://sourceforge.net/p/radstudiodemos/code/HEAD/tree/branches/RADStudio_Tokyo/Object%20Pascal/Mobile%20Snippets/WebBrowser

# See Also

- Mobile Tutorial: Using Combo Box Components to Pick Items from a List (iOS and Android)
- Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)
- FMX.KeyboardTypes Sample
- TWebBrowser
- TToolBar
- TButton
- TEdit

- KeyboardType
- StyleLookup

# Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)

Tabs are defined by **FMX.TabControl.TTabControl**, which is a container that can hold several tab pages. Each tab page can contain any control as a UI element. You can hide the tab for these pages, and change pages without showing tabs.


For each tab, you can specify:

- A text label — for both iOS and Android
- Predefined icons — for iOS only
- Custom icons — for both iOS and Android

## Using the Native Style for Tabs on iOS and Android

This tutorial shows tabs with the same style on both iOS and Android, but this practice is not recommended.
We recommend that you observe the native style of each platform, as follows:

- **On Android:**
  - Tabs are commonly placed at the top of the screen (so you should set TTabPosition either to **Top** or to **PlatformDefault**).
  - Tabs traditionally display only text. However, FireMonkey allows you to specify custom icons to be displayed on tabs (see Using Custom Multi-Resolution Icons for Your Tabs).
- **On iOS:**
  - Tabs are typically shown at the bottom of the screen (so you should set TTabPosition either to **Bottom** or to **PlatformDefault**).

- Tab items always display both text and an icon, which can be set via the StyleLookup property for each tab.

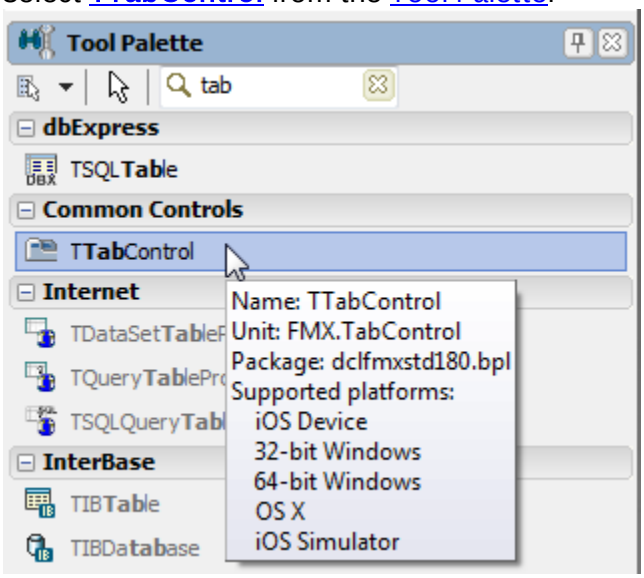
**Note:** You can use the **PlatformDefault** value of the TTabPosition enumeration to set the tab position according to the default behavior of the target platform. When **PlatformDefault** is set for TTabPosition:

- In iOS apps, tabs are aligned at the lower edge of the TTabControl.
- In Android apps, tabs are aligned at the top edge of the TTabControl.

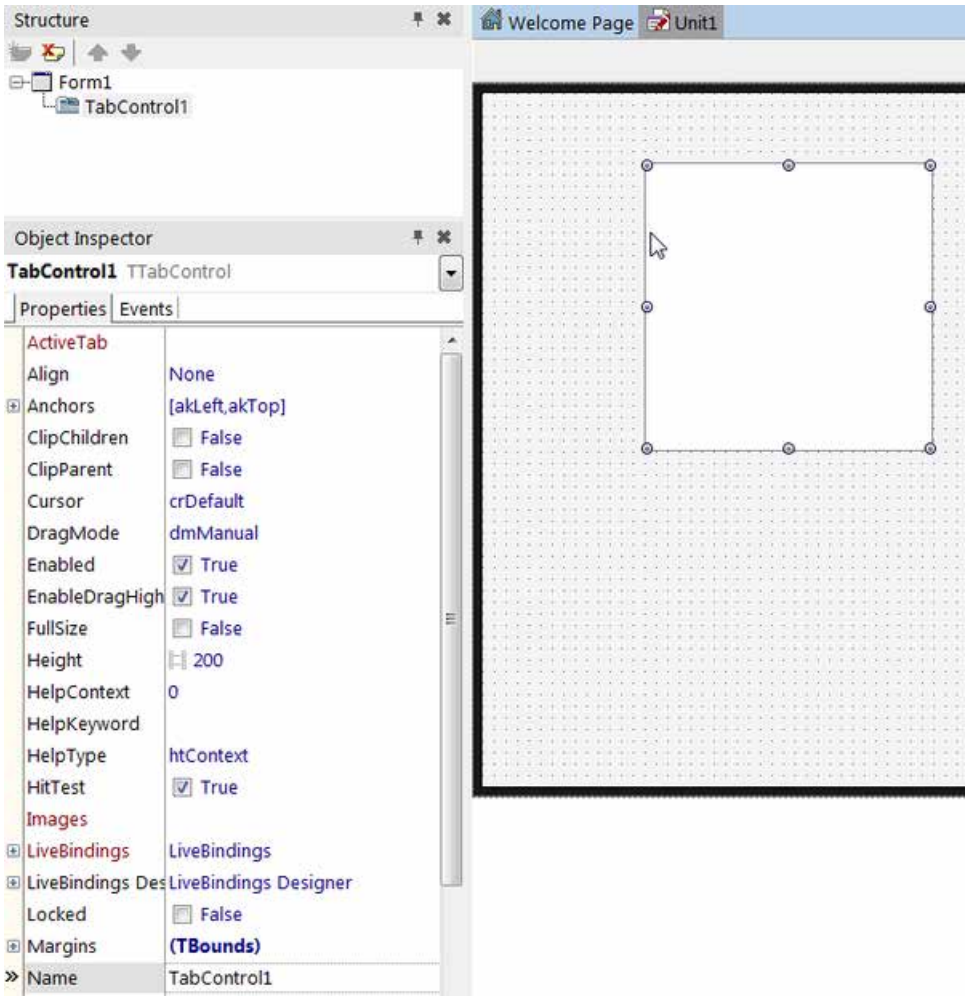
# Designing Tab Pages Using the Form Designer

To create tab pages in your application, use the **TTabControl** component with the following steps:

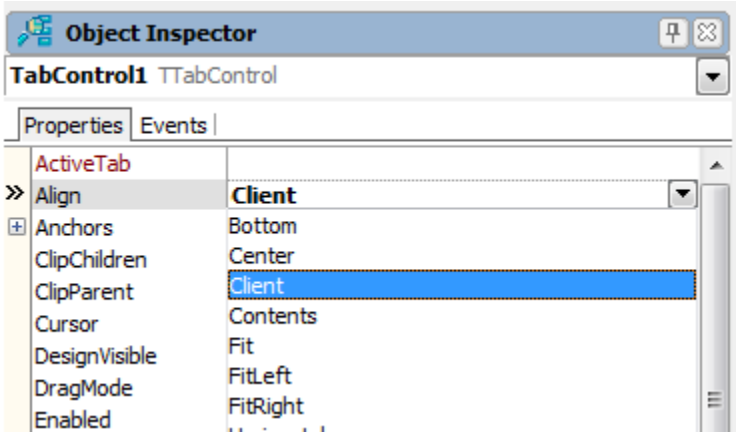
1. Select:
   - For Delphi: File > New > Multi-Device Application - Delphi > Blank Application
   - For C++: File > New > Multi-Device Application - C++Builder > Blank Application
2. Select **TTabControl** from the Tool Palette:



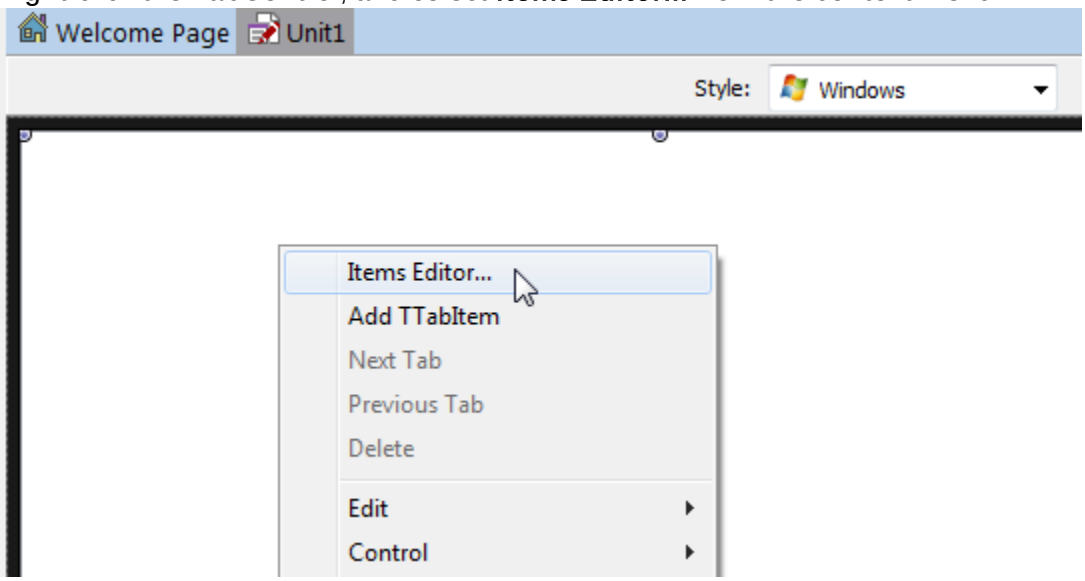

3. After you drop the **TTabControl**, an empty **TabControl** is shown on the Form Designer (you might need to manually adjust the position of the TabControl):



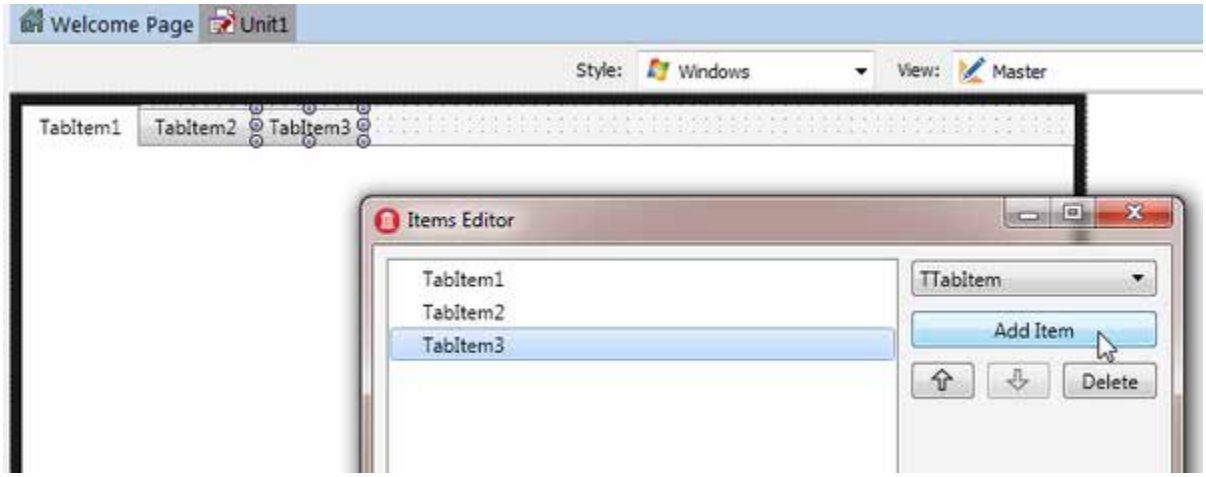

4. Typically, applications that use TabControl use the full screen to show pages. To do this, you need to change the default alignment of TabControl. In the Object Inspector, change the **Align** property of TabControl to **Client**:



5. Right-click the TabControl, and select **Items Editor...** from the context menu:

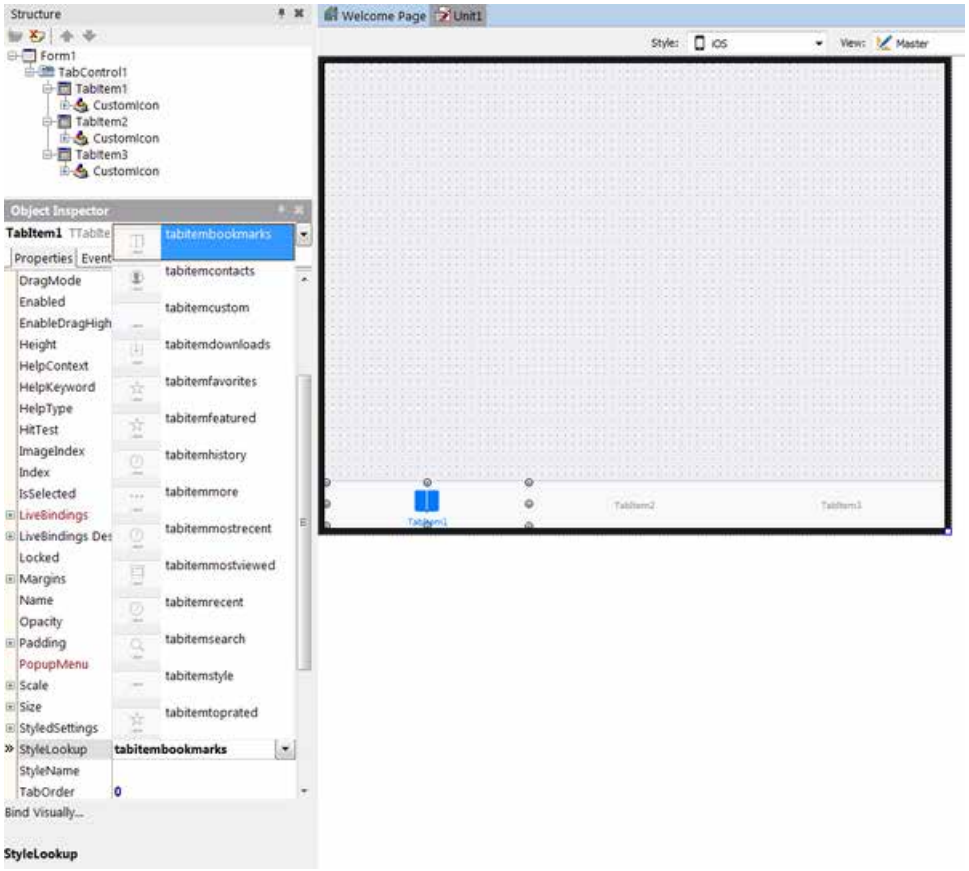



6. Click **Add Item** three times, so that now you have three instances of TabItem here. Close the dialog box.

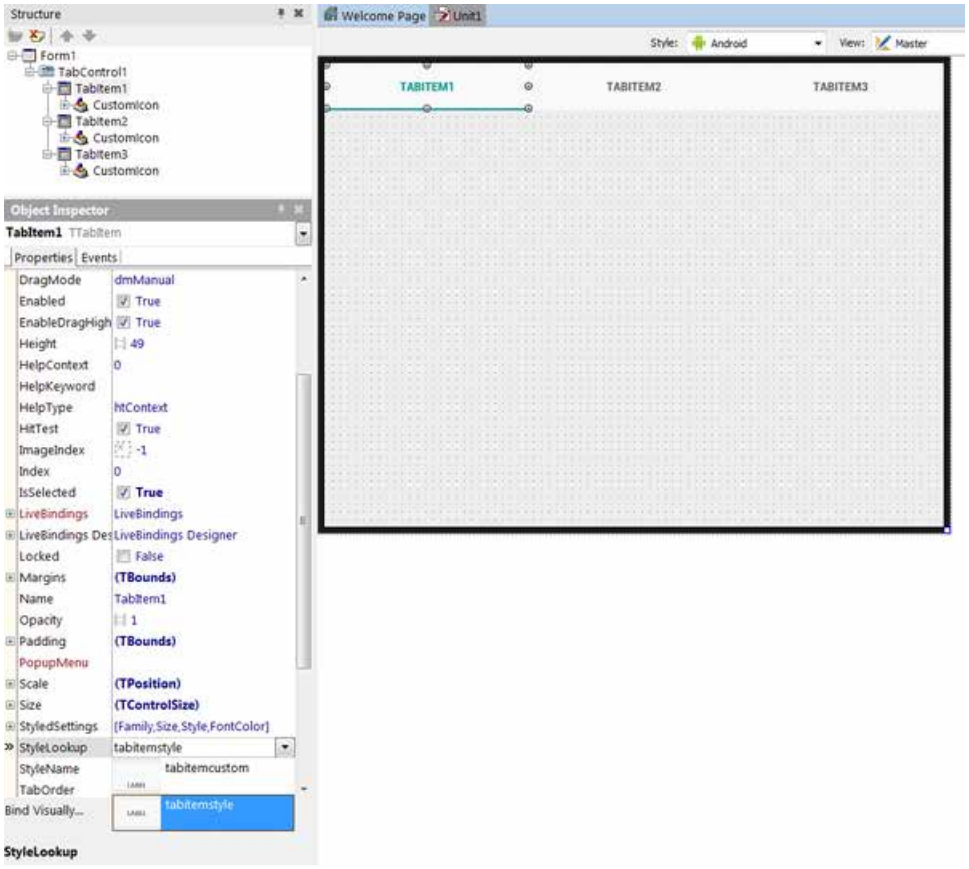



7. On the Form Designer, select the first TabItem and change its **StyleLookup** property:

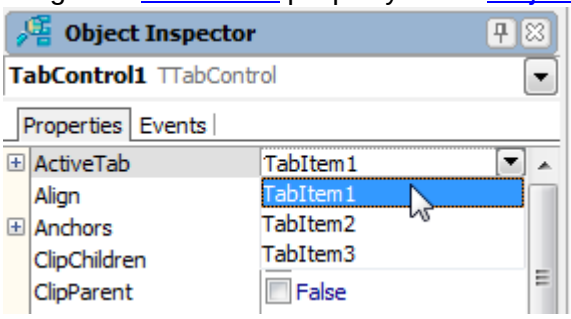
iOS


Structure
Form1
TabControl1
TabItem1
CustomIcon
TabItem2
CustomIcon
TabItem3
CustomIcon
Object Inspector
TabItem1 TTabIte
Properties
DragMode
Enabled
EnableDragHigh
Height
HelpContext
HelpKeyword
HelpType
HitTest
ImageIndex
Index
IsSelected
LiveBindings
LiveBindings Des
Locked
Margins
Name
Opacity
Padding
PopupMenu
Scale
Size
StyledSettings
StyleLookup
StyleName
TabOrder
Bind Visually...
StyleLookup
tabitembookmarks
tabitemcontacts
tabitemcustom
tabitemdownloads
tabitemfavorites
tabitemfeatured
tabitemhistory
tabitemmore
tabitemmostrecent
tabitemmostviewed
tabitemrecent
tabitemsearch
tabitemstyle
tabitemtoprated
tabitembookmarks
0
Welcome Page
Unit1
Style:
iOS
View:
Master

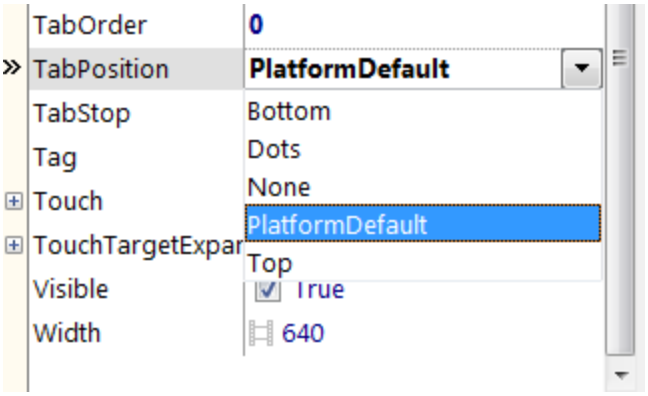
**Android**




9.

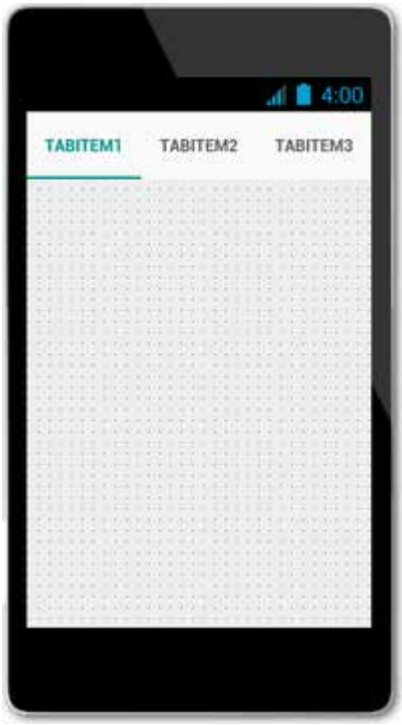
10. You can place any component on each page.
    To move to a different page, just click the tab you want on the Form Designer, or change the **ActiveTab** property in the Object Inspector:
    
    
    :

11. To change the location of tabs, select the **TabPosition** property for the TabControl component, and set it to one of the following values in the Object Inspector:

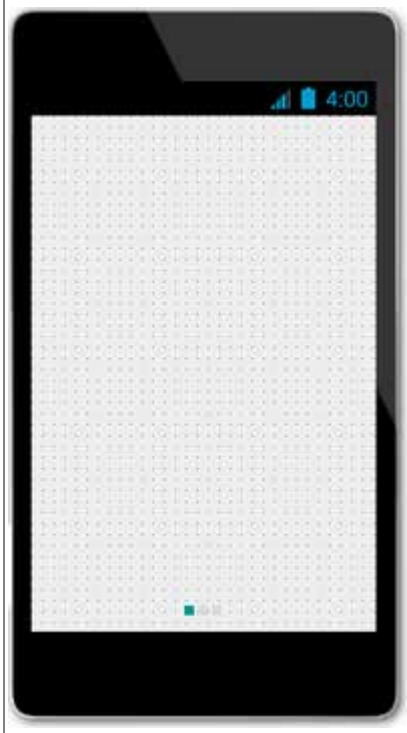


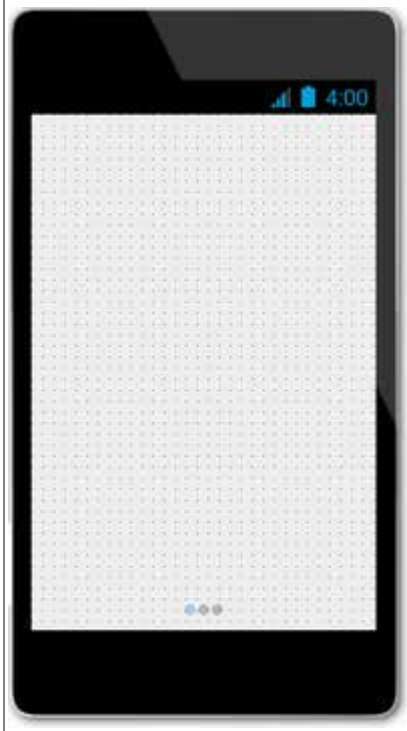

## Comparing the Tab Settings on iOS and Android

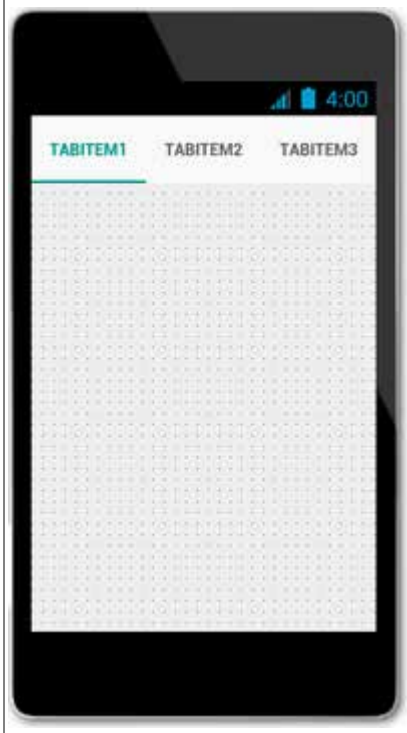
The following figures show both apps with the same **TabPosition** settings (**Top**, **Bottom**, **Dots**, and **None**) on iOS and Android.
However, you should set the appropriate different tab settings for each mobile platform, as indicated in #Using the Native Style for Tabs on iOS and Android.

| Top | |
|---|---|
| iOS | Android |
|  |  |
| Tabs are displayed at the Top. | |

| Bottom | |
|---|---|
| iOS | Android |
|  |  |
| Tabs are displayed at the Bottom. | |

| Dots | |
|---|---|
| iOS | Android |
|   |   |
| No Tabs are displayed.<br>Instead, three Dots ([...]) are displayed to indicate additional pages. | |

| None | |
|---|---|
| iOS | Android |
|   |   |
| No Tabs or Dots are displayed at run time, although you can see them at design time. Page can be changed only through code or action. | |

| Platform Default | |
|---|---|
| iOS | Android |
|  |  |
| Tabs are displayed with their platform default settings. | |



# Using Custom Multi-Resolution Icons for Your Tabs

You can use custom multi-resolution icons as well as custom text on tabs in your application. This tutorial shows you how to construct the following three tabs that have custom icons and text:


**Notes:**

- In **Android** apps, predefined icons are not supported, so you must use custom icons.
- In **iOS** apps, you can use either predefined icons or custom icons.

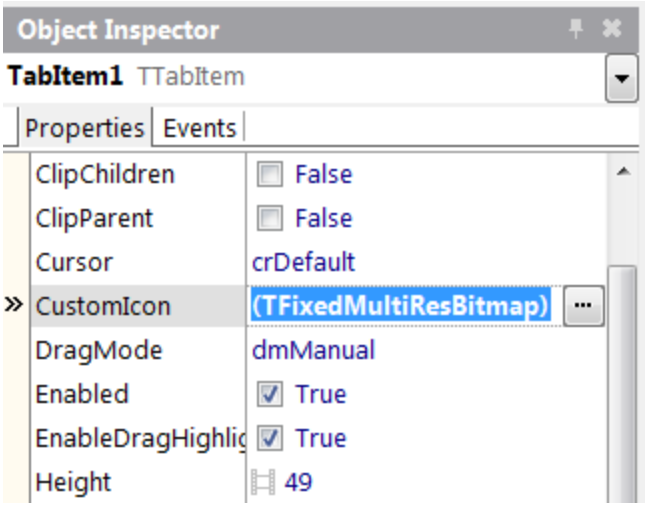
- To use custom icons on either iOS or Android, select the appropriate iOS or Android **Style** in the Form Designer, set the **StyleLookup** property of TTabItem to **tabitemcustom**, specify your custom icon as described in this section, and then build your app.
- For iOS, you can use our predefined icons by setting the **StyleLookup** property of TTabItem to the icon of your choice, such as  (**tabitemsearch**).
- The custom glyphs used in this section are available in a zip file that is delivered in your `C:\Program Files (x86)\Embarcadero\Studio\19.0\Images\GlyFX` directory. The three PNGs used here are located in the `Icons\Aero\PNG\32x32` directory:
  - **users_32** (People)
  - **unlock_32** (Security)
  - **tree_32** (Organization)

Unzip the **glyFX.zip** file before you use the MultiResBitmap Editor if you want to use these images or any others available in the GlyFX collection.

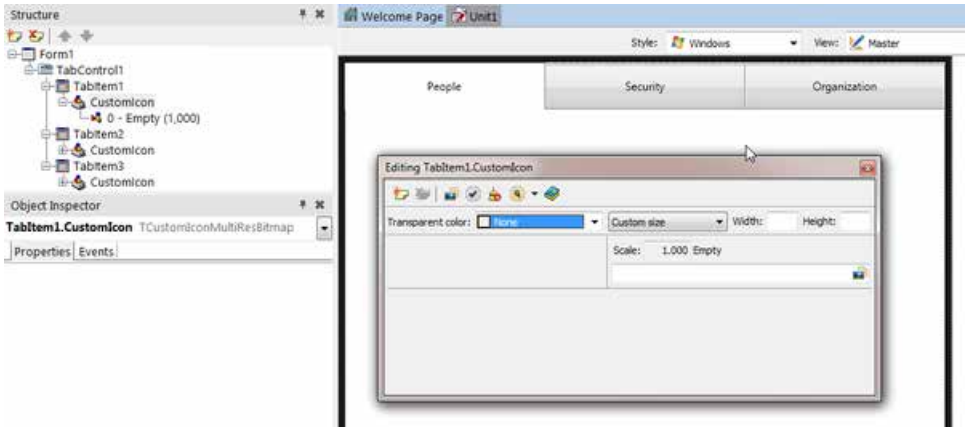
## Displaying Multi-Resolution Custom Icons on Tabs

1. In the Object Inspector, select **TabItem1**, and then change the tab caption, which is specified in the Text property to **People**; change the **Text** property of **TabItem2** to **Security**, and **TabItem3** to **Organization**.

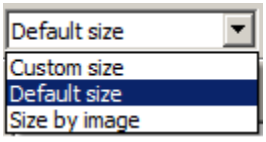

2. Select a tab, and click the ellipsis button [...] on the **CustomIcon** property of TTabItem in the Object Inspector:



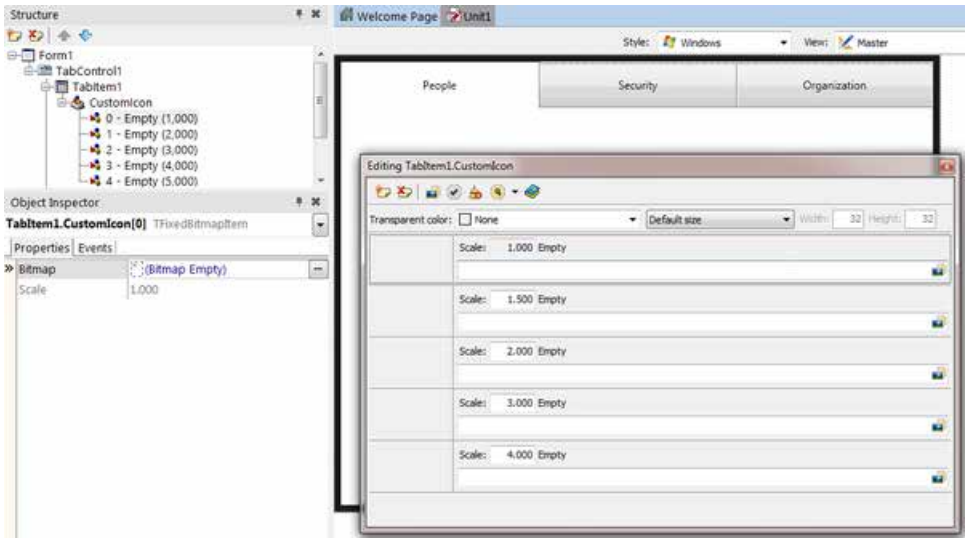

3. The MultiResBitmap Editor opens:




4. Ensure that you are in the Master view, and in the MultiResBitmap Editor, click the array next to **Custom size**, and then choose **Default size**.

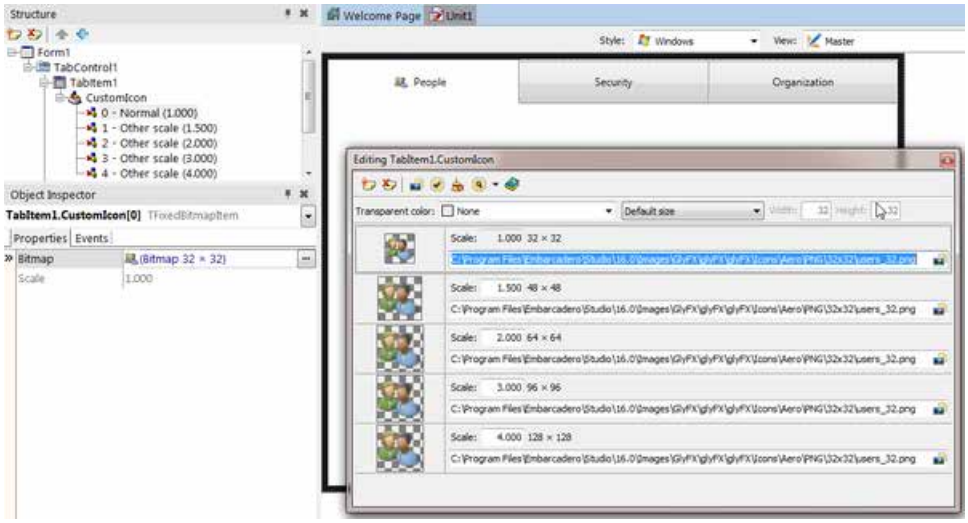



5. Repeat the following steps to add any additional scales that you want to support:
    1. In the MultiResBitmap Editor, click (**Add new item**).
    2. Enter the additional Scale you want to support, such as 1.5, 2, or 3.
        - When you have added all the scales you want, the editor looks like this:




6. Click the **Fill All from File** button , navigate to an image file you want to use, and then click **Open**.

The selected image now appears appropriately scaled in each of the Scale entries on the MultiResBitmap Editor:



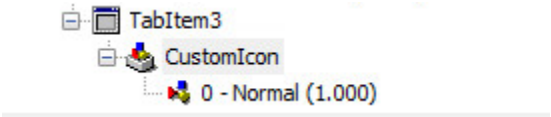

7. Close the MultiResBitmap Editor.
8. Repeat Steps 2 to 7 for each of the remaining tabitems, and assign each tabitem a custom icon image.

After you define a custom icon, the FireMonkey framework generates a **Selected Image** and **Non-Selected (dimmed) Image** based on the given .png file. This transformation is done using the Alpha-Channel of the bitmap data. For example:

Original Image Selected Image Non-Selected Image

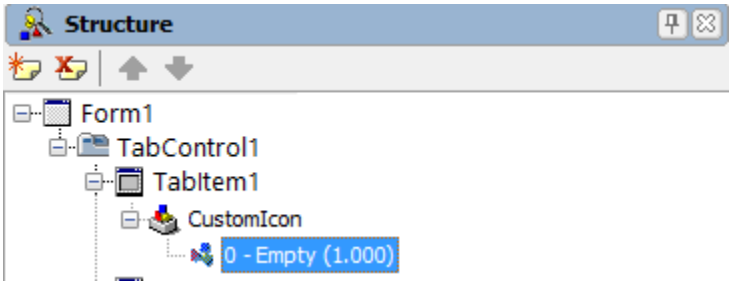

## Using a Single-Resolution Bitmap for a Custom Icon

You can also use only a single-resolution bitmap by using the **Bitmap Editor**. A single-resolution bitmap displays only one scale in the Structure View:

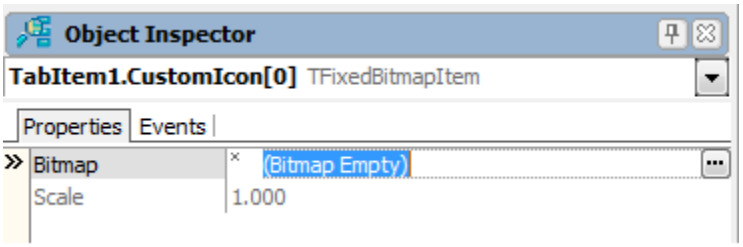



To specify a single-resolution bitmap for a custom icon, perform the first step of the procedure above and then proceed as follows:

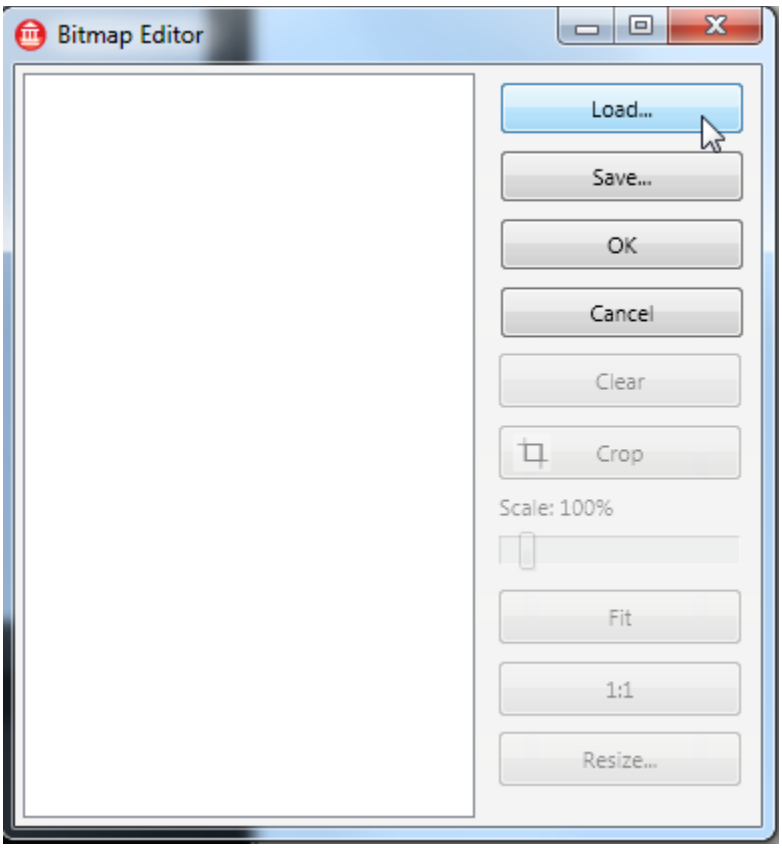
1. In the Structure View, select **Empty** under CustomIcon:




2. Now, in the Object Inspector, click the ellipsis button [...] in the **Bitmap** field (of the TabItem1.CustomIcon[0]). This opens the **Bitmap Editor**:

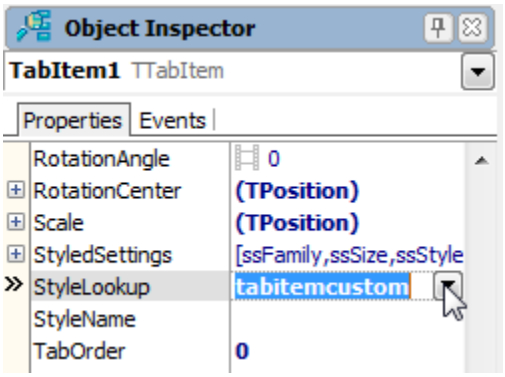



3. In the **Bitmap Editor**, click the **Load...** button, and select a PNG file.
   The recommended size is 30x30 pixels for normal resolution, and 60x60 pixels for high resolution:




4. Click **OK** to close the **Bitmap Editor**.

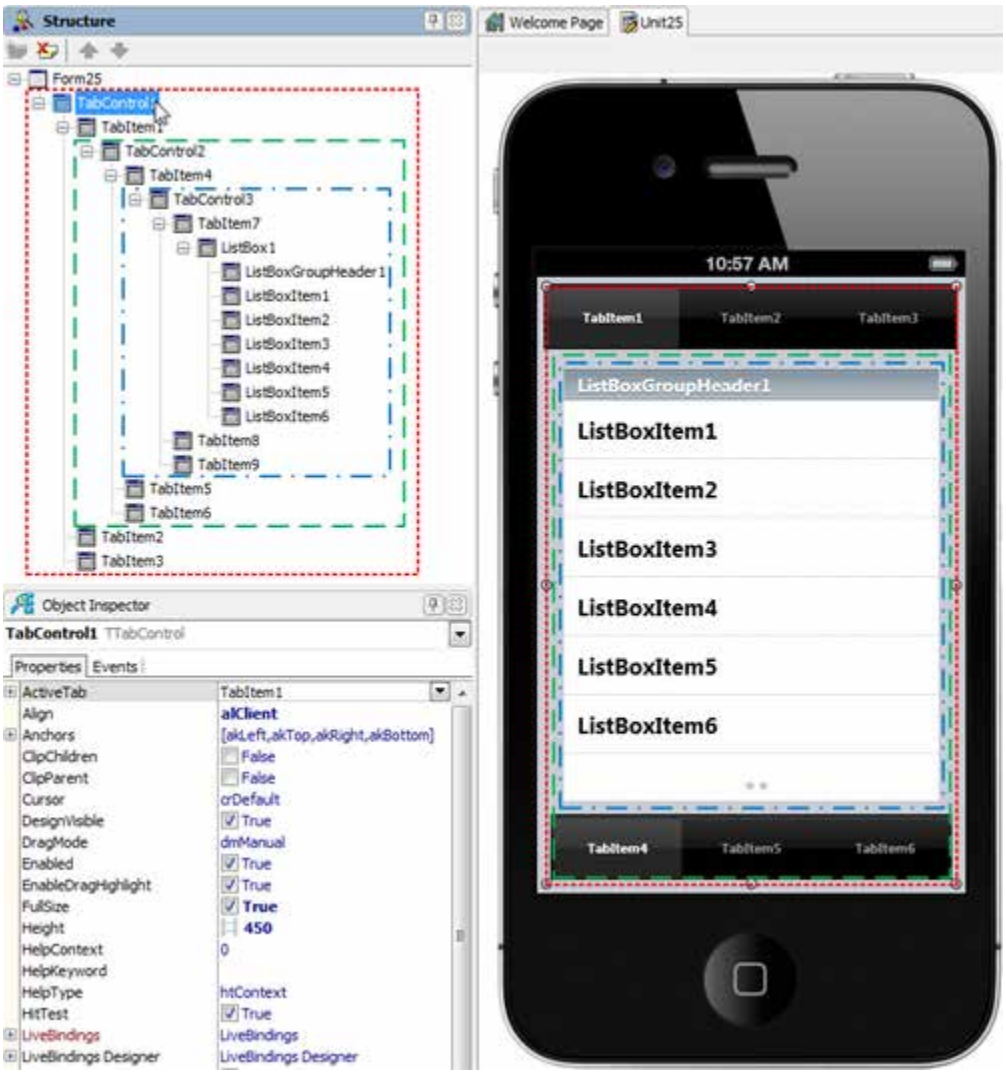
5. In the Object Inspector, set the **StyleLookup** property to be **tabitemcustom**:




# Defining Controls within a TabControl

As discussed, each Tab Page can contain any number of controls including another TabControl. In such a case, you can easily browse and manage different tab pages in the Structure View:

iOS


Structure
Form25
TabControl1
TabItem1
TabControl2
TabItem4
TabControl3
TabItem7
ListBox1
ListBoxGroupHeader1
ListBoxItem1
ListBoxItem2
ListBoxItem3
ListBoxItem4
ListBoxItem5
ListBoxItem6
TabItem8
TabItem9
TabItem5
TabItem6
TabItem2
TabItem3
Object Inspector
TabControl1 TTabControl
Properties
Events
ActiveTab
TabItem1
Align
alClient
Anchors
[akLeft,akTop,akRight,akBottom]
ClipChildren
False
ClipParent
False
Cursor
crDefault
DesignVisible
True
DragMode
dmManual
Enabled
True
EnableDragHighlight
True
FullSize
True
Height
450
HelpContext
0
HelpKeyword
HelpType
htContext
HitTest
True
LiveBindings
LiveBindings
LiveBindings Designer
LiveBindings Designer
Welcome Page
Unit25
10:57 AM
TabItem1
TabItem2
TabItem3
ListBoxGroupHeader1
ListBoxItem1
ListBoxItem2
ListBoxItem3
ListBoxItem4
ListBoxItem5
ListBoxItem6
TabItem4
TabItem5
TabItem6

**Android**


# Changing the Page at Run Time

## By the User Tapping the Tab

If Tabs are visible (when the TabPosition property is set to other than None), an end user can simply tap a Tab to open the associated page.

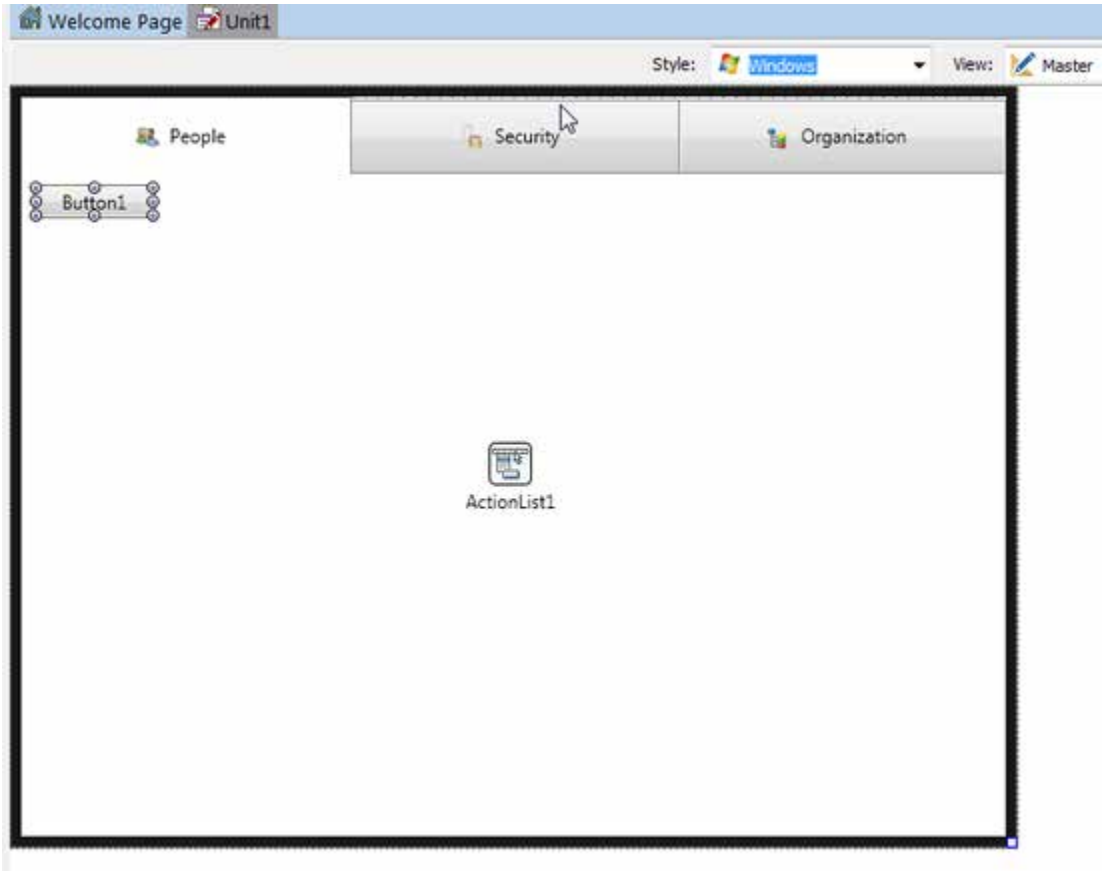
## By Actions and an ActionList

An action corresponds to one or more elements of the user interface, such as menu commands, toolbar buttons, and controls. Actions serve two functions:

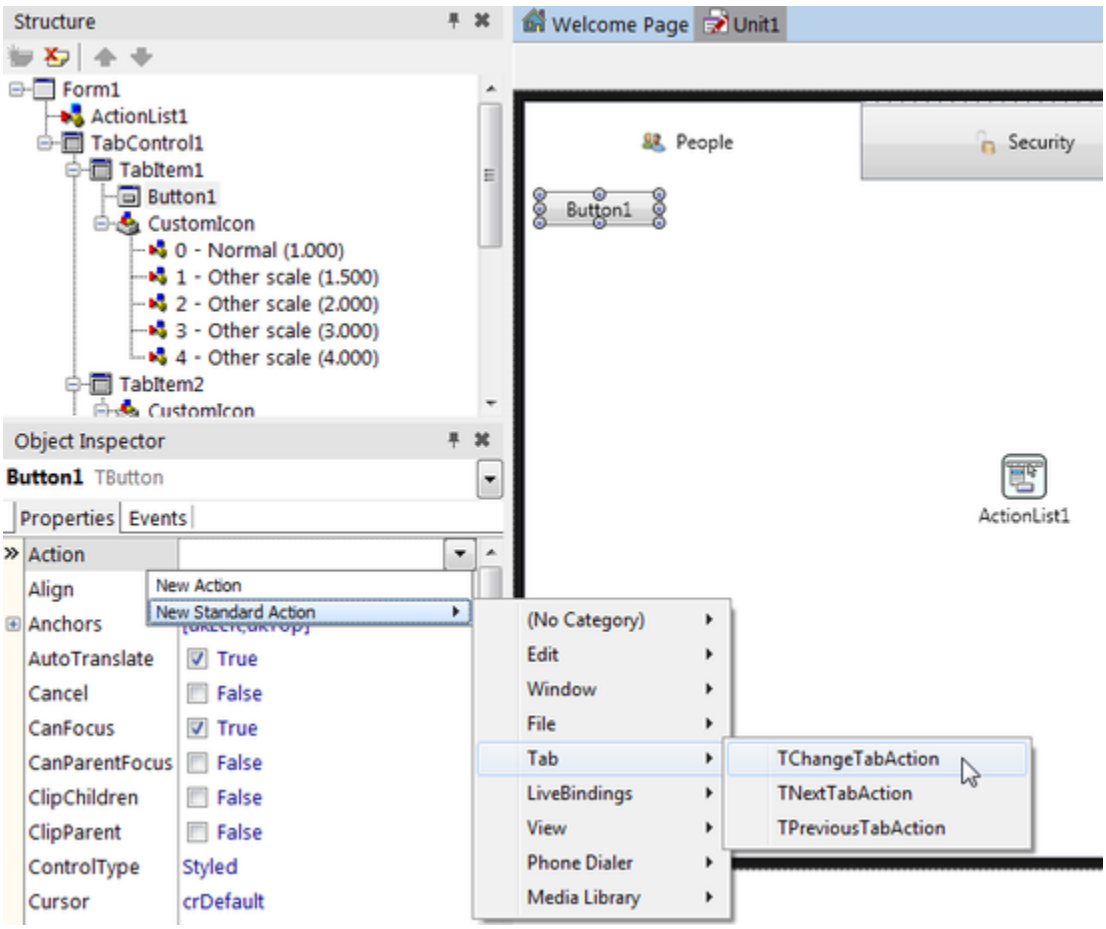
- Actions represent properties common to the user interface elements, such as whether a control is enabled or whether a check box is selected.
- Actions respond when a control fires, for example, when the application user clicks a button or chooses a menu item.

Here are the steps to enable a user to move to different tab pages by clicking a button:

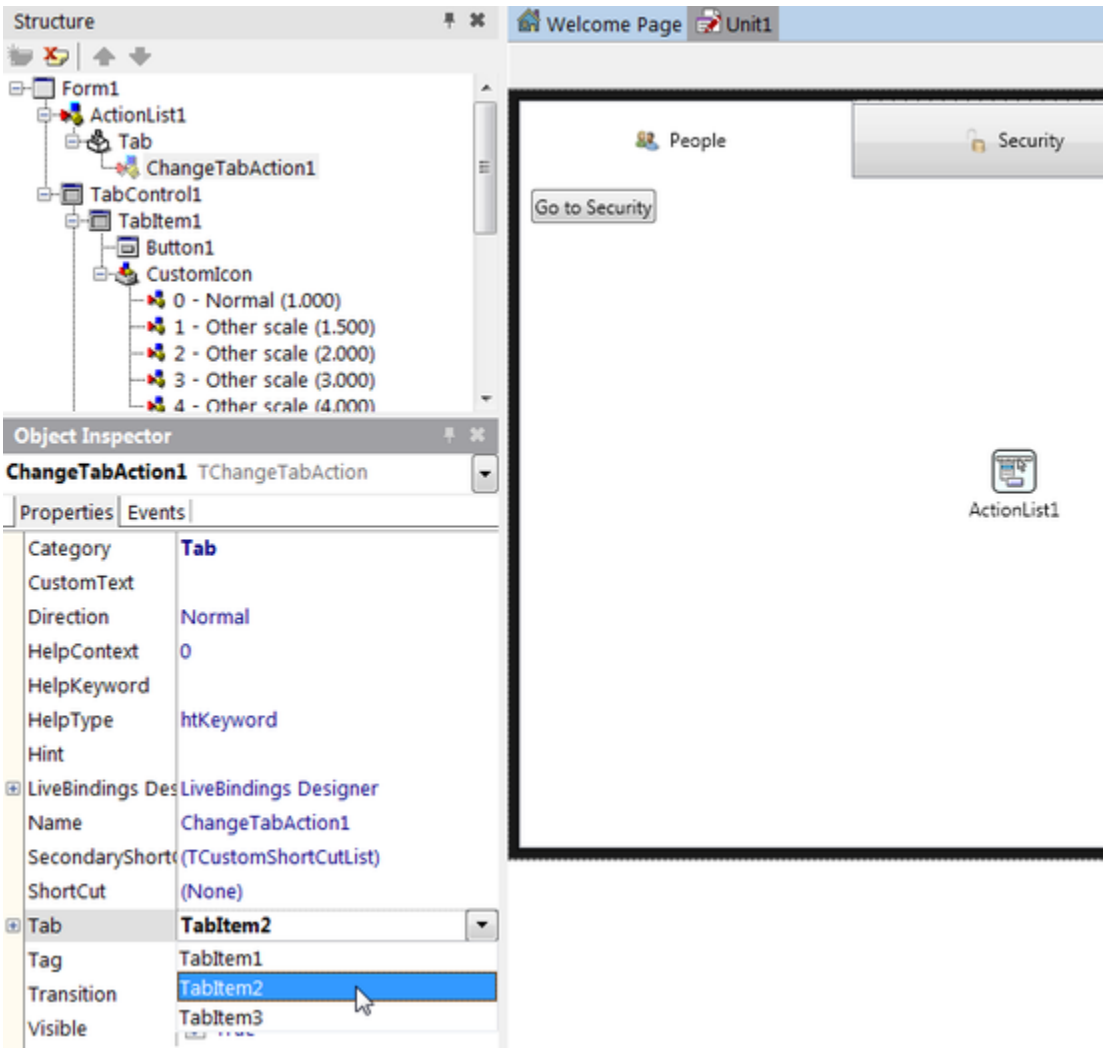
1. On the Form Designer, click **TabItem1** to select it.
2. From the **Tool Palette**, add a **TActionList** component to the form, and then add a TButton to **TabItem1**:



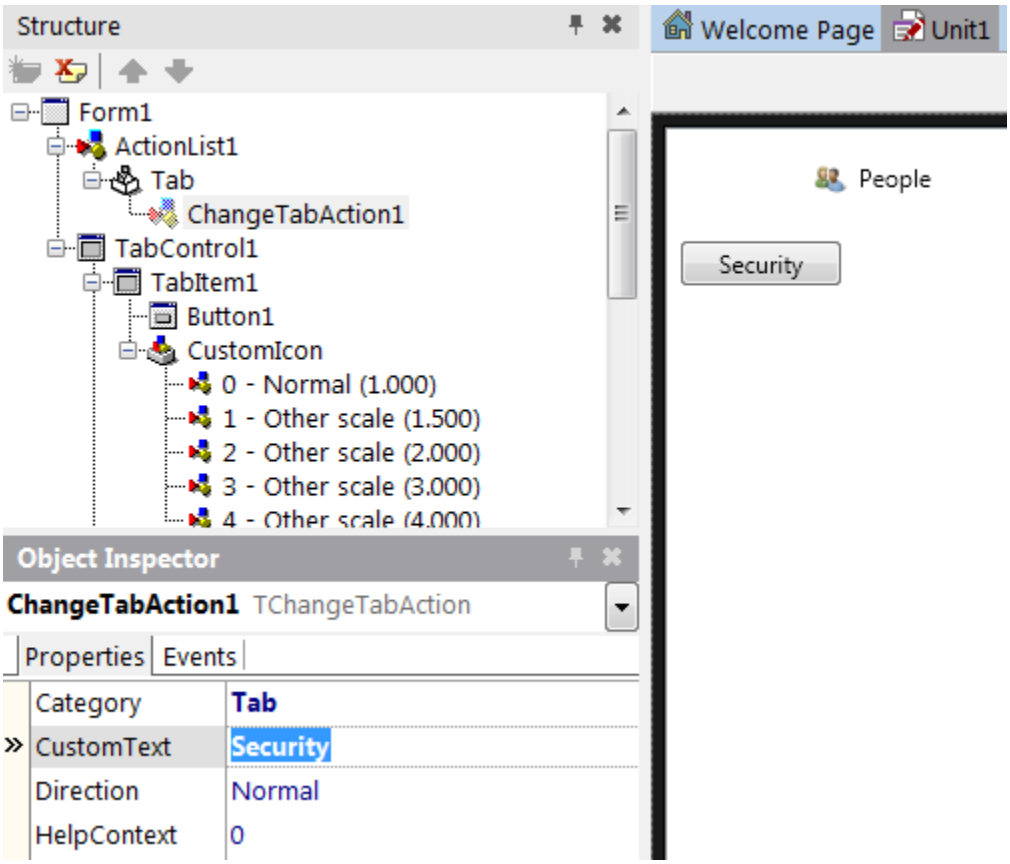

3. With the button selected, in the Object Inspector, select **Action | New Standard Action | Tab >** **TChangeTabAction** from the drop-down menu. After the user clicks this button, the action you just defined is performed (the tab page changes):



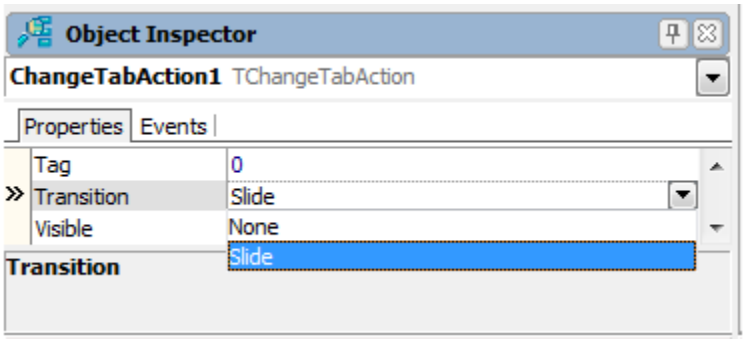

4. Select **ChangeTabAction1** in the Structure View, and then select **TabItem2** for the **Tab** property in the Object Inspector. By linking to **TabItem2**, this action can change the page to **TabItem2**:




5. With the previous step, the caption (the Text property) of the button is automatically changed to "Go To Security" because the caption of **TabItem2** is "Security" in our example. Set the CustomText property of the **ChangeTabAction1** component to **Security** as shown below and change the size of the button to fit the new caption text, if required.

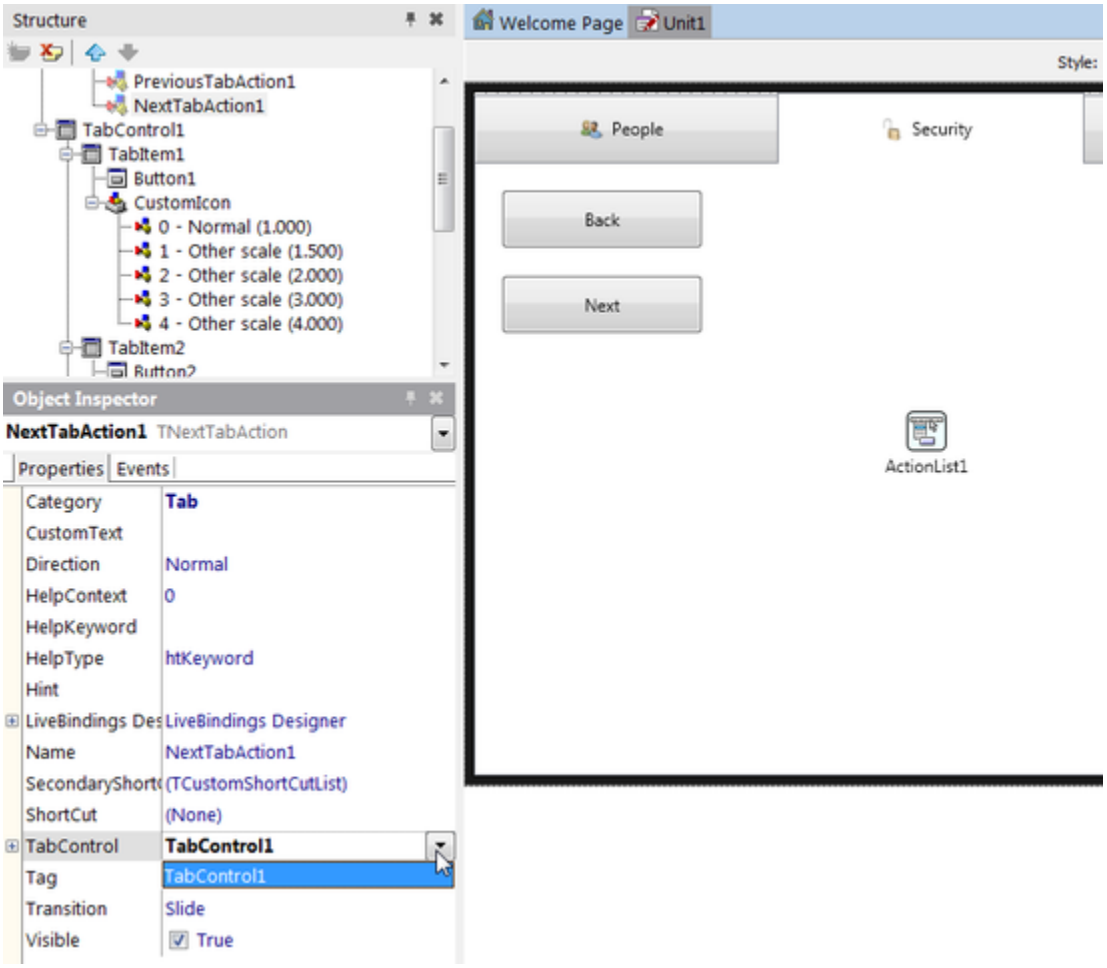



6. ChangeTabAction also supports the **Slide** animation to indicate a transition between pages. To use it, set the Transition property to **Slide**:




7. On the Form Designer, select **TabItem2** and drop two TButtons from the Tool Palette to **TabItem2**.
8. On the Form Designer, select **Button2** and in the Object Inspector, select **Action | New Standard Action | Tab >** TPreviousTabAction from the drop-down menu.
9. On the Form Designer, select **Button3** and in the Object Inspector, select **Action | New Standard Action | Tab >** TNextTabAction from the drop-down menu.
10. Select **PreviousTabAction1** in the Structure View and in the Object Inspector, set its TabControl property to **TabControl1**.

11. Select **NextTabAction1** in the Structure View and in the Object Inspector, set its TabControl property to **TabControl1**.



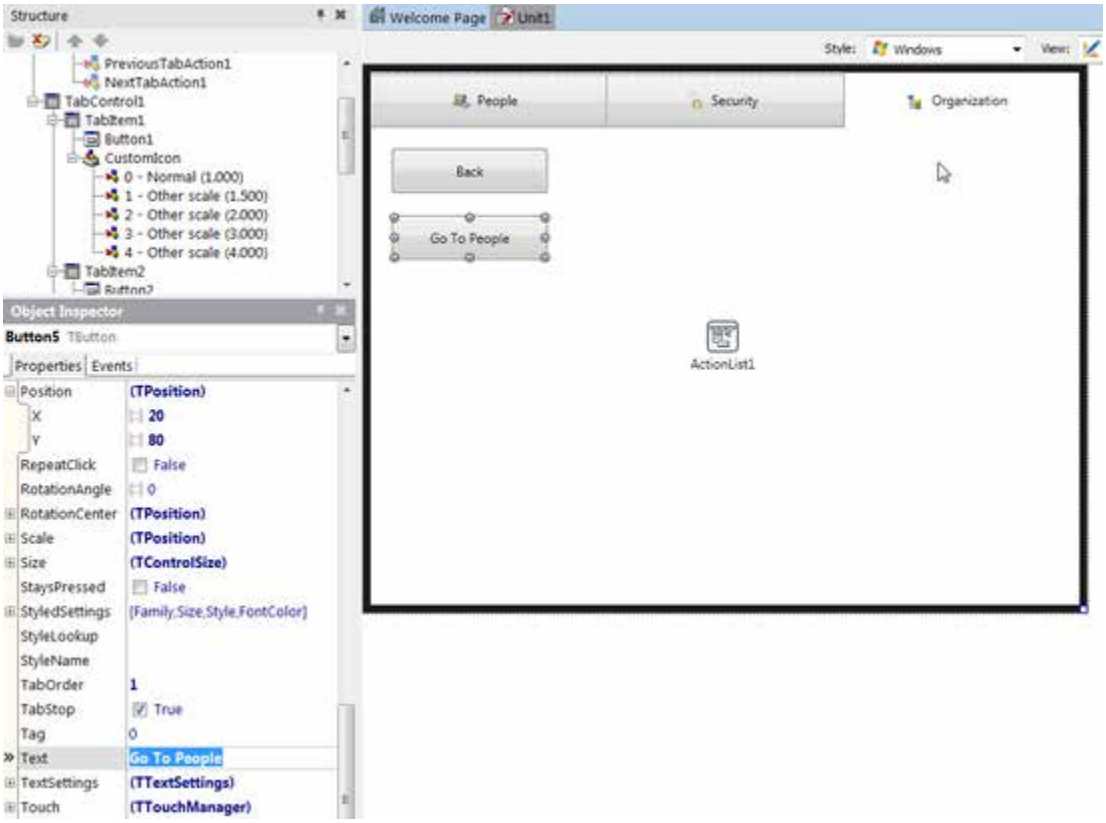

12. On the Form Designer, select **TabItem3** and drop a TButton from the Tool Palette to **TabItem3**.
13. In the Object Inspector, set the Action property of the button to **PreviousTabAction1**.

## By Source Code

You can use any of the following three ways to change the active tab page from your source code, by clicking the button.

### Assign an Instance of TTabItem to the ActiveTab Property

1. From the Tool Palette, add a TButton to **TabItem3**.
2. In the Object Inspector, set its **Text** property to **Go To People**.




3. On the Form Designer, double-click the button to create the OnClick event handler and add the following code:

**Delphi:**

```
TabControl1.ActiveTab := TabItem1;
```

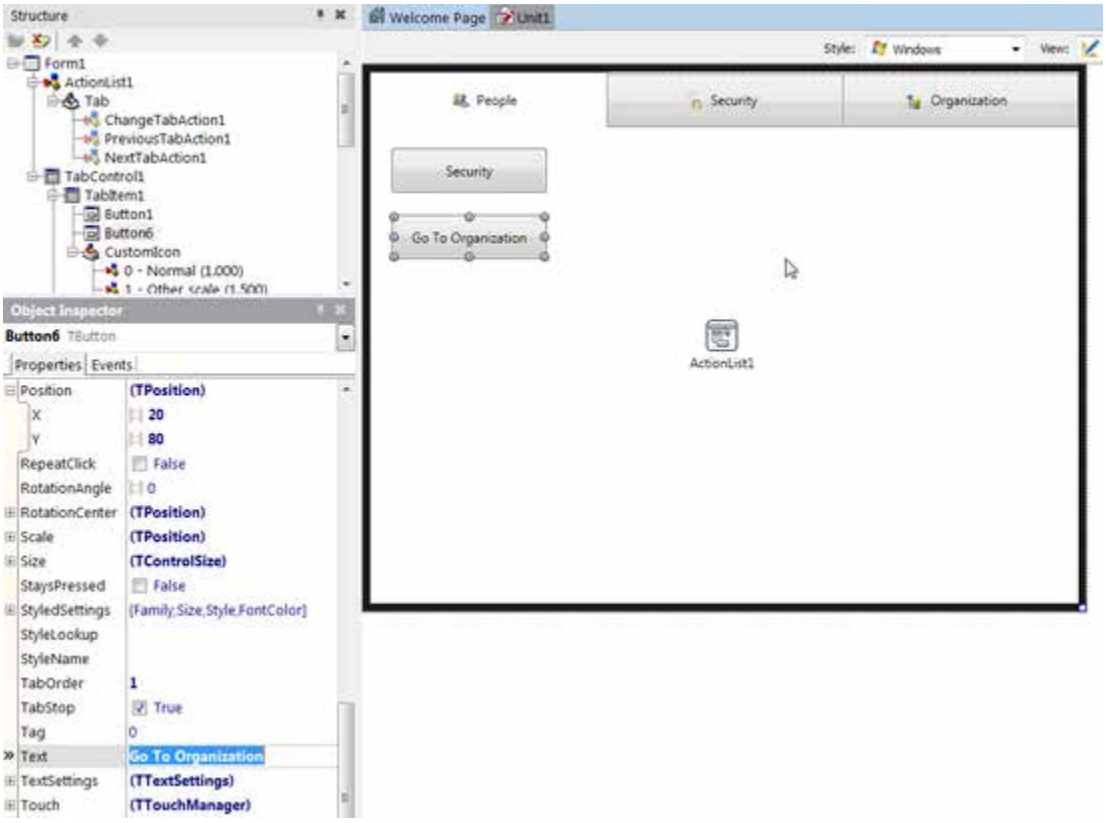
**C++:**

```
TabControl1->ActiveTab = TabItem1;
```

## Change the TabIndex Property to a Different Value

The TabIndex property is a zero-based Integer value. You can specify any number between `0` and `TabControl1.TabCount - 1`.

1. From the Tool Palette, add a TButton to **TabItem1**.
2. In the Object Inspector, set its **Text** property to **Go To Organization**.




3. On the Form Designer, double-click the button to create the OnClick event handler and add the following code:

**Delphi:**

```
TabControl1.TabIndex := 2;
```

**C++:**

```
TabControl1->TabIndex = 2;
```

## Call the ExecuteTarget Method of a Tab Action

You can call the ExecuteTarget method for any of the tab control actions (TChangeTabAction, TNextTabAction, and TPreviousTabAction). You must ensure to define the TChangeTabAction.Tab, TPreviousTabAction.TabControl or the TNextTabAction.TabControl properties.

**Delphi:**

```
// You can set the target at run time if it is not defined yet.
ChangeTabAction1.Tab := TabItem2;
```

```
// Call the action
ChangeTabAction1.ExecuteTarget(nil);
```

**C++:**

```
// You can set the target at run time if it is not defined yet.
ChangeTabAction1->Tab = TabItem2;

// Call the action
ChangeTabAction1->ExecuteTarget(NULL);
```

# See Also

- Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)
- Mobile Tutorial: Using the Web Browser Component (iOS and Android)
- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)
- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

## Samples

- Tab Sliding

# Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

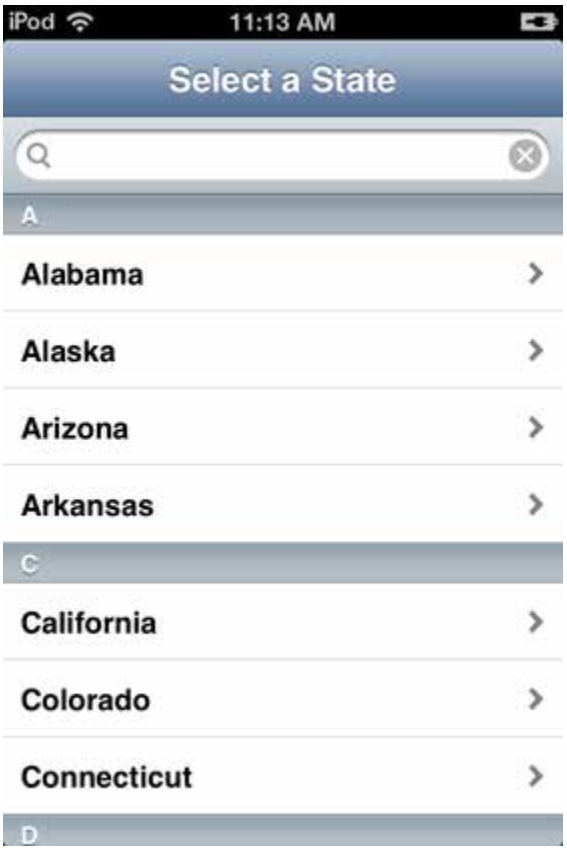
## Using ListBox Components to Display a Table View in Mobile Platforms

On the mobile platform, FireMonkey uses the **FMX.ListBox.TListBox** component to present a **Table View** in a mobile style, like the following ListBoxes.

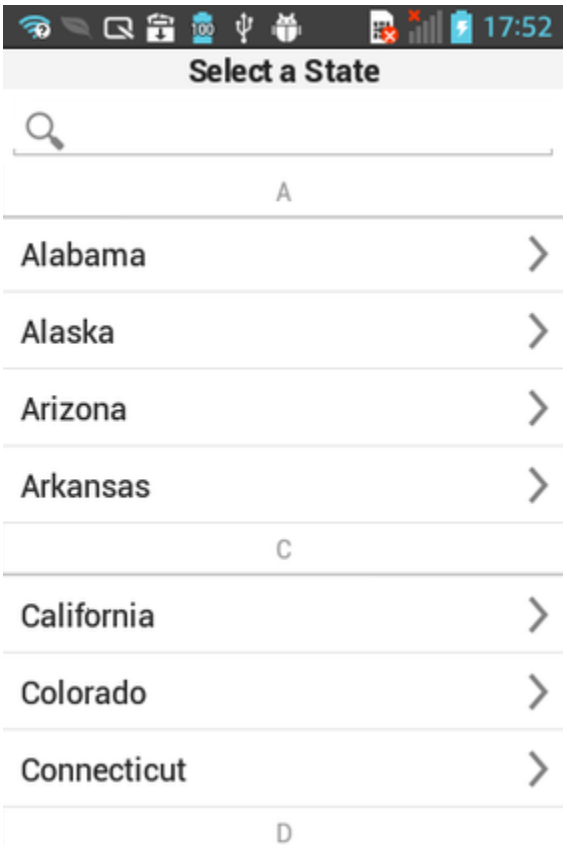
> **Note:** FMX.ListBox.TListBox performance can be slow on mobile. Use TListView if you want to develop more complex applications, especially apps with large databases.

Plain List

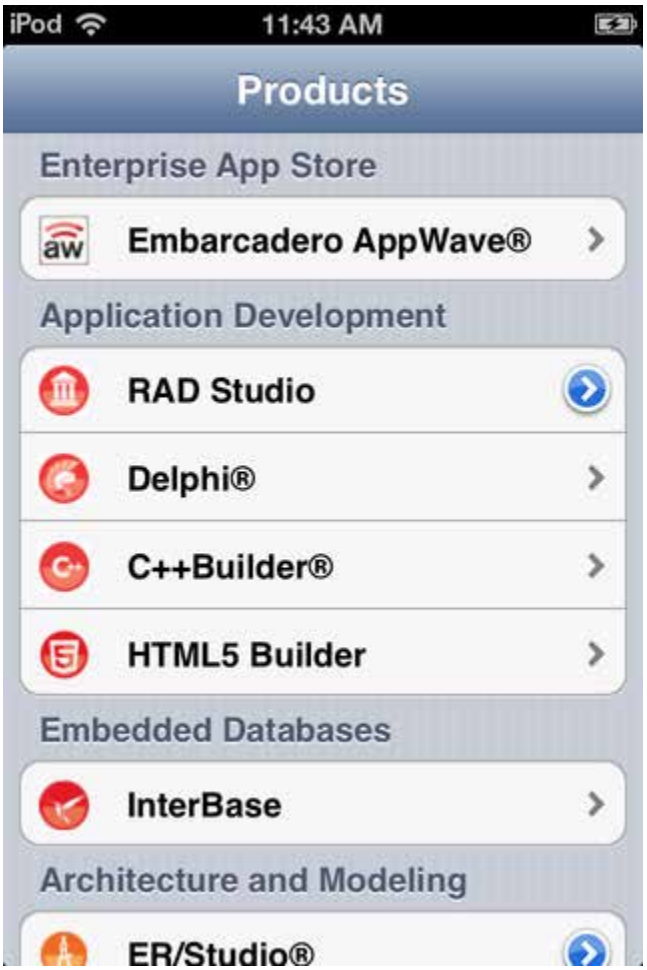
iOS




Android (LG E-612)



## Grouped List



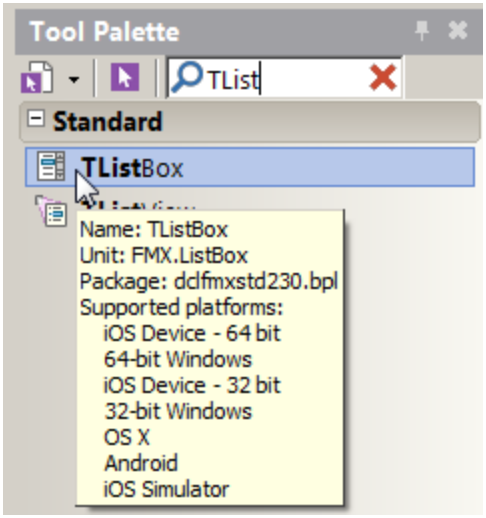

## Search Box

You can add a search box to a ListBox. With a search box, users can easily narrow down a selection from a long list as in the following pictures.


This tutorial describes the basic steps to build items for a Table View in your multi-device applications for mobile platforms.

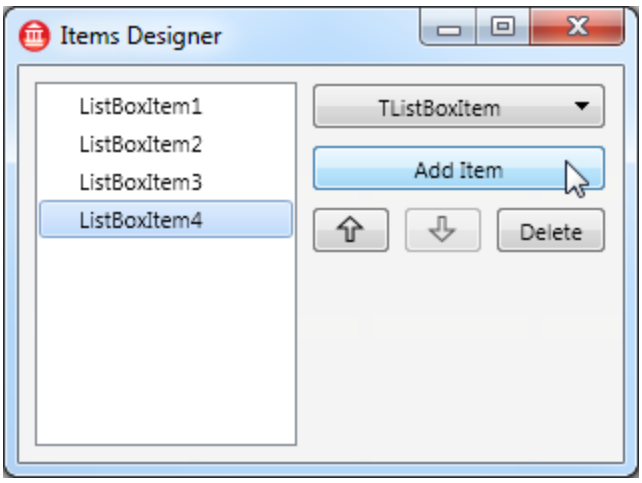
# Create Items on the ListBox Component

1. Select:
   - For Delphi: **File > New > Multi-Device Application - Delphi > Blank Application**
   - For C++Builder: **File > New > Multi-Device Application - C++Builder > Blank Application**
2. Select the **TListBox** component in the Tool Palette, and drop it on the Form Designer. To find TListBox, enter a few characters (such as "TList") in the **Search** box of the Tool Palette:

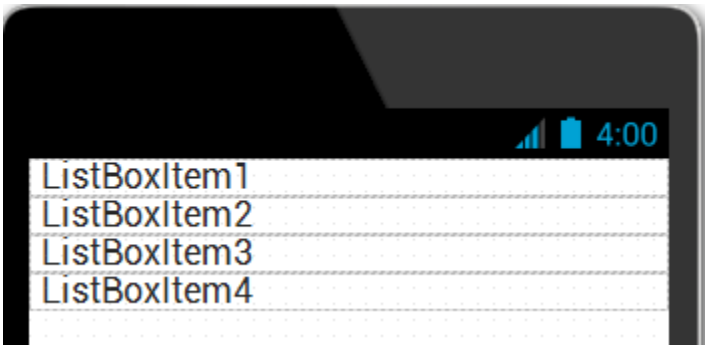



3. Select the **TListBox** component on the Form Designer, go to the Object Inspector and select **Client** for the **Align** property.
4. On the Form Designer, right-click the TListBox component, and select **Items Editor**:


5. On the **Items Designer**, click the **Add Item** button several times to add several items to the ListBox:

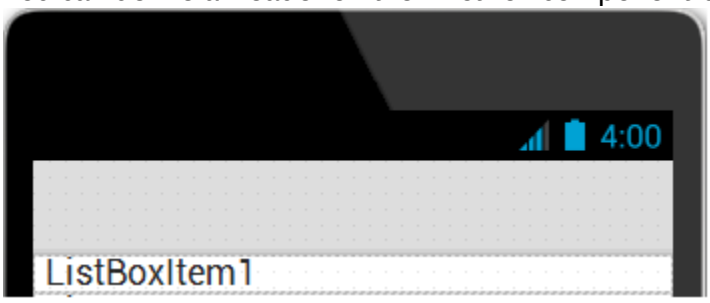



6. Close the Items Designer. Now you can find your ListBox Items on the TListBox component. For example:



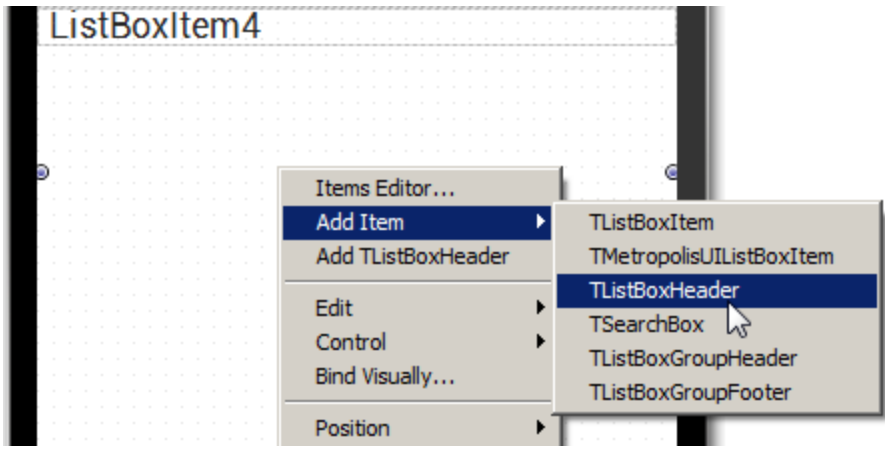

## Add a Header

You can define a Header on the TListBox component by using the following steps:

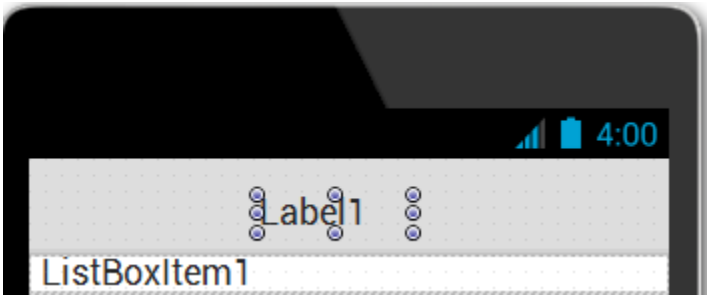



A Header for a TListBox

1. On the Form Designer, right-click the TListBox component, and select **Add Item > TListBoxHeader**:




2. On the Tool Palette, select the **TLabel** component and drop it on top of the **TListBoxHeader** component you just added:

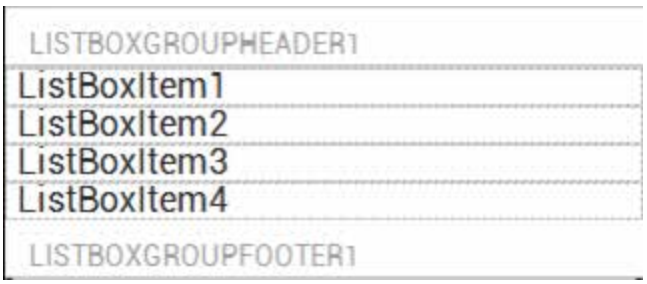



3. In the Object Inspector, change the properties of the **TLabel** component as follows:

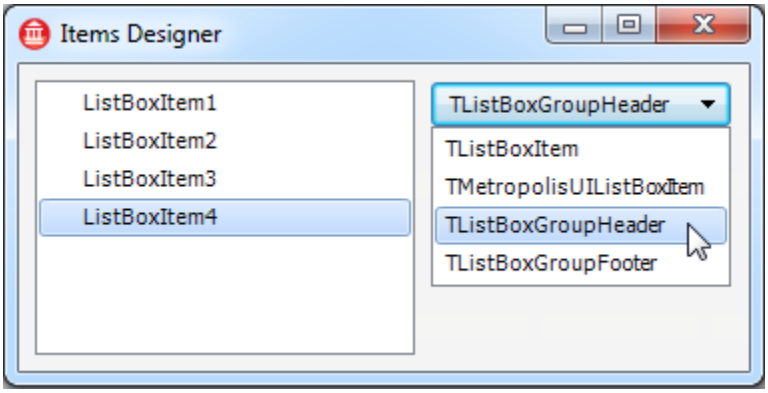
| Property | Value |
|---|---|
| Align | Client |
| StyleLookup | toollabel |
| TextSettings.HorzAlign | Center |
| Text | (Text value as you want) |

## Add a Group Header/Footer to the List

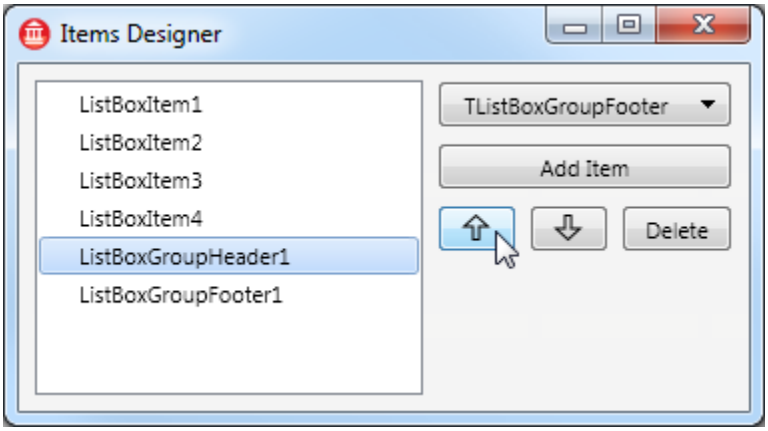
You can define a Group Header and a Group Footer for items on TListBox as follows:




1. On the Form Designer, right-click the **TListBox** component, and select **Items Editor**.
2. On the **Item Designer**, select **TListBoxGroupHeader** from the drop-down list, and then select **Add Item**:



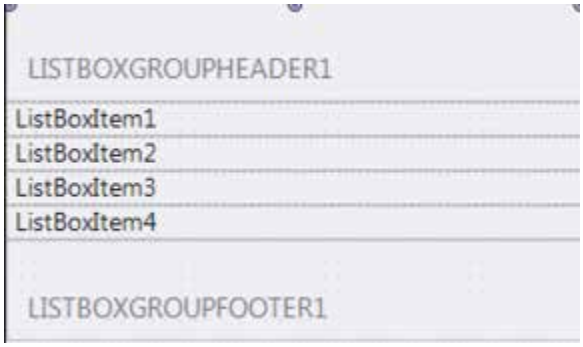

3. Select **TListBoxGroupFooter** from the drop-down list, and then select **Add Item**.
4. Select **ListBoxGroupHeader1** in the list of items, and click the **Up** button several times until this item becomes the top item on the list:




5. Close the dialog box. Now you have a Group Header and a Group Footer on the TListBox component.

## Show List Items as Separate Grouped Items

Items on a ListBox can be shown as either a **Plain** list or a **Grouped** list. This choice is controlled by the GroupingKind property and the StyleLookup property, as shown in the following graphic:

| Show Items as Plain List | Show Items as Grouped List |
| --- | --- |
|  |  |
| Plain = GroupingKind Property Value | Grouped = GroupingKind Property Value |
| listboxstyle = StyleLookup Property Value | transparentlistboxstyle = StyleLookup Property Value |

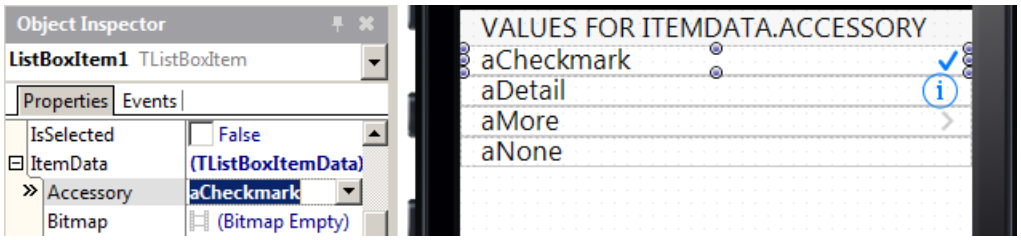


You can select the GroupingKind property and the StyleLookup property in the Object Inspector when the ListBox is selected in the Form Designer.

## Add a Check Box or Other Accessory to a ListBox Item

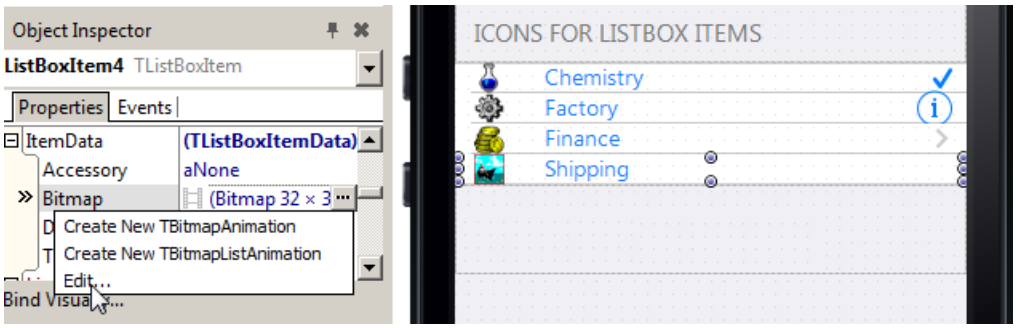
Each item in a TListBox can use an Accessory such as Check Mark through the ItemData.Accessory property. The following picture shows the value you can assign to ItemData.Accessory and the Accessory assigned:


You can select the Accessory property in the Object Inspector when ListBox Item is selected in the Form Designer.




## Add an Icon to a ListBox Item

Each Item on a ListBox component can contain Bitmap data, as an **Icon**, through the ItemData.Bitmap property:



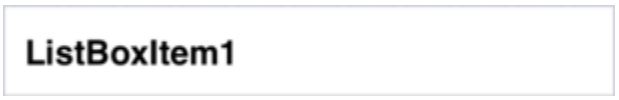
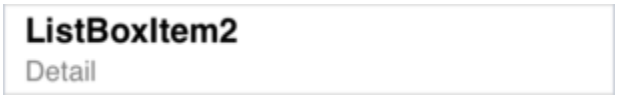
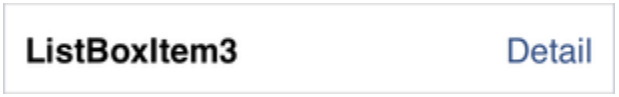

You can select the **Bitmap** property in the Object Inspector when the ListBoxItem is selected in the Form Designer.

In order to view the **Icon**, you must select a StyleLookup which supports the **Bitmap** property. Change the StyleLookup property to listboxitemleftdetail.

## Add Detail Information to an Item

You can add additional text information to each item on the ListBox component.

Specify additional text in the ItemData.Detail property, and select the location of the Detail Text through the StyleLookup property, as shown in the following table:

| StyleLookup property | Look & Feel |
| --- | --- |
| listboxitemnodetail |   |
| listboxitembottomdetail |   |
| listboxitemrightdetail |   |
| listboxitemleftdetail |   |

### Running Your Application

Run the application either by choosing Run > Run or by pressing F9.

## Create Your ListBox Application

1. Select:
   - For Delphi: File > New > Multi-Device Application - Delphi > Blank Application
   - For C++Builder: File > New > Multi-Device Application - C++Builder > Blank Application
2. Select the TListBox component in the Tool Palette, and drop it on the Form Designer.
3. Select the **TListBox** component on the Form Designer, go to the Object Inspector and select **Client** for the **Align** property.

### Add Items to a ListBox from Your Code

To add regular items to a ListBox, you can simply call the **Items.Add** method as shown in the following code snippet:

- **Delphi:**

```
ListBox1.Items.Add('Text to add');
```

- **C++:**

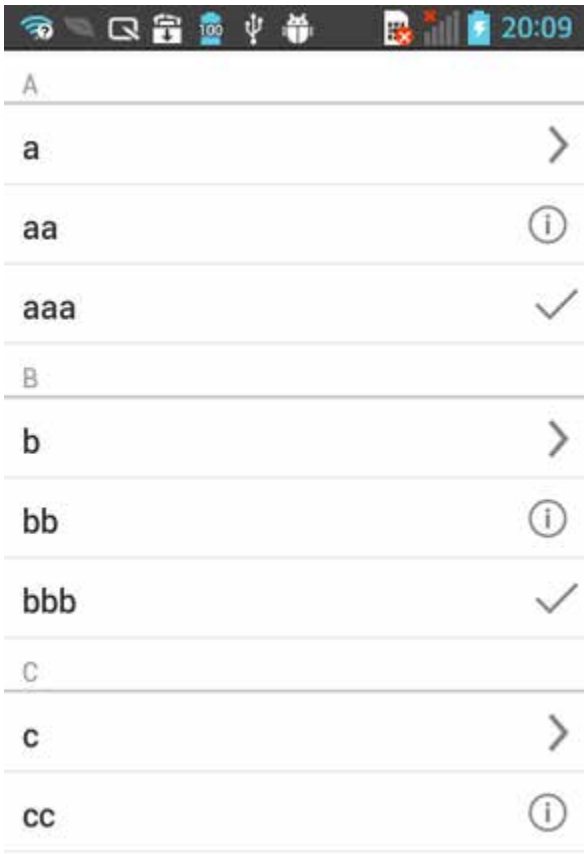
```
ListBox1->Items->Add("Text to add");
```

If you want to create items other than a simple item, or control other properties, you can create an instance of the item first, and then add it to the list box.

The following sample codes add items to a ListBox, as shown in the picture:

iOS


Android (LG E-612)




Delphi:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  c: Char;
  i: Integer;
  Buffer: String;
  ListBoxItem : TListBoxItem;
  ListBoxGroupHeader : TListBoxGroupHeader;
begin
  ListBox1.BeginUpdate;
  for c := 'a' to 'z' do
  begin
    // Add header ('A' to 'Z') to the List
```

```
    ListBoxGroupHeader := TListBoxGroupHeader.Create(ListBox1);
    ListBoxGroupHeader.Text := UpperCase(c);
    ListBox1.AddObject(ListBoxGroupHeader);

    // Add items ('a', 'aa', 'aaa', 'b', 'bb', 'bbb', 'c', ...) to the list
    for i := 1 to 3 do
    begin
      // StringOfChar returns a string with a specified number of repeating
characters.
      Buffer := StringOfChar(c, i);
      // Simply add item
      // ListBox1.Items.Add(Buffer);

      // or, you can add items by creating an instance of TListBoxItem by yourself
      ListBoxItem := TListBoxItem.Create(ListBox1);
      ListBoxItem.Text := Buffer;
      // (aNone=0, aMore=1, aDetail=2, aCheckmark=3)
      ListBoxItem.ItemData.Accessory := TListBoxItemData.TAccessory(i);
      ListBox1.AddObject(ListBoxItem);
    end;
  end;
  ListBox1.EndUpdate;
end;
```

C++:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{

  char c;
  int i;
  String Buffer ;
  TListBoxItem *ListBoxItem  ;
  TListBoxGroupHeader *ListBoxGroupHeader  ;

  ListBox1->BeginUpdate();
  for (c = 'a'; c <= 'z'; c++)
  {
        // Add header ('A' to 'Z') to the List
        ListBoxGroupHeader = new TListBoxGroupHeader(ListBox1);
        ListBoxGroupHeader->Text = UpperCase(c);
        ListBox1->AddObject(ListBoxGroupHeader);

        // Add items ('a', 'aa', 'aaa', 'b', 'bb', 'bbb', 'c', ->->->) to the list
        for (i = 1; i < 4; i++)
        {
          // StringOfChar returns a string with a specified number of repeating
characters->
          Buffer = StringOfChar(c, i);
          // Simply add item
          // ListBox1->Items->Add(Buffer);

          // or, you can add items by creating an instance of TListBoxItem by yourself
          ListBoxItem = new TListBoxItem(ListBox1);
          ListBoxItem->Text = Buffer;
          // (aNone=0, aMore=1, aDetail=2, aCheckmark=3)
          ListBoxItem->ItemData->Accessory =
static_cast<TListBoxItemData::TAccessory>(i);
          ListBox1->AddObject(ListBoxItem);
        };
```

```
  };
  ListBox1->EndUpdate();
}
```

## Create an Overflow Menu

An overflow popup menu is accessed via the Action Bar and is used to provide access to additional items or items that are used less often.

In FireMonkey, you can easily implement an overflow menu using TListBox:

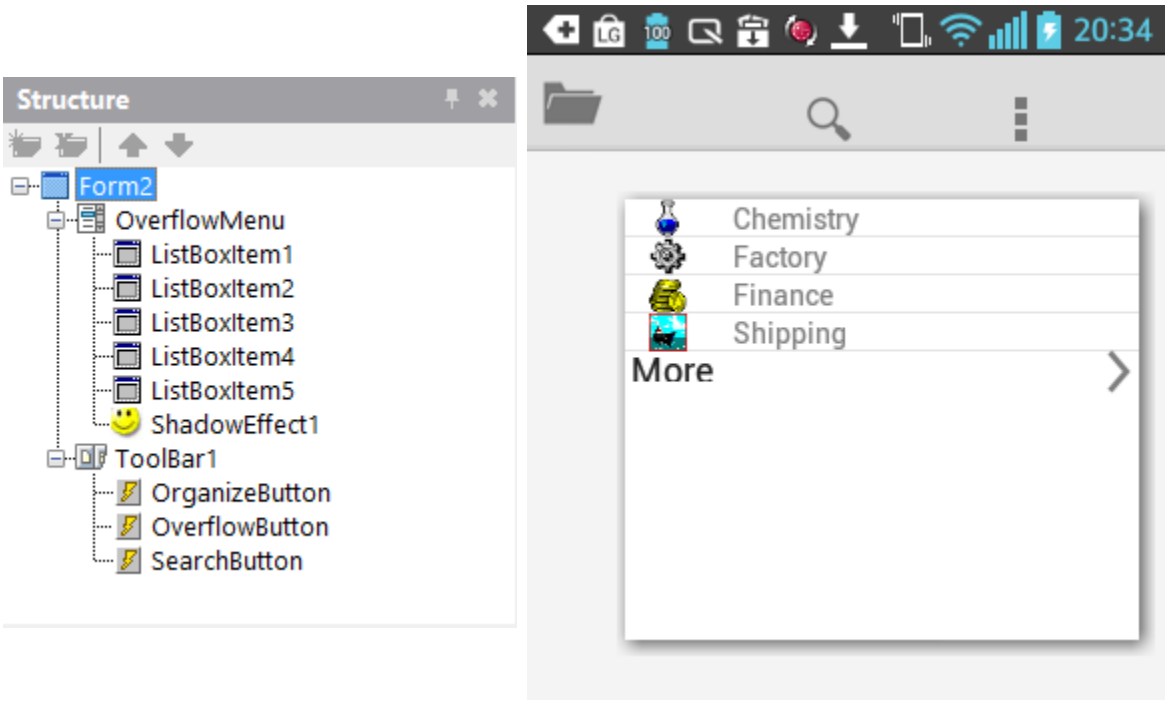
1. Add a TToolBar component on the form and set the alignment to **Top**.
2. Place three TSpeedButton components on the TToolBar component:

   - For the first TSpeedButton:
     - Set the Align property to **Left**.
     - Change the Name property to **OrganizeButton**.
     - Set the StyleLookup to **organizetoolbutton**.
   - For the second TSpeedButton:
     - Set the Align property to **Right**.
     - Change the Name property to **OverflowButton**.
     - Select **detailstoolbutton** for the StyleLookup property.
   - For the last TSpeedButton:
     - Set the Align property to **Right**.
     - In Object Inspector expand the Margins node and set the **Right** margin to **5**.
     - Change the Name property to **SearchButton**.
     - Set the StyleLookup to **searchtoolbutton**.
3. Drop a TListBox to the form.
   - Add five TListBoxItem from **Items Editor**.
   - Check **akTop** and **akRight** from the Anchors property of the TListBox component.
   - Set Height to **220**.
   - Change Name to **OverflowMenu**.

- Set the Visible property to **False**.

4. For the first four TListBoxItem components in TListBox go to Object Inspector, expand ItemData:
   - Define the Bitmap property.
   - Change the Text property to the text value that you want.
   - Select **listboxitemleftdetail** for the StyleLookup property.
5. For the last TListBoxItem, in the Object Inspector expand ItemData:
   - Set Accessory to **aMore** and Text to **More**.
6. Add a TShadowEffect component to the overflow menu.

**Structure View** **Android LG-E612**




## Creating the Event Handler for the Overflow Button

In the Form Designer, double-click the **OverflowButton** component. Add the following code to this event handler:

- **Delphi:**

```
procedure TForm1.OverflowButtonClick(Sender: TObject);
begin
OverflowMenu.Visible := not OverflowMenu.Visible; //change the visibility status
  if OverflowMenu.Visible then // the Overflow Menu is displayed
  begin
```

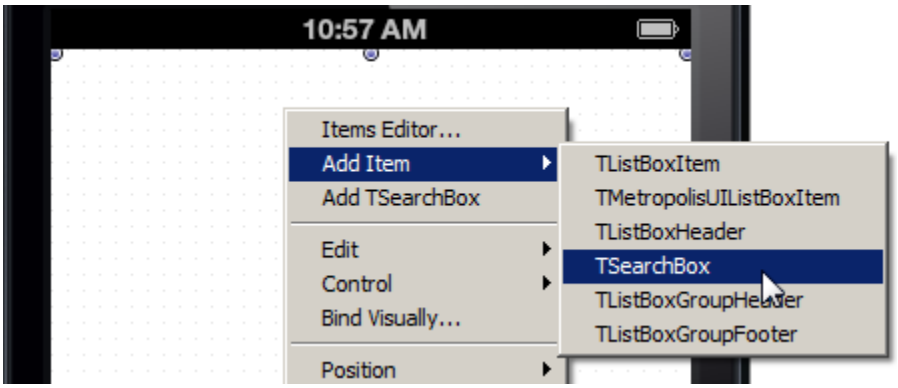
```
    OverflowMenu.BringToFront;
    OverflowMenu.ItemIndex := -1; //  the ItemIndex property specifies the currently
selected item(default value is -1 that means that no item is selected)
    OverflowMenu.ApplyStyleLookup;
    OverflowMenu.RealignContent; // realigns the children TListBoxItem controls of the
OverflowMenu TListBox
  end;
end;
```

- **C++:**

```
void __fastcall TForm1::OverflowButtonClick(TObject *Sender)
{
        OverflowMenu->Visible = !(OverflowMenu->Visible); //change the visibility
status
                if (OverflowMenu->Visible) {   // the Overflow Menu is displayed
                OverflowMenu->BringToFront();
                OverflowMenu->ItemIndex = -1; //  the ItemIndex property specifies the
currently selected item(default value is -1 that means that no item is selected)
                OverflowMenu->ApplyStyleLookup();
                OverflowMenu->RealignContent(); // realigns the children TListBoxItem
controls of the OverflowMenu TListBox
        }
}
```

## Add a Search Box

- To add a Search Box to the ListBox component, right-click the **TListBox** component and simply select **Add Item > TSearchBox** from the context menu:




- To add it to the Action Bar:
  - Set the Visible property to **False**.
  - To create the event handler for the **SearchButton**, double-click it and add the following code:

**Delphi:**

```
procedure TForm1.SearchButtonClick(Sender: TObject);
```

```
begin
  SearchBox1.Visible := not SearchBox1.Visible; //change the visibility status
end;
```

**C++:**

```
void __fastcall TForm1::SearchButtonClick(TObject *Sender) {
        SearchBox1->Visible = !(SearchBox1->Visible); //change the visibility status
}
```

## Running Your Application

1. Select either:
    - **Run > Run**
    - **Run > Run Without Debugging**
2. To invoke the overflow menu, click the vertical ellipsis on the Action bar.
3. To view the search box, click the **SearchButton**.

Android (Samsung Tab 2.0)


Displaying the overflow menu

Android (Samsung Tab 2.0)

Displaying the search box

## See Also

- FMX.ListBox.TListBox
- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)
- Mobile Tutorial: Using LiveBindings to Populate a ListBox in Mobile Applications (iOS and Android)
- Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)
- Mobile Tutorial: Using the Web Browser Component (iOS and Android)
- Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)

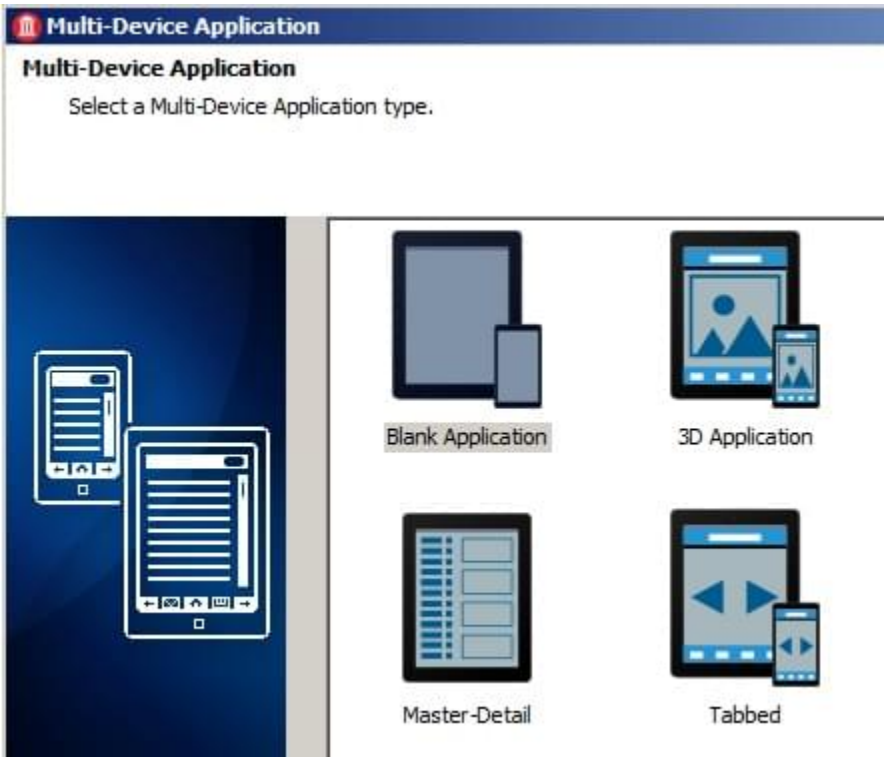
# Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

This tutorial shows how to use **LiveBindings Designer** to populate a **FireMonkey ListView component** from a **TPrototypeBindSource** containing some sample data. The tutorial shows you how to add the sample data and how to make the bindings between the prototyping source and the list view component in order to fill the list.

Like every LiveBinding, this tutorial requires **no code**. However, in order to create a useful application, you do need to add event handlers and other code.

## Step 1: Creating the Project

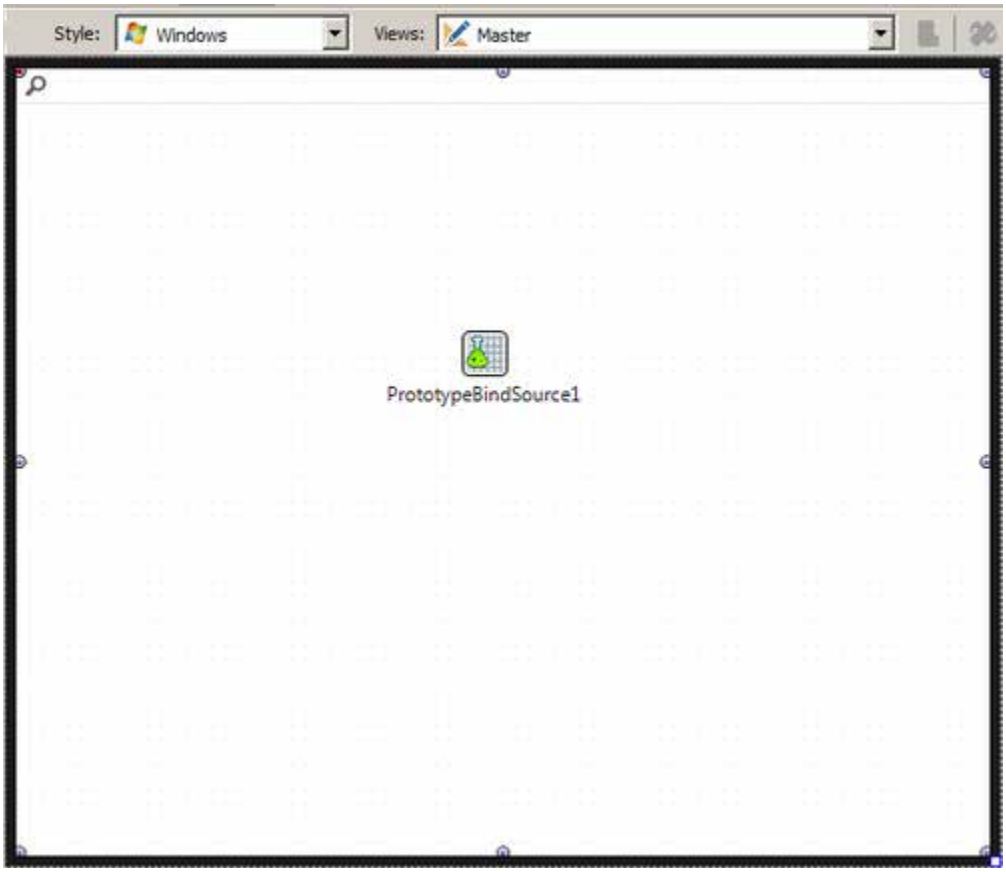
1. Create a new project. Choose a Multi-Device Application for this example. In the wizard, choose **Blank Application**.




2. In the Tool Palette, locate a **TListView** component and drop it onto the form.
3. Add a **TPrototypeBindSource** component to the form.

4. On the form, select the **ListView1** component, and then in the Object Inspector, set the **Align** property to `Client` and the **SearchVisible** property to `True`.

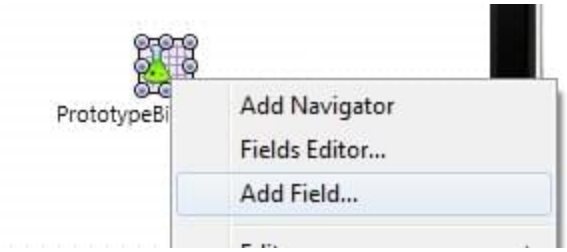
The form should now look like the following screen, before you set the **Style** or **View** in the Form Designer:



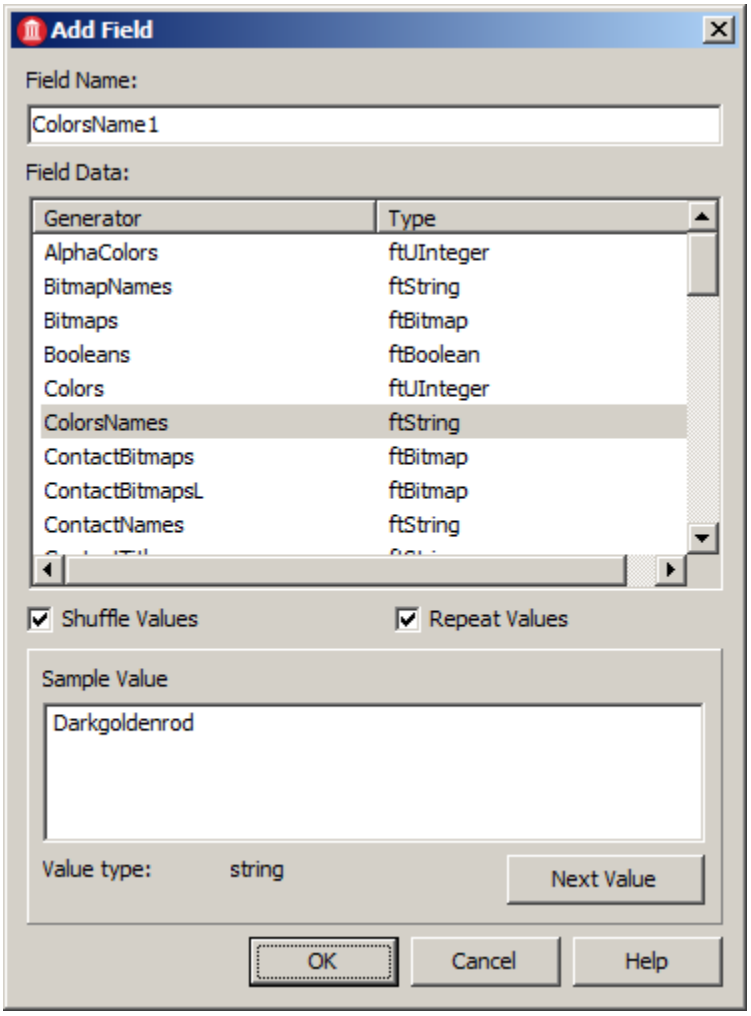

**Note:** For more information about the selection of the **Style** and **Views**, see Style Selector and Using FireMonkey Views.

# Step 2: Adding Fields

1. Right-click the TPrototypeBindSource component and then select **Add Field...**.



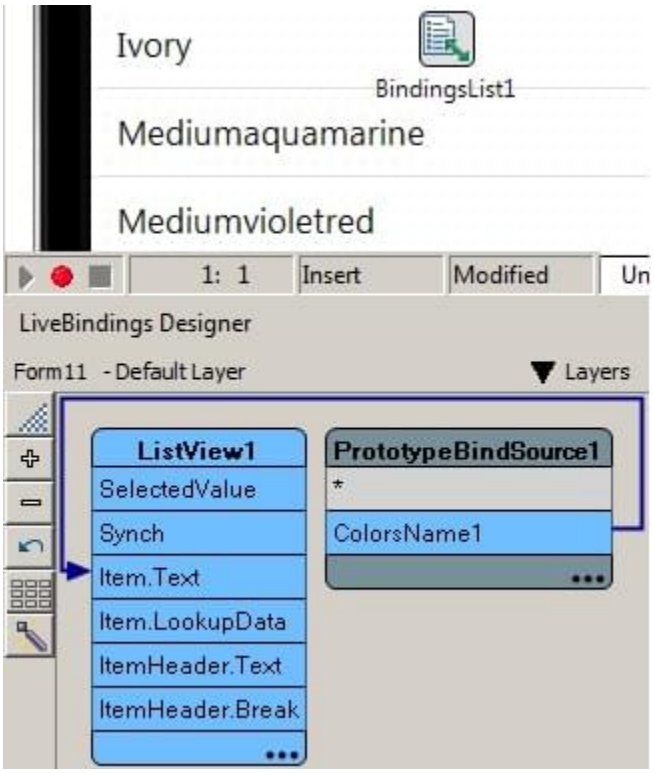

2. From the Add Field dialog box, select **ColorsNames** and click **OK**.




# Step 3: Creating LiveBindings

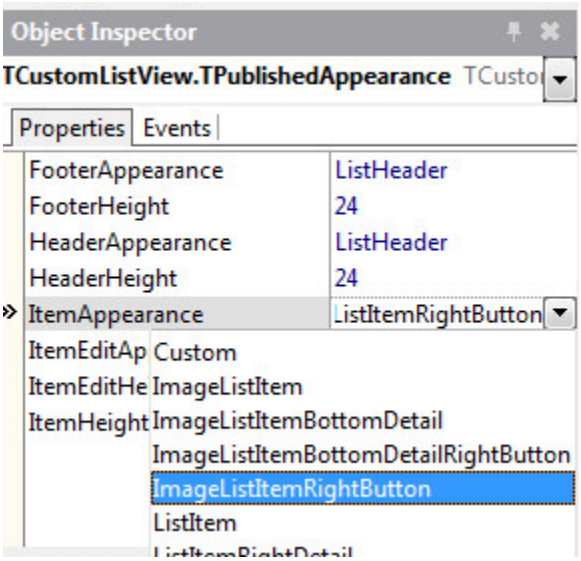
1. Open the LiveBindings Designer (choose **View > LiveBindings Designer**), and drag the `ColorsName1` property of the TPrototypeBindSource onto the `Item.Text` property of the ListView to bind these properties.

The ListView component automatically populates its items with color names from the prototyping data component:




2. Set TListView.ItemAppearance to **ImageListItemRightButton**, as follows:

   - Place focus on the ListView component by selecting it (in the Structure View, the Form Designer, or the Object Inspector).
   - Then, in the Object Inspector, locate the **ItemAppearance** node, expand it and change the **ItemAppearance** property to `ImageListItemRightButton`:



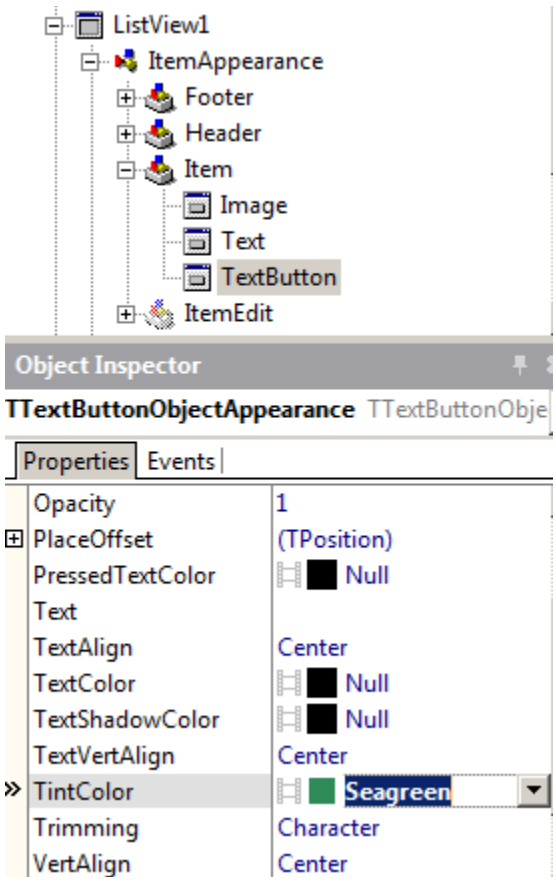

3.

4. Optionally, you can apply a tint to the **TListView** text buttons. Do the following:

   - Place focus on the ListView component by selecting it (in the Structure View, the Form Designer, or the Object Inspector).
   - In the Structure View, under **ItemAppearance**, expand **Item** and then select **TextButton**.
   - In the Object Inspector, locate the **TintColor** property and set its value to an appropriate value, such as `Seagreen`.

   The following image shows both the Structure View and the Object Inspector. In the Structure View, **Item** is expanded and **TextButton** is selected, and in the Object Inspector, TintColor is set to `Seagreen`:

5.

6. 

7.

> **Note**: At design time, the tint color that you applied to text buttons might not be visible. To make your changes visible, choose the **Master** view in the Style selector to change the current style of your Form Designer to either `Android` or `iOS`. For details, see Form Designer.

At this point in the tutorial, you have configured the ListView component to display an image on the left-hand side of the item text, and to display a button on the right-hand side of the item text.
In the next step, you populate the image and the button with sample data.

# Step 4: Adding More Fields (Bitmaps, Currency)

You need to add two more fields in order to make the list view component display an image and some text on the button associated with each list item.

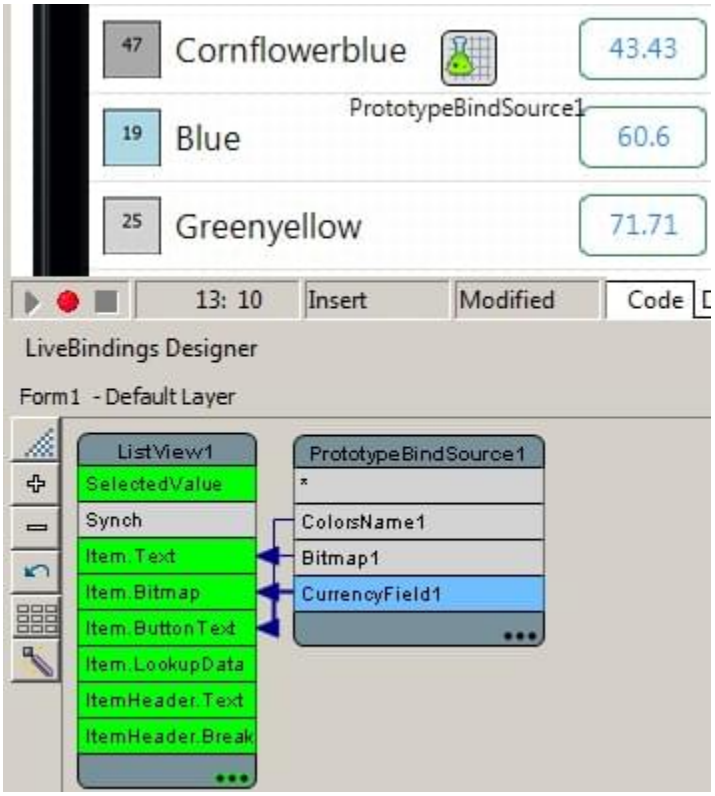
1. Right-click the TPrototypeBindSource component and select **Add Field...**.
2. In the Add Field dialog box, `Ctrl+Click` to select **Bitmaps** and **Currency** field data. When finished, click **OK**.
3. Go to the LiveBindings Designer and do the following:

1. Connect the `Bitmap1` property of the prototyping source data to the `Item.Bitmap` property of the list view component.

This step adds a button representing the color and number of each list view item, such as Blue 19.

2. Connect the `CurrencyField1` property from the prototyping source data to the `Item.ButtonText` property of the list view component.

This step displays the currency field value on the button located on the right-hand side of each list view item.




Now the list view displays some color data associated with each item and also displays sample currency data on the button associated with each list item.

# Step 5: Adding the onButtonClick Event Handler

To create a usefull application, you can add the **onButtonClick** event handler that fires when you click a ListView item.

### To add the onButtonClick event handler

1. On the multi-device application form, select the **ListView1** component.

2. In the Object Inspector, open the **Events** tab, and then double-click **OnButtonClick**.
3. In the Code Editor, implement an appropriate **OnButtonClick** event handler.

The following sample code adds the event handler that displays a message box when you click a ListView item:

**Delphi:**

```
procedure TForm1.ListView1ButtonClick(const Sender: TObject;
  const AItem: TListViewItem; const AObject: TListItemSimpleControl);
begin
   ShowMessage(AItem.Text + ' ' +AItem.ButtonText + ' is clicked.');
end;
```

**C++Builder:**

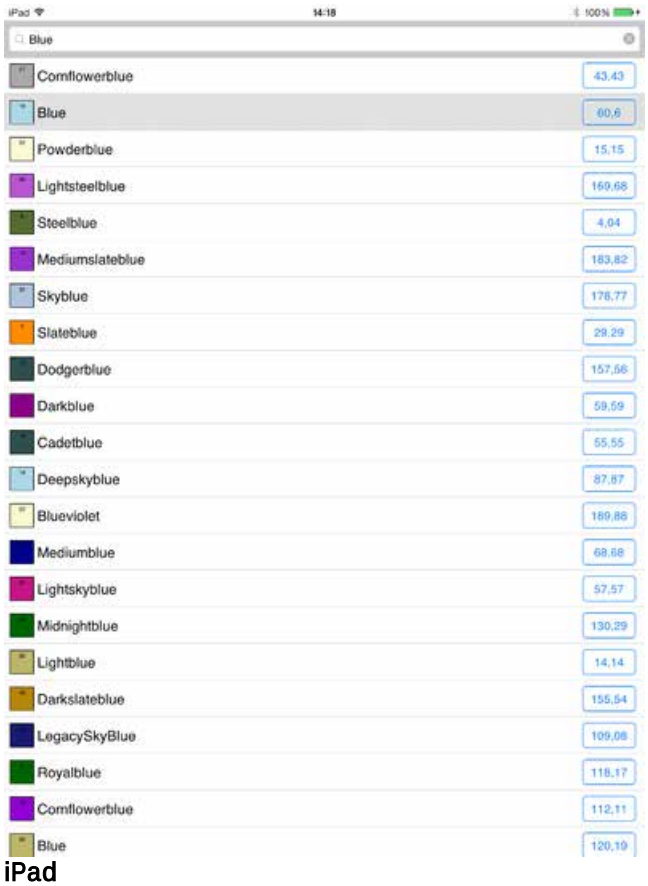
```
void __fastcall TForm1::ListView1ButtonClick(TObject * const Sender,
        TListViewItem * const AItem, TListItemSimpleControl * const AObject) {
        ShowMessage(AItem->Text + " " + AItem->ButtonText + " is clicked.");
}
```

# The Results

To see your mobile app as it would appear on a mobile device, you need to configure your system as described in the appropriate **Setup** tutorial, available here, and set the **View** to a target mobile device (such as **iPhone 4"**) in the Form Designer. Then you need to complete the necessary steps for deploying your app to the target mobile platform.

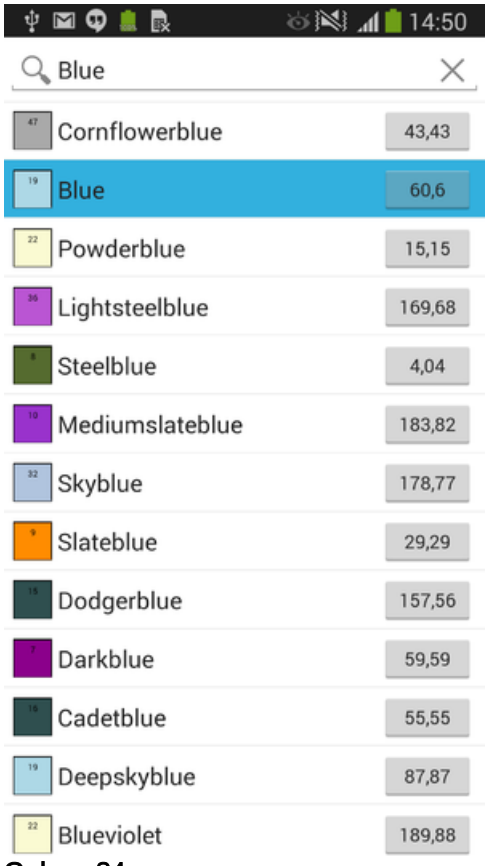
Then you can run the application on your mobile device, either by pressing **F9** or by choosing Run > Run.

iOS



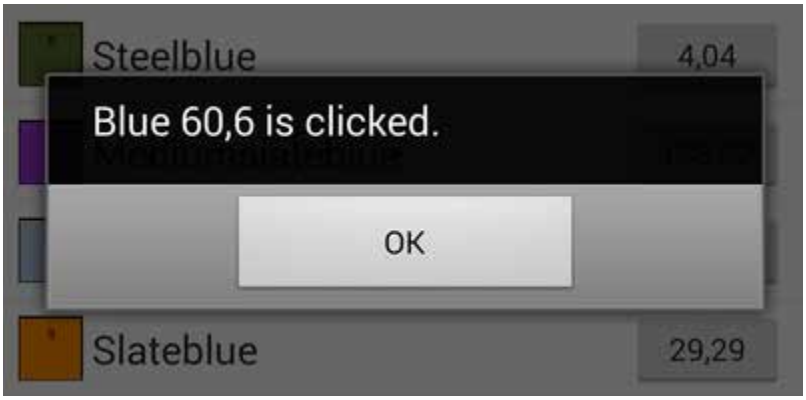

iPad

Android




Galaxy S4

If you click the **Blue** item, the application displays the following message box:




# See Also

- Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)

- Mobile Tutorial: Using LiveBindings to Populate a ListBox in Mobile Applications (iOS and Android)
- Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android)
- Customizing FireMonkey ListView Appearance
- FMX.ListView.TListView
- TPrototypeBindSource
- LiveBindings in RAD Studio
- LiveBindings Designer
- Using FireMonkey Views
- Using Styled and Colored Buttons on Target Platforms
- RAD Studio Tutorials

# Mobile Tutorial: Using LiveBindings to Populate a ListBox in Mobile Applications (iOS and Android)

This tutorial guides you through the steps of connecting data to a FireMonkey ListBox control on your mobile devices, using LiveBindings.

## Step 1: Creating the Project

In this project, you need a list box, a rectangle, and also a prototyping binding source. To add these components, follow the steps below:

1. Select:
   - For Delphi: **File > New > Multi-Device Application - Delphi**
   - For C++: **File > New > Multi-Device Application - C++Builder**
2. Select the TListBox component in the Tool Palette, and drop it on the Form Designer.
3. Select the list box component on the Form Designer, and in the Object Inspector, locate the `DefaultItemStyles.ItemStyle` property. Set it to the **listboxitemrightdetail** style.

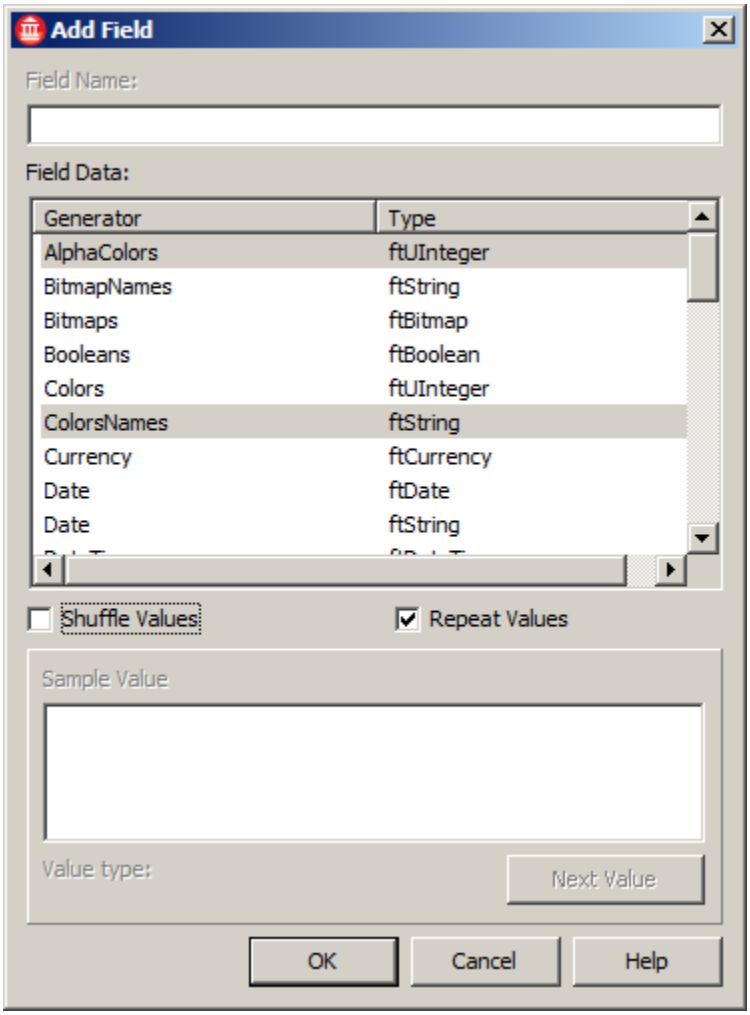

4. Unselect the list box component by clicking in a blank area of the Form Designer.
5. Select the TRectangle component in the Tool Palette, and drop it on the Form Designer.
6. Select the rectangle component on the Form Designer, and in the Object Inspector, set the `Align` property to `Bottom`.
7. Select the list box component and set its `Align` property to `Client`.
8. Now drop a TPrototypeBindSource from the **Tool Palette** onto your form.

At this point, your form should be similar to the following iOS screen:


You need to add sample data for **colors** and **color names** to the TPrototypeBindSource component in order to support the purpose of this tutorial and to link to the list box and the rectangle.

1. Right-click the TPrototypeBindSource component, and then click **Add Field** on the context menu.

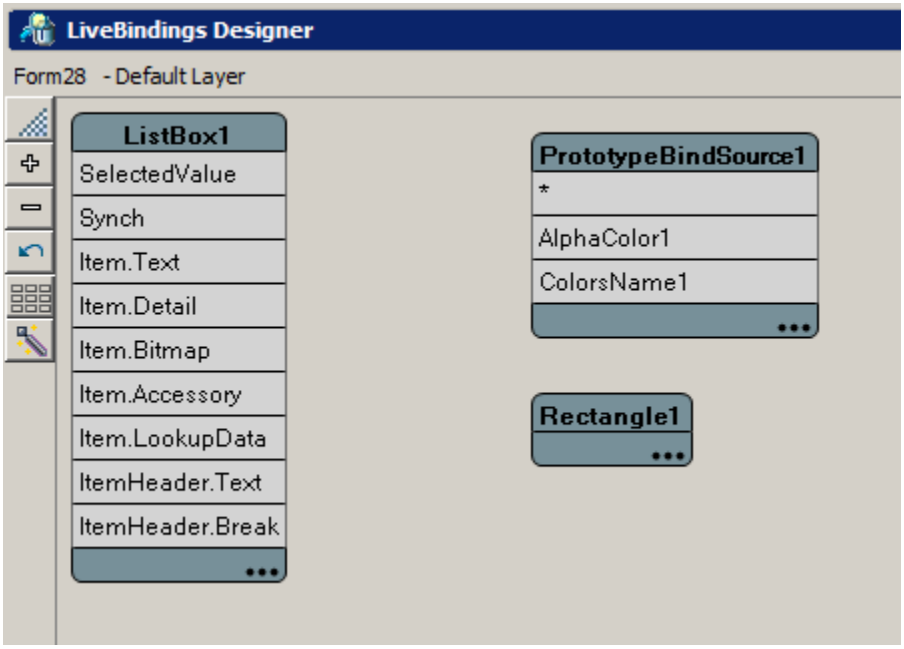
2. Select **AlphaColors** and **ColorsNames** from the **Field Data** list, and clear the **Shuffle Values** check box.
   This operation adds sample alphabetically sorted data (alpha color values and names).




3. Validate your selection by clicking the **OK** button. Now the sample data is ready to be used through LiveBindings.

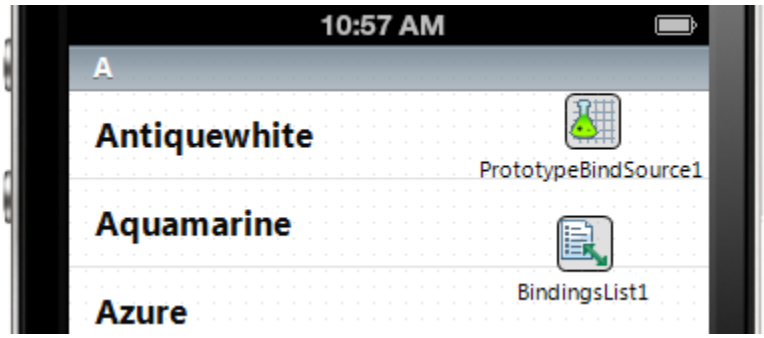
# Step 2: Creating the LiveBindings

Open the LiveBindings Designer. The diagram with no connections is similar to the following image:




1. Bind **ColorsName1** from **PrototypeBindSource1** to **Item.Text** from **ListBox1**. (Click **ColorsName1** and drag the mouse to **Item.Text**.) Now the list box displays all the color names in the prototyping binding source.
2. Bind **ColorsName1** from **PrototypeBindSource1** to **ItemHeader.Text** from **ListBox1**. Now the list box also displays headers with all the color names in the prototyping binding source.
3. In the LiveBindings Designer, click the binding that connects **ColorsName1** to **ItemHeader.Text**.
4. In the Object Inspector, locate the `FillHeaderCustomFormat` property and select the **SubString(%s, 0, 1)**binding expression from the drop-down list.

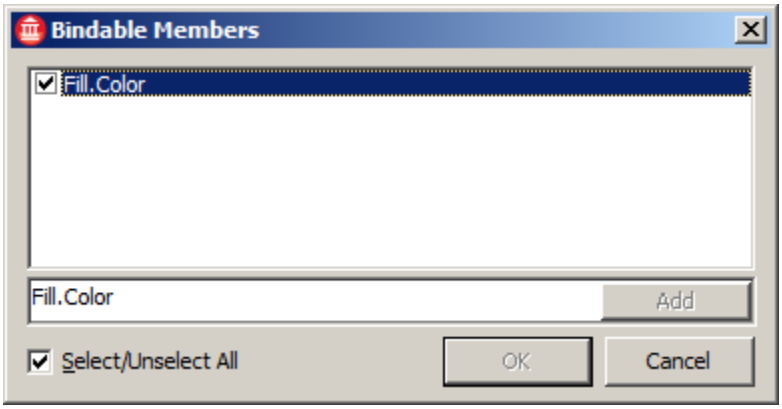
At this point, the list box groups all the color names into alphabetical categories:




5. Bind **AlphaColor1** from **PrototypeBindSource1** to **Item.LookupData** from **ListBox1**. This ensures that the selection of the color name also points to the correct alpha color value.

6. Bind **AlphaColor1** from **PrototypeBindSource1** to **Item.Detail** from **ListBox1**. This ensures the alpha color value appears in the right side of the item in the list box.

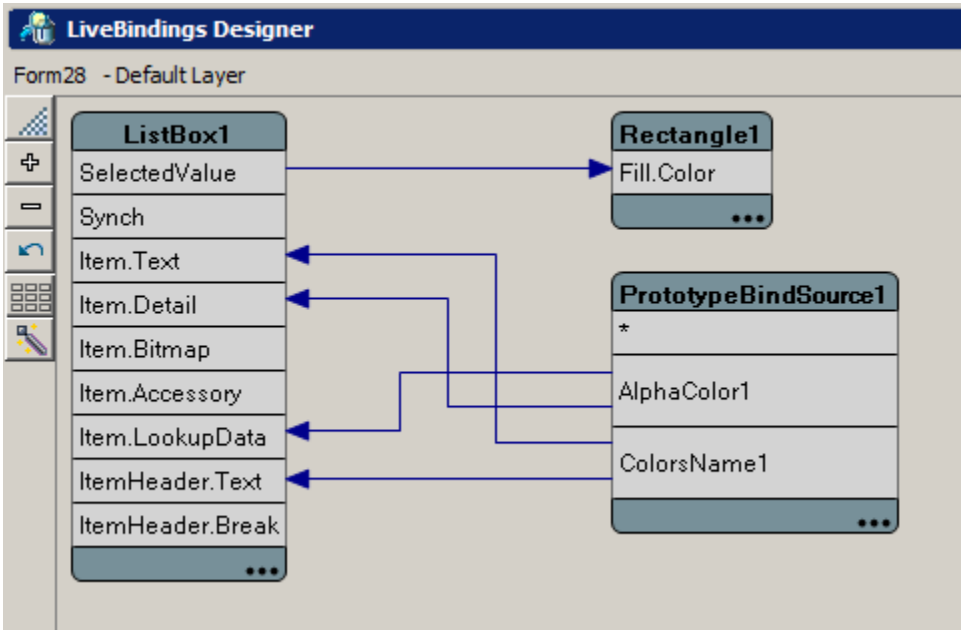
7. Click the ellipsis button [...] on the **Rectangle1** diagram block in the LiveBindings Designer, and in the dialog that opens, type **Fill.Color**. Select the **Fill.color** check box, then click **OK**:




8. Bind **Fill.Color** from **Rectangle1** to **SelectedValue** from **ListBox1**. This ensures the rectangle's fill color changes according to your list box item selection.

   **Note:** When attempting to bind **Fill.Color** to **SelectedValue**, a confirmation message box opens. In this message box, choose **Yes**.

After you complete the steps above, the LiveBindings diagram will be similar to the following image:



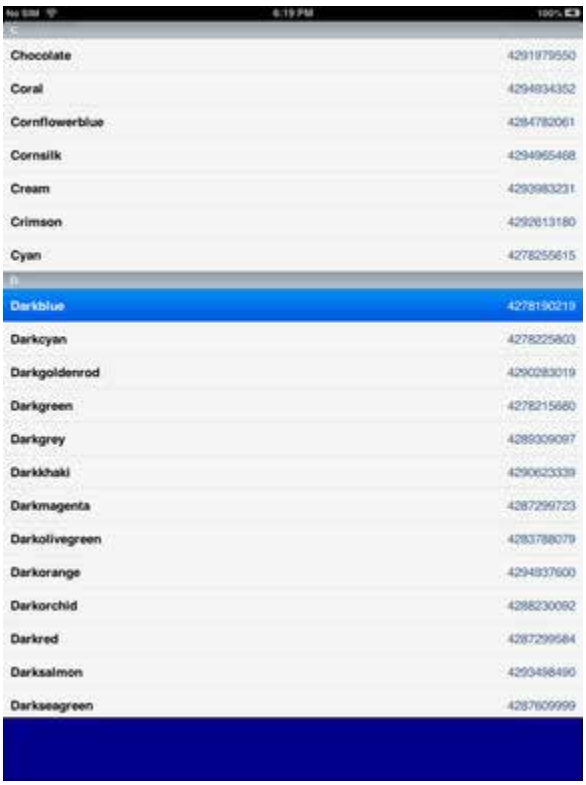

Your application is now ready to run on the mobile device.

# The Results

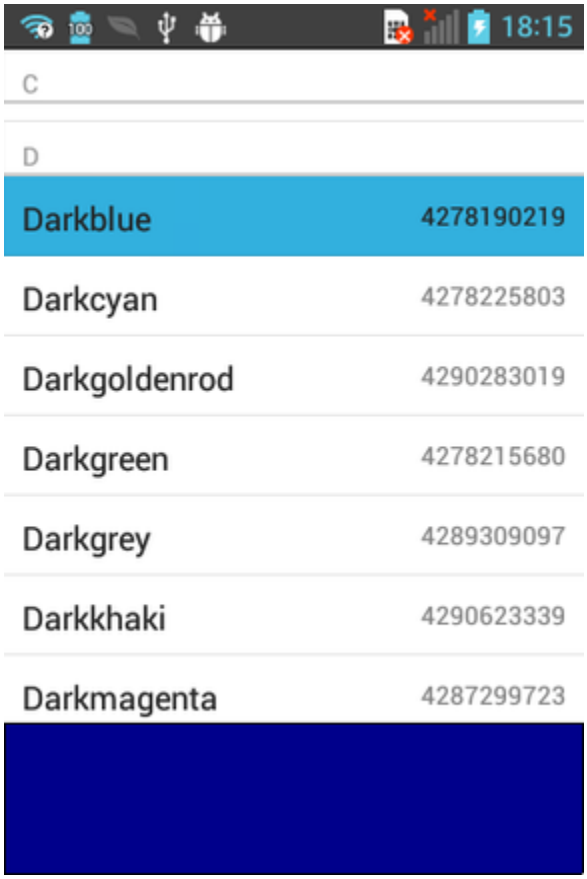
To run the application, press **F9** or choose Run > Run.

When you select a list box item, the rectangle's color changes accordingly:

**iOS (iPad)**




**Android (LG-E612)**




# See Also

- LiveBindings in RAD Studio
- LiveBindings Designer
- Using Custom Format and Parse Expressions in LiveBindings
- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)
- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)
- Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)

# Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android)

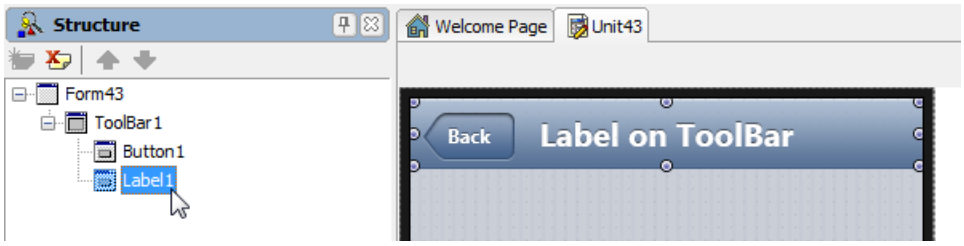
This tutorial describes a general strategy for using the **Master** view for different form factors (such as phone or tablet), without using different views for each form factor.

## Every FireMonkey Component Can Have an Owner, a Parent, and Children

First, every FireMonkey component has the idea of Owner, Parent, and Children. If you place a component on a form, the form becomes the owner and parent of the component.

If you add components (for example, a Button, Label, and others) to another component (for example, a ToolBar), the ToolBar is both parent and owner of the Button, Label, and others. You can see this parent-child relationship graphically represented in the tree view in the Structure View.

The Layout for a child is defined as a value relative to its parent. In the following picture, Label1 is the child of Toolbar1, and the Layout of Label1 is relative to Toolbar1.



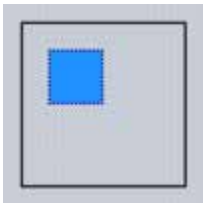

## Using Common Layout-Related Properties of a FireMonkey Component

### Using the Align Property

A control's **Align** property determines whether it is automatically repositioned and/or resized along its parent's four sides or center, both initially and as the parent is resized.

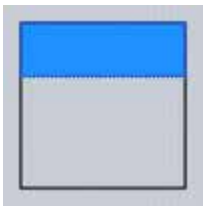
The default value for the **Align** property is **None**, which means that no automatic calculations are performed: the control stays where it is placed:

Align = None

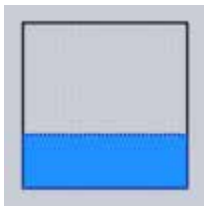


Typical values for the **Align** property are as follows (Dodgerblue indicates the area for the child):

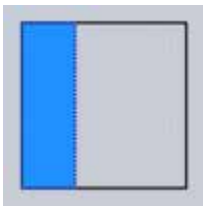
Top



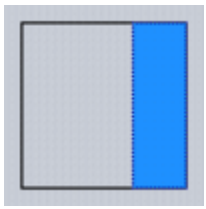
Bottom



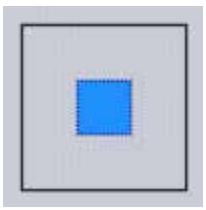
Left



Right



Center



Client

If you use an **Align** value of Top, Bottom, Left, or Right for one component, the **Align** properties for other components use the remaining area.

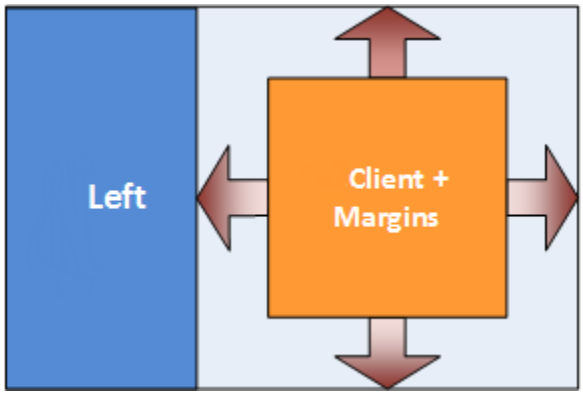
The size and shape of the remaining area (**Client**) also changes based on the orientation of the device, and based on the form factor (iPhone or iPad).

The following pictures show the layout for landscape (horizontal) and for portrait (vertical) when you have two (2) components that use **Top**, and one (1) component that uses **Client**.


## Using the Margins Property

**Margins** ensure separation between controls automatically positioned by a parent.

In the following picture, the right side of the component (Align = **Client**) uses the **Margins** property to ensure space around the component.

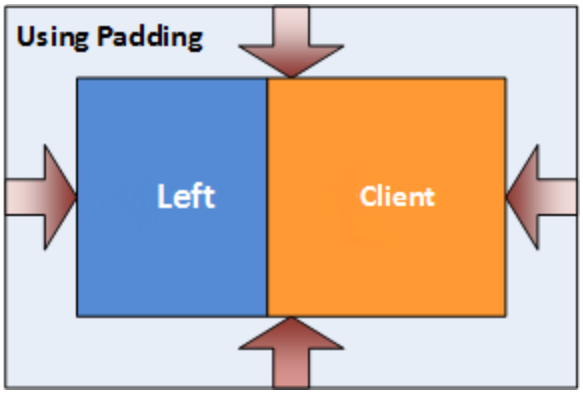



## Using the Padding Property

**Padding** sets aside space on the interior of the parent's content box. In the Object Inspector, you can set values (in pixels) for the Padding:

- Left
- Right
- Bottom
- Top

In the following picture, the parent component (which contains two regions) uses the **Padding** property to ensure space inside the parent component:




## Using the Anchors Property

Anchors are needed when a control must maintain its position at a certain distance from the edges of its parent, or must stretch while maintaining the original distance between its edges and the edges of its parent. Anchored controls 'stick' to the sides of containers and stretch, if so specified.

### Anchors Property for the Edit Control

If you have an **Edit** control on top of a **ToolBar**, you may want to keep a fixed distance between the right edge of the **Edit** Control and the edge of the form (**ToolBar**). Anchors enable you to specify that a control is to remain fixed in relation to the sides of its parent.

If you want the **Edit** control to maintain the same relative position in relation to the ToolBar (its parent), you can set the **Anchors** property to **akLeft, akTop, akRight**. When the ToolBar is resized, the Edit control is resized according to the Anchors settings:

**iOS**


**Android**

### Anchors Property for Button Control

If you have a Button control at the right end of the ToolBar, you may want to keep the same distance between the **right** edge of the Button control and the edge of the Form. However, you might not want to maintain the same distance between the **left** edge of the Button control and the **left** edge of the Form. In this case, you can set the **Anchors** property to **akTop, akRight** (de-select **akLeft**), so that the Button control maintains the same distances with the ToolBar (parent) for **Top** and **Right**.

iOS:

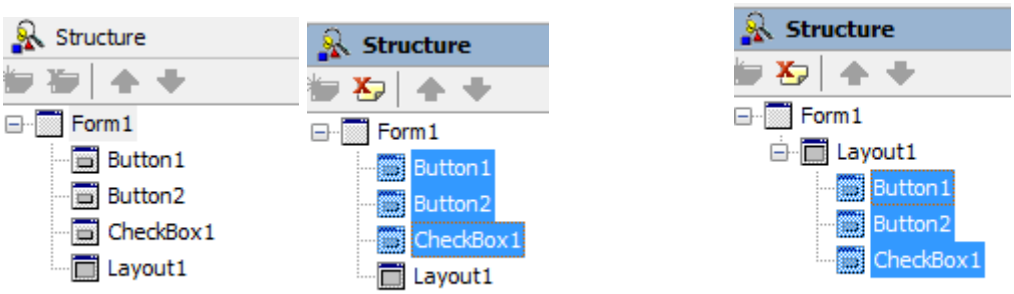

Android:

# Using the TLayout Component

TLayout, a component that is not visible at run time, can be used to group its child controls to be manipulated as a whole. For example, you can set the visibility of a group of controls at one time by setting the Visible property of the layout. TLayout does not automatically set any of the properties of its children.

To make selected controls children of TLayout, use the Structure View.
Highlight the controls you want to move. Then drag the group of controls over the control that should be the parent, and drop the controls there. In the Structure View, the group of controls are now children of the new parent:

**1. Initial State** **2. Highlight the Controls to Move** **3. Drag onto Parent**




You can use **Align**, **Padding**, **Margins**, **Anchors**, and other properties of TLayout to define the layout for a specific area. You can use the TLayout component just like the **DIV** tag in HTML.

# See Also

- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)
- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)
- Mobile Tutorial: Using Location Sensors (iOS and Android)

- Tutorial: Using FireMonkey Layouts
- FireMonkey Layouts Strategies
- Arranging FireMonkey Controls
- Gestures in FireMonkey

# Mobile Tutorial: Taking and Sharing a Picture, and Sharing Text (iOS and Android)

Before starting this tutorial, you should read and perform the following tutorial:

- Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)

This tutorial covers the following typical tasks for using pictures and sharing text with your applications in mobile platforms.

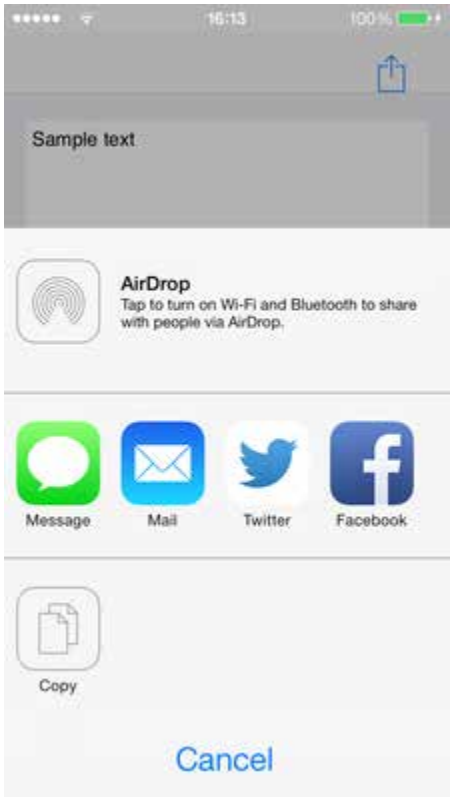
### On iOS Devices:

**Taking a picture with the device camera**


**Using a picture from the device Photo Library**

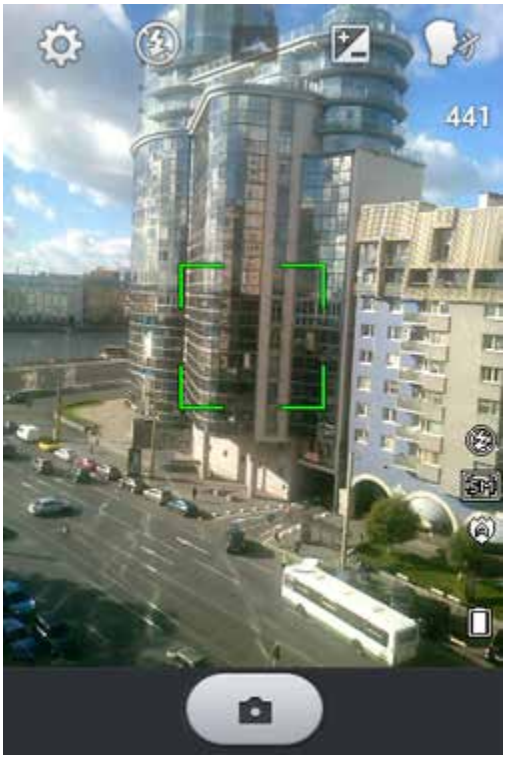
Sharing or printing a picture


Sharing text



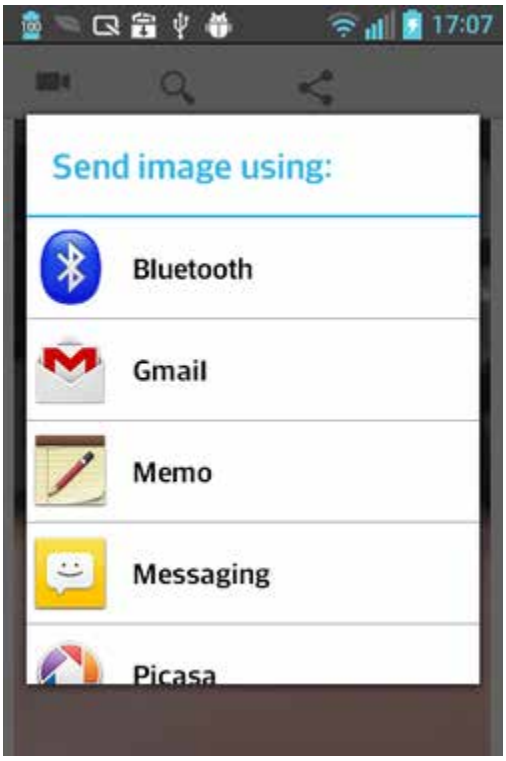

On Android Devices:

Taking a picture with the device camera

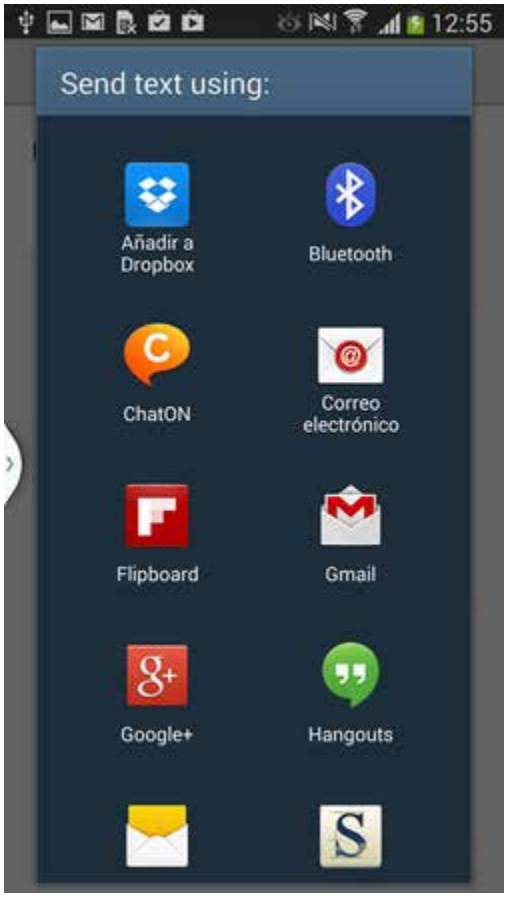



Using a picture from the device Photo Library

Sharing or printing a picture




Sharing text




# Topics

- Taking and Sharing Pictures and Text Using Action Lists
- Taking Pictures Using FireMonkey Interfaces

# See Also

- Mobile Tutorial: Using Location Sensors (iOS and Android)
- Mobile Tutorial: Using Notifications (iOS and Android)
- Mobile Code Snippets
- FireMonkey Actions
- FMX.StdCtrls.TButton
- FMX.Objects.TImage

- FMX.MediaLibrary
- FMX.PhotoEditorDemo Sample
- FMX.CameraComponent Sample
- http://appleinsider.com/articles/12/02/16/share_sheets

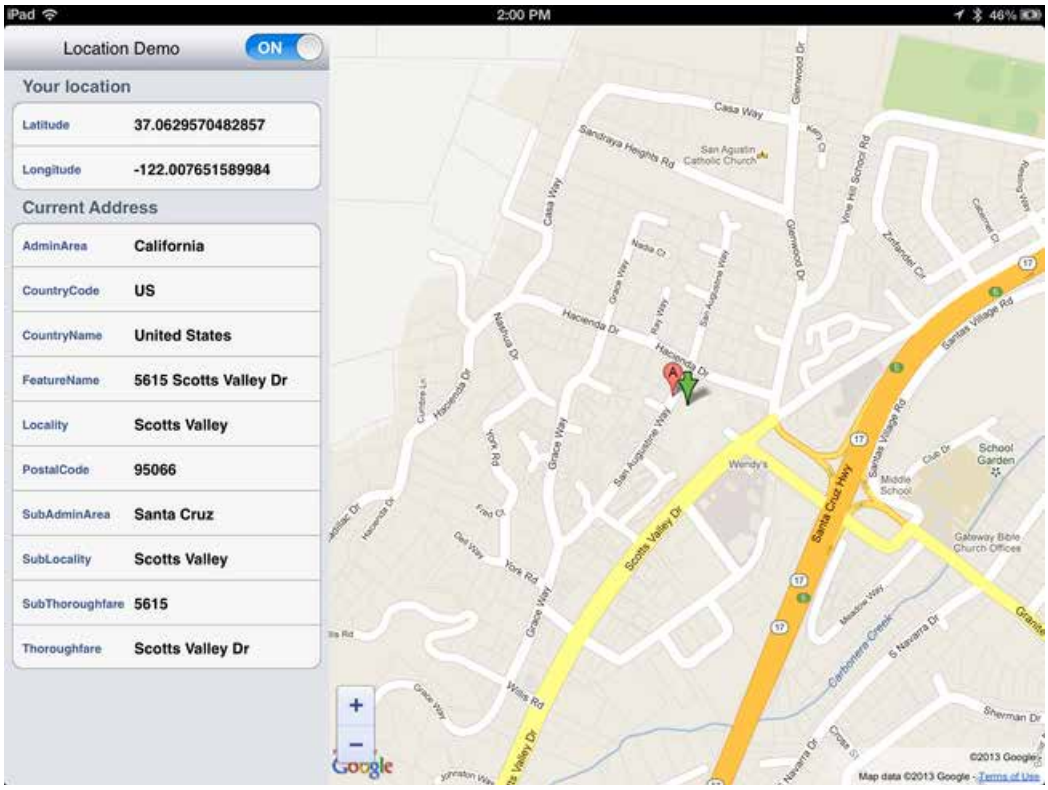
# Mobile Tutorial: Using Location Sensors (iOS and Android)

Before starting this tutorial, you should read and perform the following tutorial sessions:

- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)
- Mobile Tutorial: Using the Web Browser Component (iOS and Android)
- Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android)

**Note:** On Android devices, TLocationSensor requires specific Uses Permissions to be set, specifically **Access coarse location** and **Access fine location**.

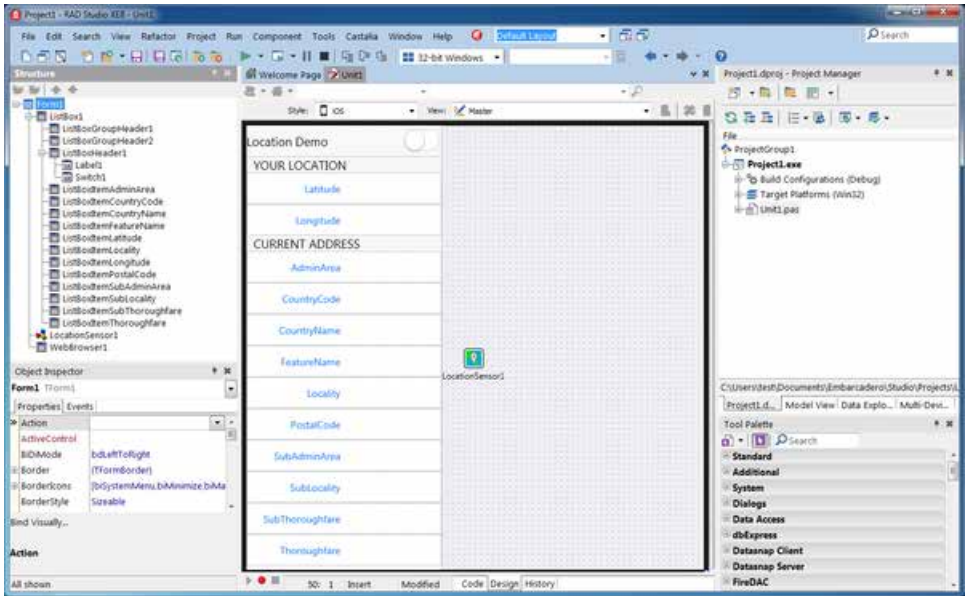
This tutorial describes the basic steps to locate your mobile device (using latitude and longitude), and to use **Reverse Geocoding** to convert to a readable address, such as in the following picture:



# Design the User Interface

This demo application is designed with two major components: a TListBox (on the left-hand side) and a TWebBrowser.

> **Note**: Before proceeding with this scenario, in the Project Manager, set the active target platform to `iOS Device` or `Android`. Otherwise, you cannot add the TWebBrowser component.




Drop a TListBox and a TWebBrowser component on the form.

- In the TListBox, set the **Align** property to `Left` to reserve the left side of the UI. Then create the following subcomponents under the ListBox:
  - A TListBoxHeader component with the following sub-components:
    - A TLabel component with the text "Location Demo"
    - A TSwitch (Switch1) component to select on/off of TLocationSensor
  - A TListBoxGroupHeader with the text "Your Location"
  - A TListBoxItem with the name "ListBoxItemLatitude" and "Latitude" as text
  - A TListBoxItem with the name "ListBoxItemLongitude" and "Longitude" as text
  - A TListBoxGroupHeader with the text "Current Address"
  - A TListBoxItem with the name "ListBoxItemAdminArea" and "AdminArea" as text
  - A TListBoxItem with the name "ListBoxItemCountryCode" and "CountryCode" as text

        - A TListBoxItem with the name "ListBoxItemCountryName" and "CountryName" as text
        - A TListBoxItem with the name "ListBoxItemFeatureName" and "FeatureName" as text
        - A TListBoxItem with the name "ListBoxItemLocality" and "Locality" as text
        - A TListBoxItem with the name "ListBoxItemPostalCode" and "PostalCode" as text
        - A TListBoxItem with the name "ListBoxItemSubAdminArea" and "SubAdminArea" as text
        - A TListBoxItem with the name "ListBoxItemSubLocality" and "SubLocality" as text
        - A TListBoxItem with the name "ListBoxItemSubThoroughfare" and "SubThoroughfare" as text
        - A TListBoxItem with the name "ListBoxItemThoroughfare" and "Thoroughfare" as text
    - A TWebBrowser component (WebBrowser1) to show the Web Page (Google Maps). Set the **Align** property to `Client`.

After you create these components, select all **TListBoxItem** items and select **listboxitemleftdetail** in the StyleLookup property. This allows TListBoxItem to show both a label and detailed text.

# The Location Sensor

The location sensor is wrapped by the **TLocationSensor** component.

**TLocationSensor** fires an OnLocationChanged event when the device detects movement. You can adjust the sensitivity of **TLocationSensor** using the Distance and Accuracy properties.

- The **Distance** property specifies the minimum distance (in meters) by which the device must move in order to make the location sensor relocate the device and return new location information. For example, if you set **Distance** to "10", **TLocationSensor** fires an OnLocationChanged event when you move "10 meters".
- The **Accuracy** property represents the level of precision (in meters) by which the sensor locates the device geographically, relative to the geographical point at which the device is actually located.

**Tip**: You should specify the lowest accuracy that works for your application; the higher the accuracy, the more time and power that the sensor requires to determine the location. The recommended values: **Distance**=0; **Accuracy=0**.

# Read Location Information (Latitude, Longitude) from the LocationSensor Component

The TLocationSensor component needs to be activated for use. You can turn on/off **TLocationSensor** based on your input, such as a **TSwitch** component, or other Application events.

1. Place a TLocationSensor component from the Tool Palette.
2. On the Form Designer, select the TSwitch component.
3. In the Object Inspector, in the **Events** tab double-click **OnSwitch** event.
4. Add the following code to the **OnSwitch** event handler:

**Delphi**:

```
procedure TForm1.Switch1Switch(Sender: TObject);
begin
  LocationSensor1.Active := Switch1.IsChecked;
end;
```

**C++**:

```
void __fastcall TForm1::Switch1Switch(TObject *Sender)
{
        LocationSensor1->Active = Switch1->IsChecked;
}
```

As discussed earlier, **TLocationSensor** fires an OnLocationChanged event when you move the mobile device. You can show the current location (Latitude and Longitude) using parameters with an event handler.

1. On the Form Designer, select the TLocationSensor.
2. In the Object Inspector, in the **Events** tab double-click **OnLocationChange** event.
3. Add the following code to the **OnLocationChange** event handler:

**Delphi**:

```
procedure TForm1.LocationSensor1LocationChanged(Sender: TObject;
  const OldLocation, NewLocation: TLocationCoord2D);
var
  LDecSeparator: String;
begin
  LDecSeparator := FormatSettings.DecimalSeparator;
  FormatSettings.DecimalSeparator := '.';
  // Show current location
  ListBoxItemLatitude.ItemData.Detail  := Format('%2.6f', [NewLocation.Latitude]);
  ListBoxItemLongitude.ItemData.Detail := Format('%2.6f', [NewLocation.Longitude]);
```

```
end;
```

**C++:**

```
void __fastcall TForm1::LocationSensor1LocationChanged(TObject *Sender, const
TLocationCoord2D &OldLocation,
                  const TLocationCoord2D &NewLocation)
{
        char LDecSeparator = FormatSettings.DecimalSeparator;
        FormatSettings.DecimalSeparator = '.';
        // Show current location
        ListBoxItemLatitude->ItemData->Detail = ListBoxItemLatitude->ItemData-
>Detail.sprintf(L"%2.6f", NewLocation.Latitude);
        ListBoxItemLongitude->ItemData->Detail = ListBoxItemLongitude->ItemData-
>Detail.sprintf(L"%2.6f", NewLocation.Longitude);
}
```

# Show the Current Location Using Google Maps via a TWebBrowser Component

As discussed in the Mobile Tutorial: Using the Web Browser Component (iOS and Android), the TWebBrowser component wraps a Web browser for mobile platforms.

You can call Google Maps from the TWebBrowser component with the following URL parameters:

```
https://maps.google.com/maps?q=(Latitude-value),(Longitude-value)
```

So you can add this URL to your previously created event handler **OnLocationChanged** as follows:

**Delphi:**

```
procedure TForm1.LocationSensor1LocationChanged(Sender: TObject;
  const OldLocation, NewLocation: TLocationCoord2D);
var
  URLString: String;
begin
  // code for previous step goes here

  // Show Map using Google Maps
  URLString := Format(
    'https://maps.google.com/maps?q=%s,%s',
      [Format('%2.6f', [NewLocation.Latitude]), Format('%2.6f',
[NewLocation.Longitude])]);
  WebBrowser1.Navigate(URLString);
end;
```

**C++:**

```
void __fastcall TForm1::LocationSensor1LocationChanged(TObject *Sender, const
TLocationCoord2D &OldLocation,
                  const TLocationCoord2D &NewLocation)
{
        // code for previous step goes here

        // Show Map using Google Maps
        String LLongitude = FloatToStr(NewLocation.Longitude, FormatSettings);
        String URLString = "";
        URLString = URLString.sprintf(L"https://maps.google.com/maps?q=%2.6f,%2.6f",
                NewLocation.Latitude, NewLocation.Longitude);

        FormatSettings.DecimalSeparator = LDecSeparator;
        WebBrowser1->Navigate(URLString);
}
```

# Use Reverse Geocoding

TGeocoder is an object which wraps the Geocoding (or Reverse Geocoding) service.

**Geocoding** is the process of transforming geographic data, such as the address and zip code, into geographic coordinates. Reverse geocoding is the process of transforming geographical coordinates into other geographical data, such as the address.

In this case, we use **TGeocoder** to "Reverse Geocode" our location (in Latitude and Longitude) to readable address information.

Here is the basic sequence of actions with **TGeocoder**:

1. Create an instance of **TGeocoder**.
2. Define an event OnGeocodeReverse so that you can receive the event later.
3. Set data to execute "Reverse Geocoding".
4. **TGeocoder** accesses the service on the network to resolve the address information.
5. **TGeocoder** fires an OnGeocodeReverse event.
6. Your iOS App receives the address information through the parameter on the OnGeocodeReverse event and updates the user interface.

**Note:** As **TGeocoder** is not a component (this is just a class), you need to define these steps through your code (you cannot drop a component, nor assign an event handler through the Object Inspector).

First, define a new field "FGeocoder" in the private section of the form. You can also define an "OnGeocodeReverseEvent procedure" as in the following code snippets.

**Delphi:**

```
type
  TForm1 = class(TForm)
    // IDE defines visible (or non-visual) components here automatically
```

```
  private
    { Private declarations }
    FGeocoder: TGeocoder;
    procedure OnGeocodeReverseEvent(const Address: TCivicAddress);
  public
    { Public declarations }
  end;
```

**C++:**

**Note:** Place this code snippet in the header file (.h)

```
class TForm1 : public TForm
{
        // IDE defines visible (or non-visual) components here automatically
private:        // User declarations
        TGeocoder *FGeocoder;
        void __fastcall OnGeocodeReverseEvent(TCivicAddress* const Address);
public:         // User declarations
        __fastcall TForm1(TComponent* Owner);
};
```

Now you can create an instance of **TGeocoder** and set it up with data with the following Delphi or C++ code.

TGeocoder.Current gives the type of class that actually implements the Geocoding Service. The code in "`TGeocoder.Current.Create`" calls the constructor (Create) for the specified type, and saves it to the **FGeocoder** field. You also need to specify an event handler, which is fired when TGeocoder completes Reverse Geocoding. Assign **OnGeocodeReverseEvent** (which you just defined in the previous step) to **FGeocoder.OnGeocodeReverse**.

Finally, if you successfully created an instance of **TGeocoder**, and **TGeocoder** is not running, call TGeocoder.GeocodeReverse with location information. After **TGeocoder** receives data, the OnGeocodeReverseEvent event is fired.

**Delphi:**

```
procedure TForm1.LocationSensor1LocationChanged(Sender: TObject;
  const OldLocation, NewLocation: TLocationCoord2D);
begin
  // code for previous steps goes here
try
    // Setup an instance of TGeocoder
    if not Assigned(FGeocoder) then
    begin
      if Assigned(TGeocoder.Current) then
        FGeocoder := TGeocoder.Current.Create;
      if Assigned(FGeocoder) then
        FGeocoder.OnGeocodeReverse := OnGeocodeReverseEvent;
    end;

    // Translate location to address
    if Assigned(FGeocoder) and not FGeocoder.Geocoding then
```

```
      FGeocoder.GeocodeReverse(NewLocation);
  except
    ListBoxGroupHeader1.Text := 'Geocoder service error';
  end;
end;
```

**C++:**

```
void __fastcall TForm1::LocationSensor1LocationChanged(TObject *Sender, const
TLocationCoord2D &OldLocation,
                  const TLocationCoord2D &NewLocation)
{
        // code for previous steps goes here

                // Setup an instance of TGeocoder
        try {
        if (FGeocoder == NULL) {
            if (TGeocoder::Current != NULL) {
                FGeocoder = (TGeocoder*)new TGeocoderClass(TGeocoder::Current);
            }
            if (FGeocoder != NULL) {
                FGeocoder->OnGeocodeReverse = OnGeocodeReverseEvent;
            }
        }
        // Translate location to address

        if ((FGeocoder != NULL) && (!(FGeocoder->Geocoding()))) {
            FGeocoder->GeocodeReverse(NewLocation);
        }
    }
    catch (...) {
        ListBoxGroupHeader1->Text = "Geocoder service error";
    }

}
```

# Show a Readable Address in the ListBox Component

As described earlier, after Reverse Geocoding is completed, an OnGeocodeReverseEvent is fired.

Next, assign properties in the TCivicAddress address parameter to show readable address information in the list box fields:

**Delphi:**

```
procedure TForm1.OnGeocodeReverseEvent(const Address: TCivicAddress);
begin
  ListBoxItemAdminArea.ItemData.Detail       := Address.AdminArea;
  ListBoxItemCountryCode.ItemData.Detail     := Address.CountryCode;
  ListBoxItemCountryName.ItemData.Detail     := Address.CountryName;
  ListBoxItemFeatureName.ItemData.Detail     := Address.FeatureName;
  ListBoxItemLocality.ItemData.Detail        := Address.Locality;
  ListBoxItemPostalCode.ItemData.Detail      := Address.PostalCode;
  ListBoxItemSubAdminArea.ItemData.Detail    := Address.SubAdminArea;
```

```
  ListBoxItemSubLocality.ItemData.Detail     := Address.SubLocality;
  ListBoxItemSubThoroughfare.ItemData.Detail := Address.SubThoroughfare;
  ListBoxItemThoroughfare.ItemData.Detail    := Address.Thoroughfare;
end;
```

**C++:**

```
void __fastcall TForm1::OnGeocodeReverseEvent(TCivicAddress* const Address)
{
          if (Address != NULL){
                ListBoxItemAdminArea->ItemData->Detail       = Address->AdminArea;
                ListBoxItemCountryCode->ItemData->Detail     = Address->CountryCode;
                ListBoxItemCountryName->ItemData->Detail     = Address->CountryName;
                ListBoxItemFeatureName->ItemData->Detail     = Address->FeatureName;
                ListBoxItemLocality->ItemData->Detail        = Address->Locality;
                ListBoxItemPostalCode->ItemData->Detail      = Address->PostalCode;
                ListBoxItemSubAdminArea->ItemData->Detail    = Address->SubAdminArea;
                ListBoxItemSubLocality->ItemData->Detail     = Address->SubLocality;
                ListBoxItemSubThoroughfare->ItemData->Detail = Address-
>SubThoroughfare;
                ListBoxItemThoroughfare->ItemData->Detail    = Address->Thoroughfare;
        }
}
```

# Describing Why Your Application Needs the User Location

Before you deploy your final application you should select **Project > Options > Version Info**, with iOS Device as target, and update the values of `NSLocationAlwaysUsageDescription` and `NSLocationWhenInUseUsageDescription` with a message that explains why your application is asking for the user location. Your users see this message when your application asks them to authorize iOS to provide the location of the iOS device to your application.

# See Also

- Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android)
- Mobile Tutorial: Using Notifications (iOS and Android)
- Mobile Tutorial: Using Remote Notifications (iOS and Android)
- System.Sensors.TGeocoder
- System.Sensors.Components.TLocationSensor
- Mobile Code Snippets: Notifications

## Samples

- Location sample

- VCL Sensors sample

# Mobile Tutorial: Using Notifications (iOS and Android)

This tutorial describes the basic steps to use notifications on your mobile device; for further details, see using notifications.

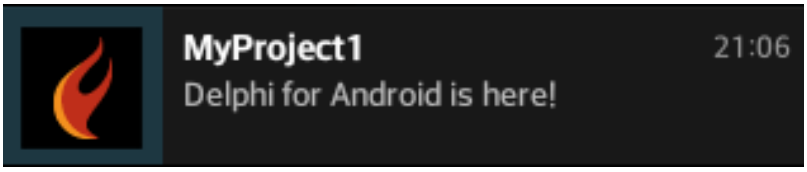
## Three Basic Notification or Alert Styles

When users set notifications for apps on their mobile devices, notifications can be delivered from apps in the three basic styles shown here. The banner appears briefly, but the alert dialog box requires dismissal by the user.

### Notification Banner on Mobile Devices

**iOS**


**Android**

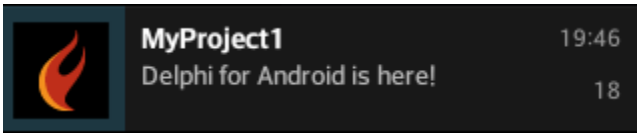


## Alert Dialogs: iOS Badge Number and Android Notification Number

**iOS Badge Number**


iPad

**Android Notification Number**



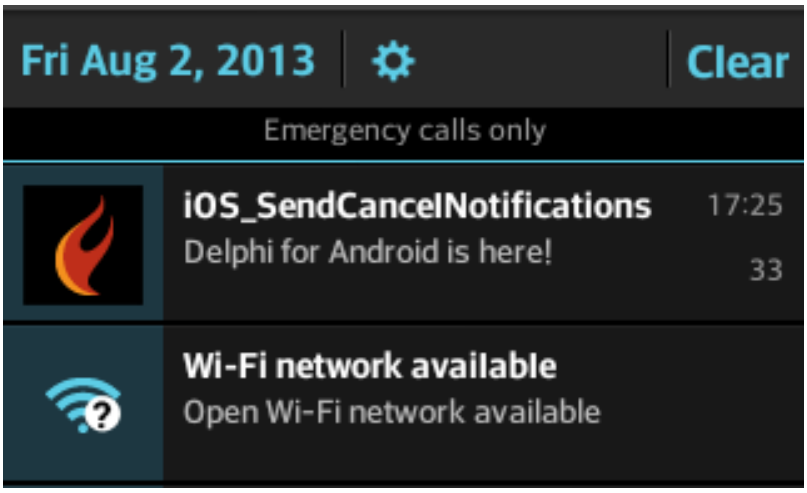

Android

# Notification Center on Mobile Devices

The following images show the notification center on an iPad (Notification Center) and Android (notification drawer), where the user can pull down the list of all recent notifications.

**iOS**

**Android**




# Access the Notification Service

RAD Studio provides the TNotificationCenter component that allows you to easily access the Notification Service.

To access the notification service, do the following:

1. Create a new mobile application:
   - For **Delphi**: **File > New > Multi-Device Application - Delphi**
   - For **C++**: **File > New > Multi-Device Application - C++Builder**
2. Select the TNotificationCenter component in the Tool Palette, and drop it on the Form Designer.
3. Check that the following unit has been automatically added to the project:
   - For Delphi applications, add the following unit to the uses clause if it is not present:

```
uses
  System.Notification;
```

   - For C++ applications, add the following **include** operator to the project header file (.h file):

```
#include <System.Notification.hpp>
```

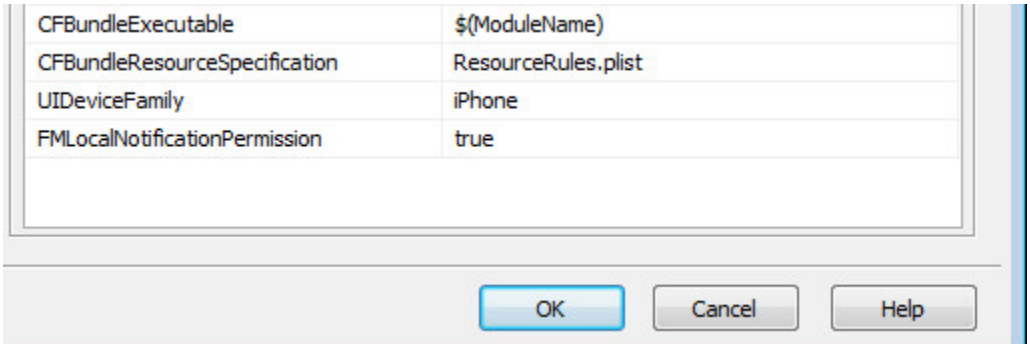
The System.Notification.TNotificationCenter.CreateNotification method allows you to create an instance of the TNotification class object.

# Add FMLocalNotificationPermission (iOS)

In order to use notifications on iOS 8+ devices, you need to ensure that the **FMLocalNotificationPermission** Key/Value pair is enabled on the Version Info page of Project Options.

1. Choose **Project > Options > Version Info**.
2. In the **Target** field, select **Debug configuration - iOS device - 32-bit platform**.
3. Set the **Value** field for **FMLocalNotificationPermission** to **true**.

The **FMLocalNotificationPermission** setting enables local notifications on iOS 8+ devices:




# Set the Icon Badge Number and Notification Number from Code

TNotification.Number defines the **Icon Badge Number** (for iOS devices) and the **notification number** (for Android devices).

To set the iOS icon badge number (for Delphi or C++) or the Android notification number (only for Delphi apps), you need to implement the corresponding method.

1. Drop a TButton to the form.
2. In the Object Inspector, change the Name property to **SetNumber**.
3. Create the OnClick event of the **SetNumber** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **SetNumber** button by adding the following code:
   - For **Delphi:**

```
procedure TForm1.SetNumberClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
  // Create an instance of TNotification
  MyNotification := NotificationCenter1.CreateNotification;
  try
```

```
      // --- your code goes here ---
      // Set the icon or notification number
      MyNotification.Number :=18;
      // Set the alert message
      MyNotification.AlertBody := 'Delphi for your mobile device is here!';
      // Note: You must send the notification to the notification center for
the Icon Badge Number to be displayed.
      NotificationCenter1.PresentNotification(MyNotification);
  finally
    MyNotification.DisposeOf;
  end;
end;
```

- For **C++:**

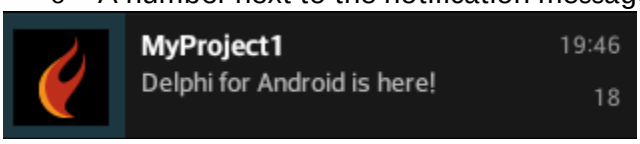
```
void __fastcall TForm1::SetNumberClick(TObject *Sender)
{
        if (NotificationCenter1->Supported()) {
                TNotification *myNotification = NotificationCenter1-
>CreateNotification();
                __try {
                        myNotification->Number = 18;
                        myNotification->AlertBody = "C++ for your mobile device
is here!";
                        NotificationCenter1-
>PresentNotification(myNotification);
                }
                __finally {
                        myNotification->DisposeOf();
                }
        }

}
```

After running your application (press F9) and clicking the **SetNumber** button, you can see the following:

- A badge on the application icon (on the iOS Home screen):


- A number next to the notification message in the notification center (on Android):




Android

# Schedule Notification

You can also schedule Notification Messages using the ScheduleNotification method that the TNotificationCenter class inherits from TCustomNotificationCenter.

To show a Notification Message, you need to create an instance of the TNotification class, and then define the Name, AlertBody, and FireDate fields.

1. Drop a new TButton to the form.
2. In the Object Inspector, change the Name property to **ScheduleNotification**.
3. Create the OnClick event of the **ScheduleNotification** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **ScheduleNotification** button by adding the following code:
    - For **Delphi:**

```
procedure TForm1.ScheduleNotificationClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
  MyNotification := NotificationCenter1.CreateNotification;
  try
    MyNotification.Name := 'MyNotification';
    MyNotification.AlertBody := 'Delphi for your mobile device is here!';
    // Fired in 10 seconds
    MyNotification.FireDate := Now + EncodeTime(0, 0, 10, 0);
    // Send notification to the notification center
    NotificationCenter1.ScheduleNotification(MyNotification);
  finally
    MyNotification.DisposeOf;
  end;
end;
```

    - For **C++**:

```
void __fastcall TForm1::ScheduleNotificationClick(TObject *Sender)
{
        if (NotificationCenter1->Supported()) {
                TNotification *myNotification = NotificationCenter1-
>CreateNotification();
                __try {
                        myNotification->Name = "MyNotification";
                        myNotification->AlertBody = "C++ for your mobile device
is here!";
                        // Fire in 10 seconds
                        myNotification->FireDate = Now() + EncodeTime(0, 0, 10,
0);
                        // Send notification to the notification center
                        NotificationCenter1-
>ScheduleNotification(myNotification);
                }
```

```
                __finally {
                        myNotification->DisposeOf();
                }

        }
}
```

After running your application (press F9) and clicking the **ScheduleNotification** button, you can see the Notification Message (AlertBody) at the top of your device Home Screen. For iOS devices, this message is similar to the following screen:


# Repeat a Notification Message

You can also repeat a Notification Message using the RepeatInterval property of the **TNotification** object.

To repeat a Notification Message, you need to create an instance of the TNotification class, and then define the Name, AlertBody, and FireDate fields.

Also you need to use the ScheduleNotification method and set the RepeatInterval property. In the following code, the repeated interval is set to a minute.

1. Drop a new TButton to the form.
2. In the Object Inspector, change the Name property to **RepeatedNotification**.
3. Create the OnClick event of the **RepeatedNotification** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **RepeatedNotification** button by adding the following code:
    - For **Delphi:**

```
procedure TForm1.RepeatedNotificationClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
  MyNotification := NotificationCenter1.CreateNotification;
  try
    MyNotification.Title := 'MyNotification';
    MyNotification.AlertBody := 'Repeating notification each minute!';
    //Fired in 10 seconds
    MyNotification.FireDate := Now + EncodeTime(0, 0, 10, 0);
    //Repeated each minute
    MyNotification.RepeatInterval := TRepeatInterval.Minute;
    // Send notification to the notification center
    NotificationCenter1.ScheduleNotification(MyNotification);
  finally
```

```
    MyNotification.Free;
  end;
end;
```

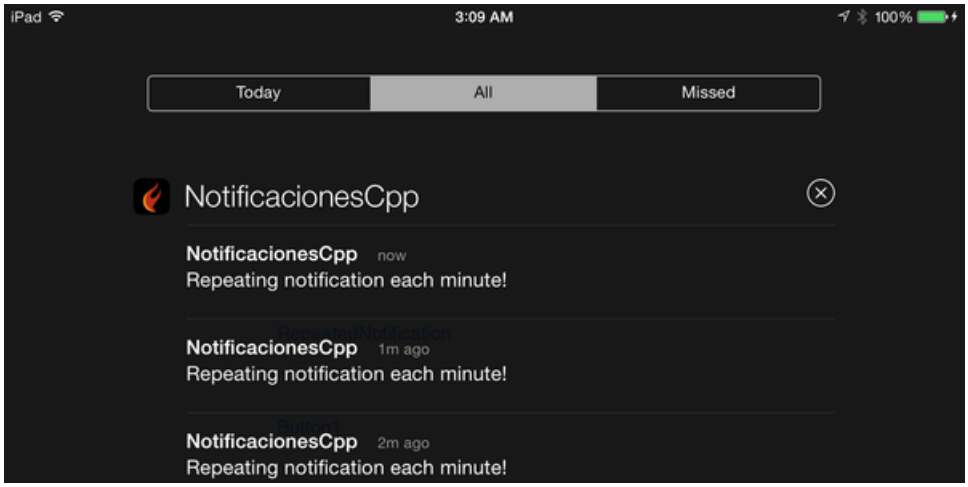
- For **C++**:

```
void __fastcall TForm1::RepeatedNotificationClick(TObject *Sender) {
    if (NotificationCenter1->Supported()) {
        TNotification *myNotification =
            NotificationCenter1->CreateNotification();
        __try {
            myNotification->Name = "MyNotification";
            myNotification->AlertBody = "Repeating notification each minute!";
            // Fire in 10 seconds
            myNotification->FireDate = Now() + EncodeTime(0, 0, 10, 0);
            // Repeated each minute
            myNotification->RepeatInterval = TRepeatInterval::Minute;
            // Send notification to the notification center
            NotificationCenter1->ScheduleNotification(myNotification);
        }
        __finally {
            myNotification->Free();
        }

    }

}
```

After running your application (press F9) and clicking the **RepeatedNotification** button, you can see the Notification Message (AlertBody) at the top of your device Home Screen. For iOS devices, this message is similar to the following screen:




# Update or Cancel a Scheduled or Repeated Notification Message

Each Scheduled Notification Message is identified through the **Name** property of the **TNotification** object.

**Note:** For the iOS platform, CancelNotification and ScheduleNotification can cancel or update those notifications that have not been presented yet.

To update a scheduled notification, simply call the ScheduleNotification method again with an instance of **TNotification** that has the same name (Name property).

To cancel a scheduled notification, you can simply call the CancelNotification method with the identifier you used.

1. Drop a new TButton to the form.
2. In the Object Inspector, change the Name property to **CancelNotification**.
3. Create the OnClick event of the **CancelNotification** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **CancelNotification** button by adding the following code:
   - For **Delphi:**

```
procedure TForm1.CancelNotificationClick(Sender: TObject);

begin
   NotificationCenter1.CancelNotification('MyNotification');
end;
```

   - For **C++**:

```
  void __fastcall TForm1::CancelNotificationClick(TObject *Sender)
{
   NotificationCenter1->CancelNotification("MyNotification");
}
```

# Present the Notification Message Immediately

You can also show the notification message immediately through the PresentNotification function.

To show a notification message, you need to create an instance of the TNotification class, and then define the Name and AlertBody fields.

1. Drop a new TButton to the form.
2. In the Object Inspector, change the Name property to **PresentNotification**.
3. Create the OnClick event of the **PresentNotification** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **PresentNotification** button by adding the following code:

§ For **Delphi:**

```
procedure TForm1.PresentNotificationClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
  MyNotification := NotificationCenter1.CreateNotification;
  try
    MyNotification.Name := 'MyNotification';
    MyNotification.AlertBody := 'Delphi for your mobile device is here!';
    // Set Icon Badge Number (for iOS) or message number (for Android) as well
    MyNotification.Number := 18;
    MyNotification.EnableSound := False;
    // Send message to the notification center
    NotificationCenter1.PresentNotification(MyNotification);
  finally
    MyNotification.DisposeOf;
  end;
end;
```

§ For **C++:**

```
void __fastcall TForm1::PresentNotificationClick(TObject *Sender)
{
  if (NotificationCenter1->Supported()) {
           TNotification *myNotification = NotificationCenter1-
>CreateNotification();
           __try {
                    myNotification->Name = "MyNotification";
                    myNotification->AlertBody = "C++ for your mobile device is
here!";
                    // Set Icon Badge Number (for iOS) or message number (for
Android) as well
                    myNotification->Number = 18;
                    myNotification->EnableSound = False;
                    // Send notification to the notification center
                    NotificationCenter1->PresentNotification(myNotification);
          }
          __finally {
                    myNotification->DisposeOf();
          }
 }

}
```

# Customizing the Notification Sound

You can use a customized sound for the notification message immediately through the **SoundName** property of the **TNotification** object.

You need to create an instance of the TNotification class, define the EnableSound and the SoundName properties. You also need to indicate the full path to the file sound and the file extension.

1. Drop a new TButton to the form.

2. In the Object Inspector, change the Name property to **SoundNotification**.
3. Create the OnClick event of the **SoundNotification** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **SoundNotification** button by adding the following code:
    - For **Delphi:**

```
procedure TForm1.SoundNotificationClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
  MyNotification := NotificationCenter1.CreateNotification;
  try
    MyNotification.Name := 'MyNotification';
    MyNotification.AlertBody := 'Delphi for your mobile device is here!';
    MyNotification.EnableSound := True;
    MyNotification.SoundName := GetSoundName;
    MyNotification.FireDate := Now + EncodeTime(0, 0, 10, 0);
    // Send message to the notification center
    NotificationCenter1.ScheduleNotification(MyNotification);
  finally
    MyNotification.Free;
  end;
end;
```

    - For **C++:**
        - In the .cpp file, add the following code:

```
void __fastcall TForm1::SoundNotificationClick(TObject *Sender)
{
TNotification *myNotification = NotificationCenter1->CreateNotification();
        __try {
            myNotification->Name = "MyNotification";
            myNotification->AlertBody = "C++ for your mobile device is here!";
            myNotification->EnableSound = true;
            myNotification->SoundName = GetSoundName();
            myNotification->FireDate = Now() + EncodeTime(0, 0, 10, 0);
            // Send notification to the notification center
           NotificationCenter1->ScheduleNotification(myNotification);
        }
        __finally {
            myNotification->Free();
        }
}
```

5. Declare the **GetSoundName** function.
    - For **Delphi:**
        - Set the GetSoundName declaration in the private section of the unit.

- Add the System.IOUtils to the project uses clauses.

```
uses
  System.IOUtils;
private
    function GetSoundName: string;
```

- For **C++**:
  - In the header file (.h file), add the following declaration:
  - Set the GetSoundName declaration in the private section of the unit.
  - Include the `System.IOUtils.hpp` to the project uses clauses.

```
#include <System.IOUtils.hpp>
private:    // User declarations
     __fastcall  UnicodeString GetSoundName ();
```

6. Implement the **GetSoundName** function.

**Note:** Depending on the target operating system, the sound data should be a different file extension.

- For **Delphi**:

```
{$R *.fmx}
function TForm1.GetSoundName: string;
begin
{$IFDEF IOS}
  Result := 'myiOSSound.caf';
{$ENDIF}
{$IFDEF ANDROID}
  Result := TPath.Combine(TPath.GetSharedDocumentsPath, 'myAndroidSound.mp3');
{$ENDIF}
end;
```

- For **C++**:
  - In the .cpp file, add the following code:

```
UnicodeString __fastcall TForm1::GetSoundName ()
{
    UnicodeString result;
#if defined(_PLAT_IOS)
    result = "myiOSSound.caf";
#endif
#if defined(__ANDROID__)
    result =
System::Ioutils::TPath::Combine(System::Ioutils::TPath::GetSharedDocumentsPath(
), "myAndroidSound.mp3");
#endif
```

```
    return result;
  //
}
```

7. Add the sound files to your project:
    - Find the myAndroidSound.mp3 and myiOSSound.caf files in the Windows Explorer, and drag them to the Project Manager. Drop them on the name of your project.

**Note**: The sound files myAndroidSound.mp3 and myiOSSound.caf are not included in the RAD Studio installation. You can use any MP3 and CAF files that you have, but remember to change the name in the code snippet above to the name of your files. See this link for more information on how to convert MP3 files to CAF format: How to convert MP3 to CAF

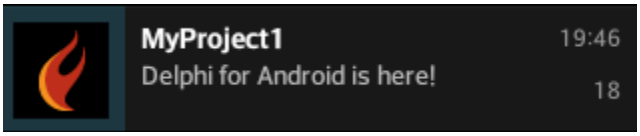
    - In the **Confirm** dialog, click **Yes** to add the sound files to your project.
8. Open the Deployment Manager to ensure that the sound files are deployed with your application. You can see the corresponding entries in the remote path column:
    - For iOS: `StartUp\Documents` (see deploying files in iOS applications).
    - For Android: `assets\internal` (see deploying files in Android applications).
9. Change the remote path of the added files in the Deployment Manager:
    - For iOS: `.\`
    - For Android: `assets\`

# Notification Banner or Notification Alert

By default, your application shows the notification banner:

- **Notification Banner on iPad**

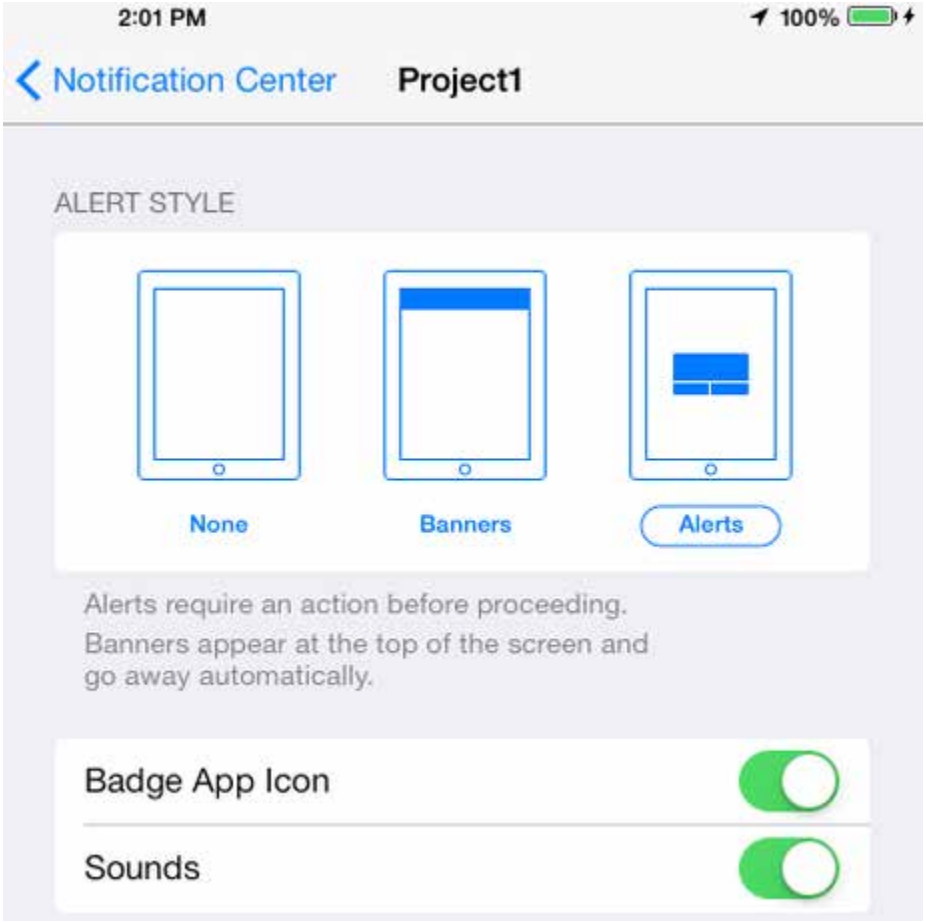

- **Notification Banner on Android devices**




Android

- **Notification Alert (only for iOS devices)**


To use a notification alert instead of a notification banner (only for iOS devices), the end user needs to change the **Alert** style to **Alerts** through the configuration page of Notification Center, available in the device Settings:




# Add Action to the Notification Alert (iOS Only)

You can also customize an alert by adding an Action button that opens the application.

To customize an Alert Action, you need to set the AlertAction field to the Action button name, and then set the HasAction field to **True**, as follows.

1. Drop a new TButton to the form.
2. In the Object Inspector, change the Name property to **ActionNotification**.
3. Create the OnClick event of the **ActionNotification** button by double-clicking the button.
4. Implement the event handler for the OnClick event of the **ActionNotification** button by adding the following code:

    - For **Delphi:**

```
procedure TForm1.ActionNotificationClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
   MyNotification := NotificationCenter1.CreateNotification;
  try
    MyNotification.Name := 'MyNotification';
    MyNotification.AlertBody := 'Delphi for iOS is here! ';
    MyNotification.Number := 2;
    MyNotification.AlertAction := 'Launch';
    MyNotification.HasAction := True;
    MyNotification.FireDate := Now + EncodeTime(0, 0, 20, 0);
    NotificationCenter1.ScheduleNotification(MyNotification);
  finally
    MyNotification.DisposeOf;
  end;
end;
```

    - For **C++:**

```
void __fastcall TForm1::ActionNotificationClick(TObject *Sender)
{
   if (NotificationCenter1->Supported()) {
           TNotification *myNotification = NotificationCenter1-
>CreateNotification();
           __try {
                   myNotification->Name = "MyNotification";
                   myNotification->AlertBody = "C++ for iOS is here!";
                   myNotification->Number = 2;
                   myNotification->AlertAction = "Launch";
                   myNotification->HasAction = True;
                   myNotification->FireDate = Now() + EncodeTime(0,0,20,0);
                   NotificationCenter1->ScheduleNotification(myNotification);
           }
           __finally {
                   myNotification->DisposeOf();
           }
   }
 }
```

**Note:** Only iOS devices support the notification alert feature.

The notification alert opens at the time that was specified through the FireDate field.


# Add Action to Notifications

The TNotificationCenter class provides the onReceiveLocalNotification event handler that allows you to write a response when the user clicks the notification message in the notification center. To write the response, double-click the TNotificationCenter component on the Form Designer, and then implement the OnReceiveLocalNotification event handler.

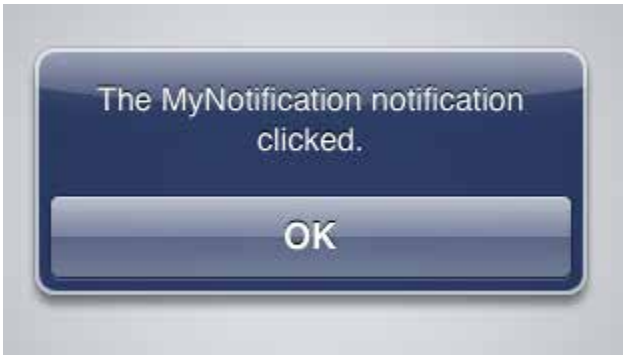
The following code snippet implements a response to show a message box that says "The <Notification name>" notification clicked."

- For **Delphi:**

```
 procedure TForm1.NotificationCenter1ReceiveLocalNotification(Sender: TObject;
ANotification: TNotification);
   begin
     ShowMessage('The ' + ANotification.Name + ' notification clicked.' );
   end;
```

- For **C++**:

```
void __fastcall TForm1::NotificationCenter1ReceiveLocalNotification(TObject *Sender,
TNotification *ANotification)
{
       ShowMessage("The " + ANotification->Name + " notification clicked.");
}
```




# Running the Application

To run the application, either choose **Run > Run** or press F9. Click the different buttons to schedule or present notifications in your device.

# See Also

- Using Notifications
- Mobile Tutorial: Taking and Sharing a Picture, and Sharing Text (iOS and Android)
- Mobile Tutorial: Using Location Sensors (iOS and Android)
- Mobile Tutorial: Using Remote Notifications (iOS and Android)
- Mobile Code Snippets: Notifications
- Creating events Index
- Using the OS X Notification Center
- How to convert MP3 to CAF

## Samples

- FireMonkey Send Cancel Notification sample
- FireMonkey Android Notification Service sample
- FireMonkey Set Reset Badge Number sample
- OSX Dock Badges (Delphi)
- FireMonkey Notification Mac (Delphi)
- FireMonkey Notification Mac (C++)

# Mobile Tutorial: Using the Phone Dialer on Mobile Devices (iOS and Android)

This tutorial describes the basic steps for using the phone dialer services on your mobile device.

## About the Phone Dialer Services on Mobile Devices

On mobile platforms, FireMonkey provides the IFMXPhoneDialerService interface that defines the structure of the phone dialer service classes, such as TPhoneDialerService.

The phone dialer services allow you to perform the following operations:

- Get the carrier-related information.
- Make a call.
- Detect the call state changes.

## Accessing the Phone Dialer Services

To create an application that uses the phone dialer services, perform the following basic steps:

1. Select:
   - For Delphi: **File > New > Multi-Device Application - Delphi**
   - For C++: **File > New > Multi-Device Application - C++Builder**
2. Open the Code Editor and do the following:

- Add the following lines to your code if they are not present:

**Delphi**:

```
uses
  FMX.Platform, FMX.PhoneDialer;
```

**C++:**

```
#include <FMX.Platform.hpp>
#include <FMX.PhoneDialer.hpp>
```

- Only for Delphi apps: Add the following line to the public section of the form definition:

```
 constructor Create(AOwner: TComponent); override;
```

```
```

- Add the following properties to the private section of the form definition:

**Delphi:**

```
 private: // User declarations
 PhoneDialerService: IFMXPhoneDialerService;
```

**C++:**

```
private: //User declarations
_di_IFMXPhoneDialerService phoneDialerService;
bool serviceSupported;
```

- Only for Delphi apps: in the implementation section, override the form constructor as follows:

```
 constructor TForm1.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService,
IInterface(PhoneDialerService));
end;
```

- Only for C++Builder apps: in the Structure View, click the form, and in the Object Inspector, open the **Events** tab, and then double-click **onCreate**. Implement the following **onCreate** event handler for the application form:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
        serviceSupported = (TPlatformServices::Current-
>SupportsPlatformService(__uuidof(IFMXPhoneDialerService)) &&
                           (phoneDialerService = TPlatformServices::Current-
>GetPlatformService(__uuidof(IFMXPhoneDialerService))));

}
```

Now your application can access the phone dialer services.

# Designing the User Interface

This tutorial uses the TLabel, TButton, and TEdit components as the user interface elements.

**To set up the UI elements, perform the following steps:**

1. Drop two TLabel components on the Form Designer, and then set their Name properties to **lblCarrierName** and **lblISOCountryCode**, respectively.

2. Set the Text property for the labels to **Carrier Name** and **ISO Country Code**, respectively.
3. Drop a **TButton** component on the Form Designer, and in the Object Inspector, set the following properties of this button:
    - Name to **btnGetCarrierInfo**
    - Text to **Get Carrier Info**

# Getting the Carrier Properties

To get information on the carrier, make the following changes:

1. On the Form Designer, select the **Get Carrier Info** button.
2. In the Object Inspector, double-click the **onClick** event, and implement the **onClick** event handler as follows:

**Delphi**:

```
procedure TForm1.btnGetCarrierInfoClick(Sender: TObject);
begin
  { test whether the PhoneDialer services are supported on your device }
  if Assigned(PhoneDialerService) then
  begin
    { if yes, then update the labels with the retrieved information }
    lblCarrierName.Text := 'Carrier Name:  '  +
PhoneDialerService.GetCarrier.GetCarrierName;
    lblISOCountryCode.Text := 'ISO Country Code: ' +
PhoneDialerService.GetCarrier.GetIsoCountryCode;
  end;
end;
```

**C++**:

```
void __fastcall TForm1::btnGetCarrierInfoClick(TObject *Sender)
{
        if (serviceSupported) {
            lblCarrierName->Text = "Carrier Name:  "  + phoneDialerService-
>GetCarrier()->GetCarrierName();
            lblISOCountryCode->Text = "ISO Country Code: " + phoneDialerService-
>GetCarrier()->GetIsoCountryCode();
        }   else ShowMessage("This device does not support the Phone Dialer
services.");

}
```

## Running the Application

**Important:** Before running your Delphi application on an Android device, verify that the following permissions are set in

Project > Options > Uses Permissions for the **All configurations - Android platform** target:

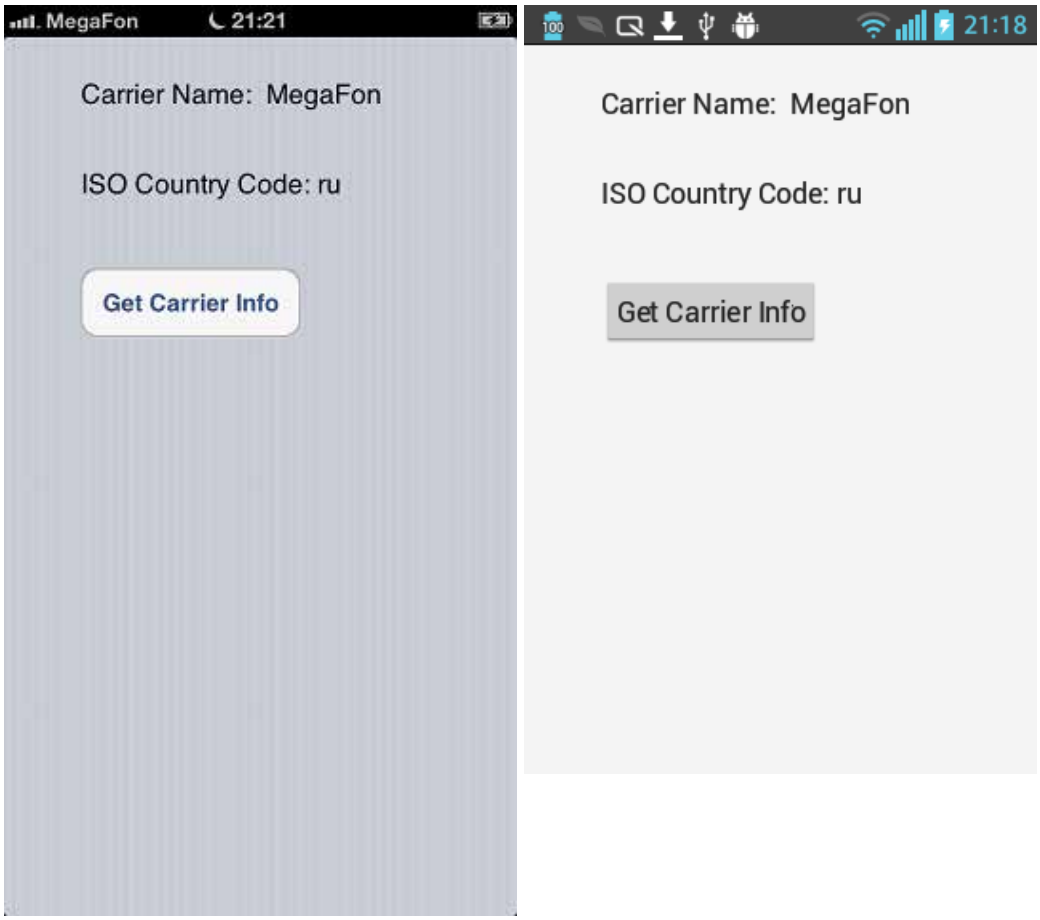
- **Call phone**
- **Read phone state**

For more information, see Uses Permissions.

To run the application, choose Run > Run or press **F9**.

After you click the **Get Carrier Info** button, the application displays the basic information about the carrier, similar to the following screens:

**iOS** **Android**




# Making a Call

FireMonkey provides the IFMXPhoneDialerService.Call method that makes a phone call to a specified phone number.

For your application to make calls, add the following elements to the Form Designer:

1. Add a TLabel component, and then set its Text property to **Telephone Number**.

2. Add a TEdit component, and in the Object Inspector, set the following properties:
    - Name to **edtTelephoneNumber**.
    - KillFocusByReturn to **True**.
    - KeyboardType to **PhonePad**.
    - ReturnKeyType to **Go**.
3. Add a TButton component, and in the Object Inspector, do the following:

- Set the Name property to **btnMakeCall**.
- Set the Text property to **Make Call**.
- On the **Events** tab, double-click **onClick**, and then implement the **onClick** event handler as follows:

**Delphi:**

```
procedure TForm1.btnMakeCallClick(Sender: TObject);
begin
  { test whether the PhoneDialer services are supported on your device }
  if Assigned(PhoneDialerService) then
  begin
    { if the Telephone Number is entered in the edit box then make the call, else
      display an error message }
    if edtTelephoneNumber.Text <> '' then
      PhoneDialerService.Call(edtTelephoneNumber.Text)
    else
    begin
      ShowMessage('Please type-in a telephone number.');
      edtTelephoneNumber.SetFocus;
    end;
  end;
end;
```

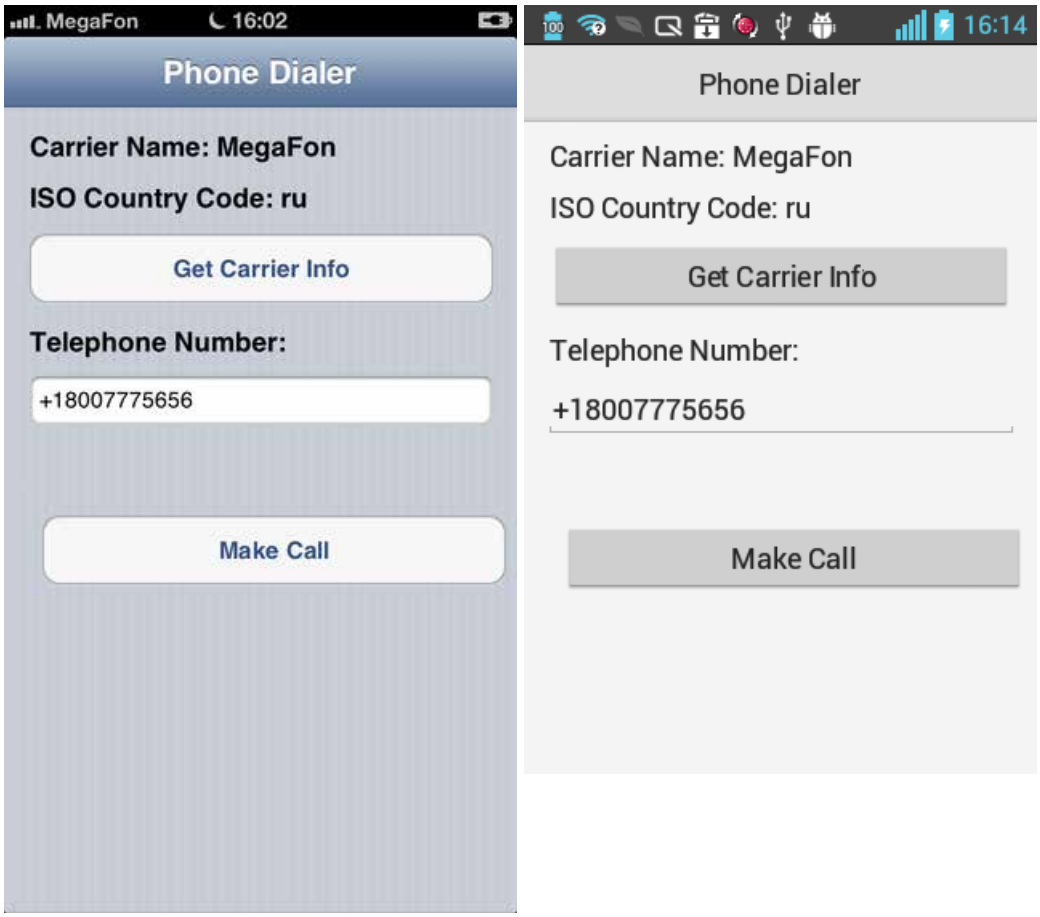
**C++:**

```
void __fastcall TForm1::btnMakeCallClick(TObject *Sender)
{
       if (serviceSupported) {
              if (edtTelephoneNumber->Text != "" )   {
                  phoneDialerService->Call(edtTelephoneNumber->Text);
              }
                   else {
                   ShowMessage("Please type-in a telephone number.");
                   edtTelephoneNumber->SetFocus();
                   }
       }   else ShowMessage("This device does not support the Phone Dialer
services.");
}
```

**To make a call:**

1. Run the application.
2. In the TEdit field under **Telephone Number**, type the phone number.
3. Click the **Make Call** button.

**iOS** **Android**




# Detecting the Call State Changes

The IFMXPhoneDialerService interface provides the OnCallStateChanged event that allows you to handle the call state changes. The TCallState enumeration describes possible phone call states.

The following table describes the items in the TCallState enumeration (the supported states for each platform are marked with "+").

| Item | Description | iOS | Android |
|---|---|---|---|
| None | No call state. | + | - |
| Connected | The phone caller is connected to the called party. | + | + |
| Incoming | An incoming phone call. | + | + |
| Dialing | The phone is in a dialing state. | + | - |
| Disconnected | Call is disconnected. | + | + |

## Implementing the OnCallStateChanged Event Handler

To implement the OnCallStateChanged event handler, perform the following steps:

1. Add the following procedure header to the private section of the form definition:

   **Delphi**:

```
 procedure MyOnCallStateChanged(const ACallID: String; const ACallState:
TCallState);
```

   **C++**:

```
void __fastcall MyOnCallStateChanged(const UnicodeString aCallID, const
TCallState aCallState);
```

2. Rewrite the form constructor (Delphi applications) or the **onFormCreate** event handler (C++ applications) that you have defined in the #Accessing the Phone Dialer Services section as follows:

   **Delphi**:

```
 constructor TForm1.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService,
IInterface(PhoneDialerService));
  if Assigned(PhoneDialerService) then
    PhoneDialerService.OnCallStateChanged := MyOnCallStateChanged;
end;
```

   **C++**:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
       serviceSupported = (TPlatformServices::Current-
>SupportsPlatformService(__uuidof(IFMXPhoneDialerService)) &&
                (phoneDialerService = TPlatformServices::Current-
>GetPlatformService(__uuidof(IFMXPhoneDialerService))));
       if (serviceSupported) {
           phoneDialerService->OnCallStateChanged = MyOnCallStateChanged;
       }
}
```

3. Add a TLabel component to the Form Designer, and in the Object Inspector, set its **Name** property to **lblCallState**.
4. In the Code Editor, add the following event handler code:

   **Delphi**:

```
procedure TForm1.MyOnCallStateChanged(const ACallID: String; const ACallState:
TCallState);
var outText: String;
 Begin
  case ACallState of
         TCallState.None:         outText := 'No calls';
         TCallState.Connected:    outText := 'Connected';
         TCallState.Incoming:     outText := 'Incoming call';
         TCallState.Dialing:      outText := 'Dialing';
         TCallState.Disconnected: outText := 'Disconnected';
     end;
   lblCallState.Text := outText;
 End;
```

   **C++**:

```
void __fastcall TForm1::MyOnCallStateChanged(const UnicodeString aCallID, const
TCallState aCallState) {
        switch (aCallState) {
            case TCallState::None:
                lblCallState->Text = "No calls";
                break;
            case TCallState::Connected:
                lblCallState->Text = "Connected";
                break;
            case TCallState::Incoming:
                lblCallState->Text = "Incoming call";
                break;
            case TCallState::Dialing:
                lblCallState->Text = "Dialing";
                break;
            case TCallState::Disconnected:
                lblCallState->Text = "Disconnected";
                break;
        }
 }
```

After you implement this event handler, the application shows the phone call state.

For example, the following iOS screen indicates that the phone is in a dialing state:


**Note:** In this sample project, the TLabel component is next to the TEdit box and the **Make Call** button, under **Telephone Number**.

# See Also

- Mobile Tutorial: Taking and Sharing a Picture, and Sharing Text (iOS and Android)
- Mobile Tutorial: Using Location Sensors (iOS and Android)
- Android Mobile Application Development
- iOS Mobile Application Development

- Mobile Code Snippets: Phone Dialer
- FMX.PhoneDialer.IFMXPhoneDialerService

## Samples

- FireMonkey Phone Dialer sample

# Mobile Tutorial: Using Remote Notifications (iOS and Android)

This tutorial gives the basic steps to configure and to use Remote Notifications (push notifications) on your iOS or Android mobile device.

**Note:** Kindle Fire devices do not support push notifications.

## Remote Push Notification

Remote Notifications are notifications sent to a mobile device using a data-channel from a service provider in real-time.

Both **iOS** and **Android** offer built-in support for remote notifications, and RAD Studio offers a REST BaaS framework that supports the following protocols, cloud providers and backend service:

- Protocols:
  - iOS: Apple Push Notification APN
  - Android: Google Cloud Messaging GCM
- Cloud service providers:
  - Parse
  - Kinvey
- Backend service:
  - EMS

In order to receive push notifications, you need to set up the messaging service (APS or GCM), the device, the cloud service provider or EMS, and your RAD Studio application, as described in the three pages of this mobile tutorial.

**Notes:**

- The term **Remote Notifications** covers *Apple Push Notification* as well as *Google Cloud Messaging*.
- **iOS** and **Android** also have local notifications which are sent from an app or from the OS to get the user's attention. For more information, see Mobile Tutorial: Using Notifications (iOS and Android).
- The term **Service Provider** covers in this tutorial the Cloud service providers (Kinvey and Parse) and the backend service provider EMS.

## REST BAAS framework

Our REST BAAS framework offers you a variety of actions:

- Create, retrieve, update and delete objects
- Sign up, login, retrieve, update and delete users
- Upload, download and delete files or streams
- Query objects and users
- Send push notifications
- Register for and receive push notifications on a device

# Topics in this Mobile Tutorial

We recommend that you perform these sequentially as steps in a procedure.

1. **Setting Up the Messaging Service**
2. **Setting up the Service Provider**. Choose one of the following options, depending on the service provider:
   - **Using the Cloud Service to Send Push Notifications**
   - **Using EMS to Send Push Notifications**
3. **Multi-Device Application to Receive Push Notifications**

# See Also

- BaaS Overview
- Getting Started with Kinvey and Parse
- EMS Push Notifications
- Enterprise Mobility Services (EMS)
- Managing BaaS Users
- Using BaaS for Backend Storage
- Mobile Tutorial: Using Notifications (iOS and Android)
- Mobile Tutorial: Using BaaS for Backend Storage (iOS and Android)

## Code Samples

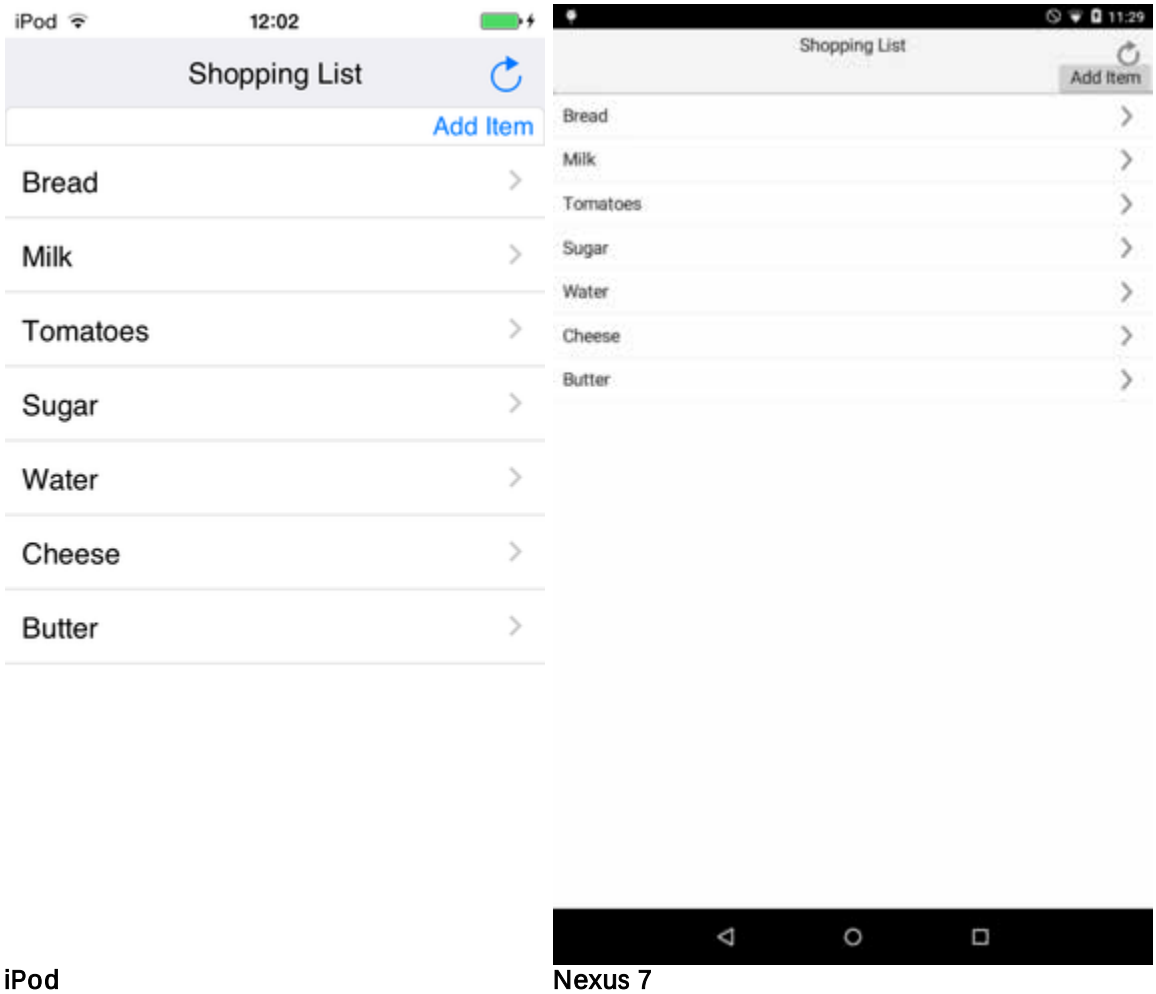
- BaaS ToDo Sample

# Mobile Tutorial: Using BaaS for Backend Storage (iOS and Android)

This tutorial gives the basic steps to use **backend storage** using **Kinvey** and **Parse** as BaaS providers.

This tutorial creates a **Shopping List** where the items added are stored in the cloud by the BaaS provider, so these items can be shared and updated among different devices if needed.

**iOS** **Android**




**iPod** **Nexus 7**

With backend storage you can store files and data in the cloud. This information can later be retrieved, updated, and deleted.

In this tutorial you are going to:

- Create new items and store them in the cloud.

- Retrieve items from the cloud.
- Delete items from the cloud.

# Getting Your App Ready in Kinvey and Parse

Before you start doing this tutorial, ensure you already have an account with **Kinvey** or **Parse** and that you already have an app created in the backend provider.

- Getting Started with Kinvey and Parse
  - Creating an Account with Kinvey
  - Creating an Account with Parse
  - Creating an App with Kinvey
  - Creating an App with Parse

# Design and Set Up of the User Interface

This tutorial uses a TListView as UI component to display the information retrieved from the cloud.

To set up the UI for this sample tutorial follow these steps:

1. Create a blank Multi-Device Application, by selecting:
   - For **Delphi**: **File > New > Multi-Device Application - Delphi > Blank Application**
   - For **C++**: **File > New > Multi-Device Application - C++Builder > Blank Application**
2. Drop a TLabel on the form and set the following properties in the Object Inspector:
   - Set the Text property to **Shopping List**.
   - Set the Align property to **Top**.
   - Set the HorzAlign property in TextSettings to **Center**.
3. Drop a TButton on the **TLabel** and set the following properties in the **Object Inspector**:
   - Set the Align property to **Right**.
   - Set the StyleLookup property to **refreshtoolbutton**.
   - Set the Name property to **RefreshButton**.
4. Drop a TEdit on the form and set the following properties in the **Object Inspector**:
   - Set the Align property to **Top**.
5. Drop a TButton on the **TEdit** and set the following properties in the **Object Inspector**:

- Set the Align property to **Right**.
- Set the Text property to **Add Item**.
- Set the Name property to **AddItemButton**.

6. Drop a TListView on the form and set the following properties in the **Object Inspector**:
   - Set the Align property to **Client**.

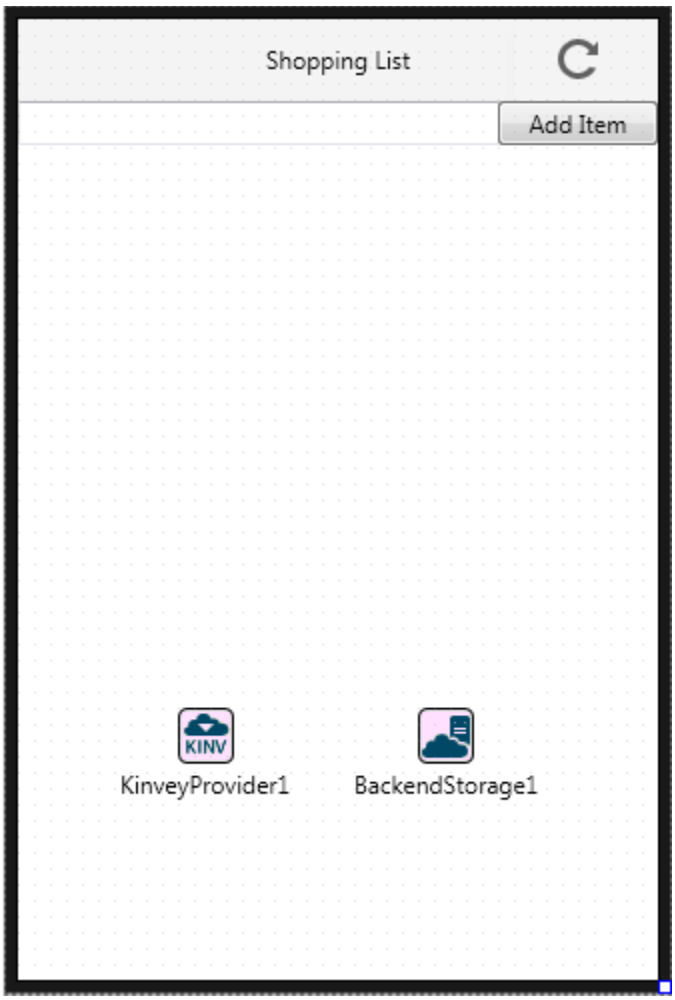
# Adding the Backend Components

This tutorial uses the TBackendStorage service component to manage the backend storage in order to create, retrieve, update, and delete objects. This app also uses one of the BaaS provider components:

- TKinveyProvider is the **Kinvey** BaaS provider component that contains the information about the Kinvey could service connection.
- TParseProvider is the **Parse** BaaS provider component that contains the information about the Parse could service connection.

Continue with the following steps:

1. Drop one of the two BaaS provider components on the form **TKinveyProvider** or **TParseProvider** depending on the backend provider you use in your app.
2. Drop a **TBackendStorage** on the form, and in the **Object Inspector** set the Provider property to the BaaS provider component you are using.
3. Depending on the BaaS provider you are using:
   - Select the **KinveyProvider1** component and in the Object Inspector set the AppKey, AppSecret and MasterSecret to the values provided by **Kinvey**.
   - Select the **ParseProvider1** component and in the Object Inspector set the ApplicationID, MasterKey and RestApiKey to the values provided by **Parse**.




# Creating and Storing Objects

In the app, when you type something in the **TEdit** and click **Add Item**, the content typed in the **TEdit** gets stored in the cloud, in the **Kinvey** or **Parse** BaaS provider. To create the object, use the CreateObject method.

1. Double-click the **AddItemButton** and type the following code:

**Delphi:**

```
procedure TForm1.AddItemButtonClick(Sender: TObject);
var
    LJSONObject: TJSONObject;
    LEntity: TBackendEntityValue;
begin
    if Edit1.Text = '' then
      ShowMessage('Please, enter an item.')
    else
      begin
        LJSONObject := TJSONObject.Create;
        LJSONObject.AddPair('item', Edit1.Text);
        BackendStorage1.Storage.CreateObject('ShoppingList', LJSONObject, LEntity);
```

```
        ShowMessage('New item created: ' +Edit1.Text);
        LJSONObject.Free;
        Edit1.Text := '';
      end
end;
```

**C++:**

```
void __fastcall TForm1::AddItemButtonClick(TObject *Sender)
{
        TJSONObject *LJSONObject;
        TBackendEntityValue LEntity;

        if (Edit1->Text == "") {
                ShowMessage("Please, enter an item.");
        }
        else  {
                LJSONObject = new TJSONObject;
                LJSONObject->AddPair("item", Edit1->Text);
                BackendStorage1->Storage->CreateObject("ShoppingList", LJSONObject,
LEntity);
                ShowMessage("New item created: "+ Edit1->Text);
                LJSONObject->Free();
                delete LJSONObject;
                Edit1->Text="";
         }
}
```

# Deleting Objects

To allow users to delete any item from the list. In the TListView component, if you swipe your finger left or right over one of the items of the **TListView**, a **Delete** button displays on the right-hand side. To delete the item when the **Delete** button of such item is clicked, do the following:

1. Select the **TListView** and in the **Events** tab of the Object Inspector, double-click the OnDeletingItem event. This event occurs before the item is deleted. Add the following code:

**Delphi:**

```
procedure TForm1.ListView1DeletingItem(Sender: TObject; AIndex: Integer; var
ACanDelete: Boolean);
var
  LQuery: string;
  LJSONArray : TJSONArray;
  LEntities: TArray<TBackendEntityValue>;
begin
  ACanDelete := False;
  LJSONArray := TJSONArray.Create;
  try
```

```
    LQuery := Format('query={"item":"%s"}', [(Sender as
TListView).Items[AIndex].Text]); // 'query={"item":"%s"}' in Kinvey and
'where={"item":"%s"}' in Parse
    BackendStorage1.Storage.QueryObjects('ShoppingList', [LQuery], LJSONArray,
LEntities);
    if (Length(LEntities) > 0) and
BackendStorage1.Storage.DeleteObject('ShoppingList', LEntities[0].ObjectID) then
      ACanDelete := True
    else
      ShowMessage ('Item could not be deleted.');
  finally
    LJSONArray.Free;
  end;
end;
```

**Note:** The line LQuery :=Format('query={"item":"%s"}',[(Sender as TListView).Items[AIndex].Text]); is for Kinvey. In Parse, the word **query** must change to **where** so if you are using Parse as BaaS provider, this line must be LQuery :=Format('where={"item":"%s"}',[(Sender as TListView).Items[AIndex].Text]);.

**C++:**

```
void __fastcall TForm1::ListView1DeletingItem(TObject *Sender, int AIndex, bool
&ACanDelete)
{
        System::UnicodeString LQuery[1];
        DynamicArray<TBackendEntityValue> LEntities;
        ACanDelete = False;
        TJSONArray *LJSONArray;
        LJSONArray = new TJSONArray;
        try {
                TListView* list = reinterpret_cast<TListView*>(Sender);
                LQuery[0] = ("query={\"item\":\""+(listView1->Items-
>operator[](AIndex)->Text)+"\"}"); // "query={\"item\":\"" in Kinvey and
"where={\"item\":\"" in Parse
                BackendStorage1->Storage->QueryObjects("ShoppingList", LQuery, 0,
LJSONArray, LEntities);
                if (((LEntities.Length)>0) & BackendStorage1->Storage-
>DeleteObject("ShoppingList", LEntities[0].ObjectID)) {
                   ACanDelete = True;
                } else {
                        ShowMessage("Item could not be deleted.");
                }
        }
        __finally {
                LJSONArray->Free();
        }
}
```

**Note:** The line LQuery[0]=("query={\"item\":\""+(list->Items->Item[AIndex]->Text)+"\"}"); is for Kinvey. In Parse, the word **query** must change to **where** so if you are using Parse as BaaS provider, this line must be LQuery[0]=("where={\"item\":\""+(list->Items->Item[AIndex]->Text)+"\"}");.

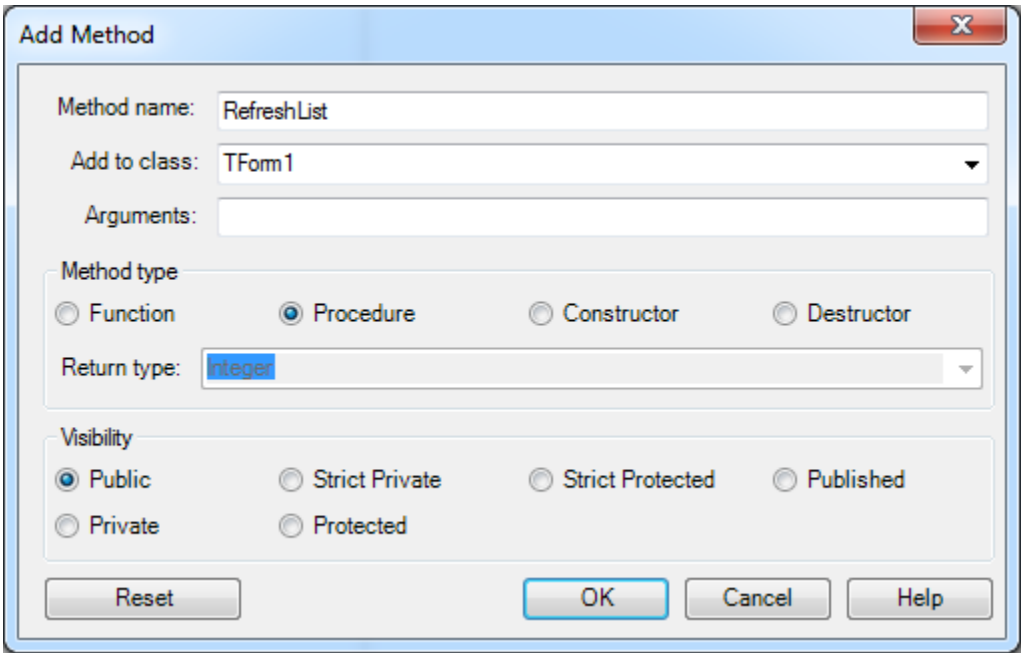
# Retrieving Objects

To update all content in the **TListView** to reflect any possible change, you can retrieve the objects with QueryObjects, clear the list view and add all items you have retrieved from the backend storage.

1. To add a new method to refresh the list:

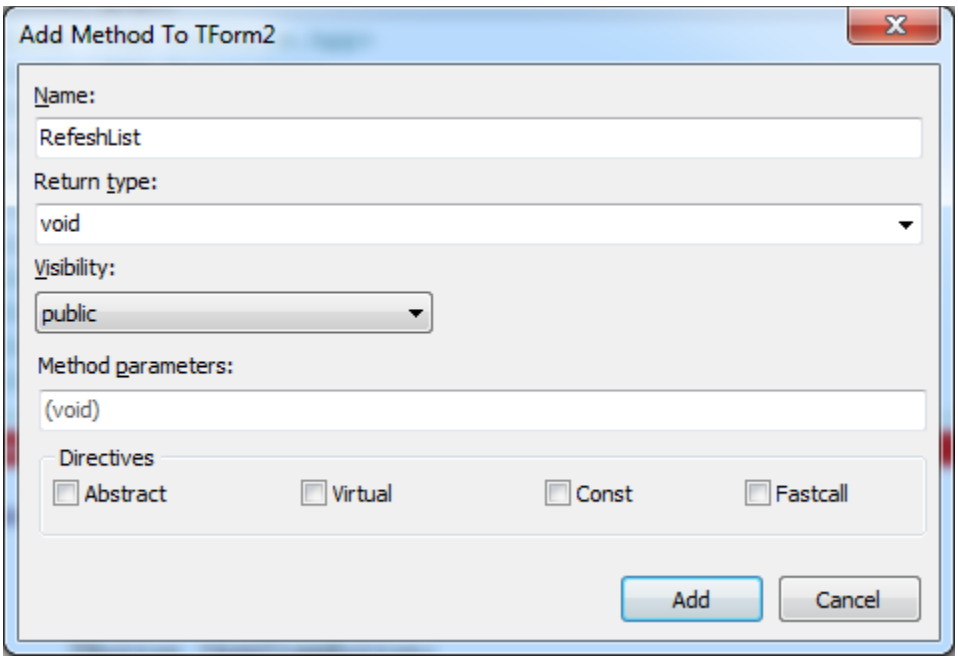
   **Delphi:**

   - Open the Class Explorer window by choosing **View > Class Explorer**.
   - In the **Class Viewer** pane, right-click **TForm1** under **Unit1** and select **Add Method**.
   - Write **RefreshList** under "Method name", select **Procedure** under "Method type" and click **OK**.




   **C++:**

   - Open the Class Explorer window by choosing **View > C++ Class Explorer**.
   - In the **Type List** pane (left pane), expand your project, right-click **TForm1** under and select **Add Method**.
   - Write **RefreshList** under "Name", **void** under "Return type", set "Visibility" to **public** and click **Add**.




2. Add the following code to the new method:

**Delphi:**

```
procedure TForm1.RefreshList;
var
  LJSONArray : TJSONArray;
  LItem: TListViewItem;
  I: Integer;
begin
  LJSONArray := TJSONArray.Create;
  try
    BackendStorage1.Storage.QueryObjects('ShoppingList', [], LJSONArray);
    ListView1.ClearItems;
    for I := 0 to LJSONArray.Count-1 do
    begin
      LItem := ListView1.Items.Add;
      LItem.Text := (LJSonArray.Items[I].GetValue<string>('item'));
    end;
  finally
    LJSONArray.Free;
  end;
end;
```

**C++:**

```
void TForm1::RefreshList()
{
        String LQuery [1];
        TJSONArray *LJSONArray;
        TListViewItem *LItem;
        TJSONObject *LJSONObject;
        String ItemText;
```

```
        int i;
        LJSONArray = new TJSONArray;
        try {
                BackendStorage1->Storage->QueryObjects("ShoppingList", LQuery,
0, LJSONArray);
                ListView1->Items->Clear();
                for (i = 0; i < LJSONArray->Count; i++) {
                        LItem = ListView1->Items->Add();
                        LJSONObject = dynamic_cast<TJSONObject *>(LJSONArray-
>Items[i]);
                        ItemText = (LJSONObject->Values["item"]->Value());
                        LItem->Text = ItemText;
                }
        }
        __finally {
                LJSONArray->Free();
        }
}
```

3. Double-click the **RefreshButton** and the following code so this button calls the **RefreshList** method:

   **Delphi:**

```
procedure TForm1.RefreshButtonClick(Sender: TObject);
begin
  RefreshList;
end;
```

   **C++:**

```
void __fastcall TForm1::RefreshButtonClick(TObject *Sender)
{
        RefreshList();
}
```

4. To refresh the list once an item is added, double-click the **AddItemButton** to access the code of the OnClick event added before and add the following code as the very last sentence of the else statement:

   **Delphi:**

```
RefreshList;
```

   **C++:**

```
RefreshList();
```

5. To refresh the list after an item is deleted, select the **TListView** and in the **Events** tab of the Object Inspector, double-click the OnDeleteItem event. This event occurs after the item is deleted. Add the following code:

   **Delphi:**

```
procedure TForm1.ListView1DeleteItem(Sender: TObject; AIndex: Integer);
begin
  RefreshList;
end;
```

   **C++:**

```
void __fastcall TForm1::ListView1DeleteItem(TObject *Sender, int AIndex)
{
        RefreshList();
}
```

# Running Your Application

To run your application, follow these steps:

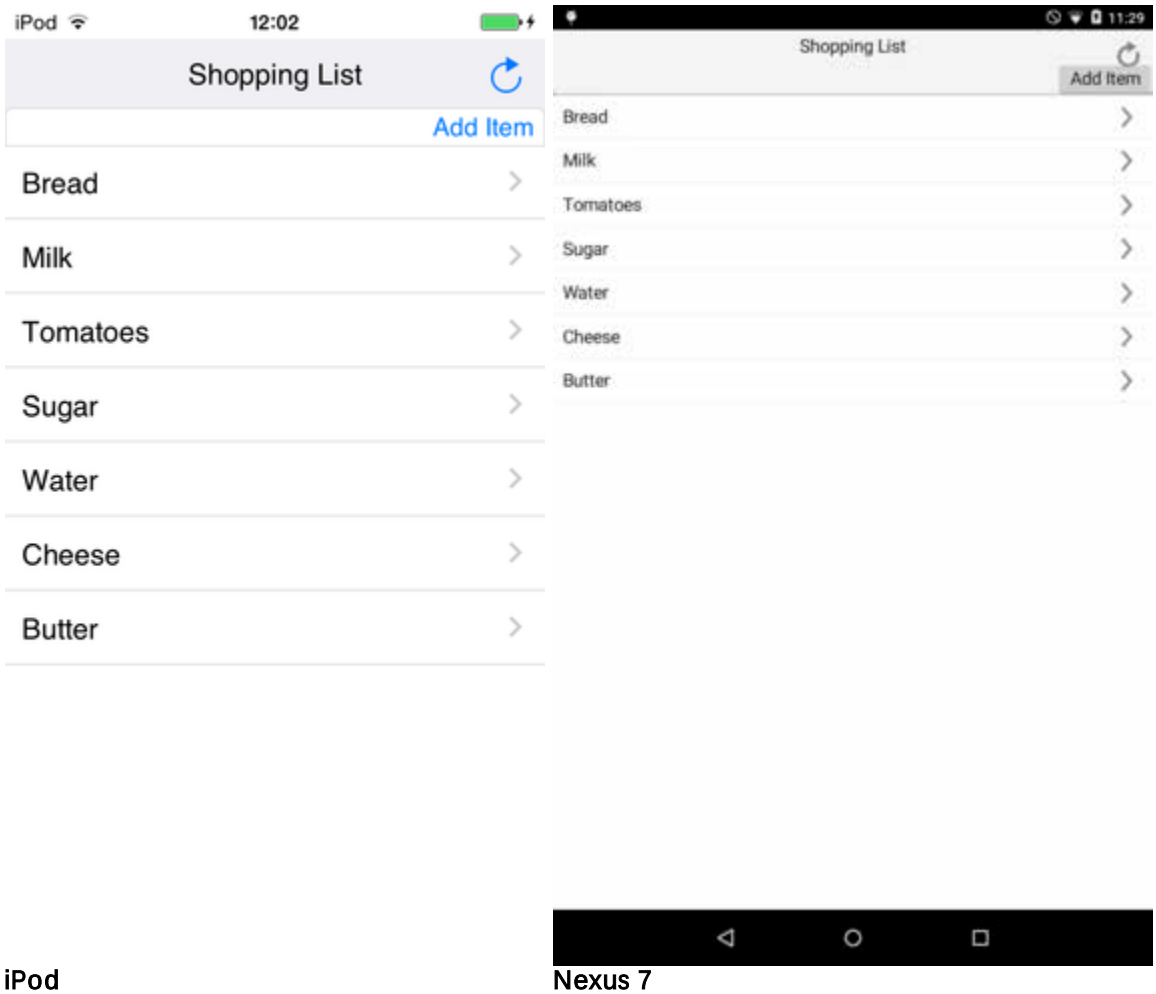
1. In the Project Manager, select the target platform where you want to run your app.

**Note:** This app uses the **swipe-to-delete** feature that is supported on mobile apps (iOS and Android), as well as desktop apps (Windows and OS X) when touch input is available.

2. Select **Run > Run** or **Run > Run Without Debugging**.
3. Once you add or delete items in your application, you can go to your Kinvey console or to your Parse dashboard to see the changes also from the cloud.

iOS

Android




iPod

Nexus 7

# See Also

- BaaS Overview
- Getting Started with Kinvey and Parse
- Managing BaaS Users
- Using BaaS for Backend Storage
- Mobile Tutorial: Using Remote Notifications (iOS and Android)

## Code Samples

- BaaS ToDo Sample

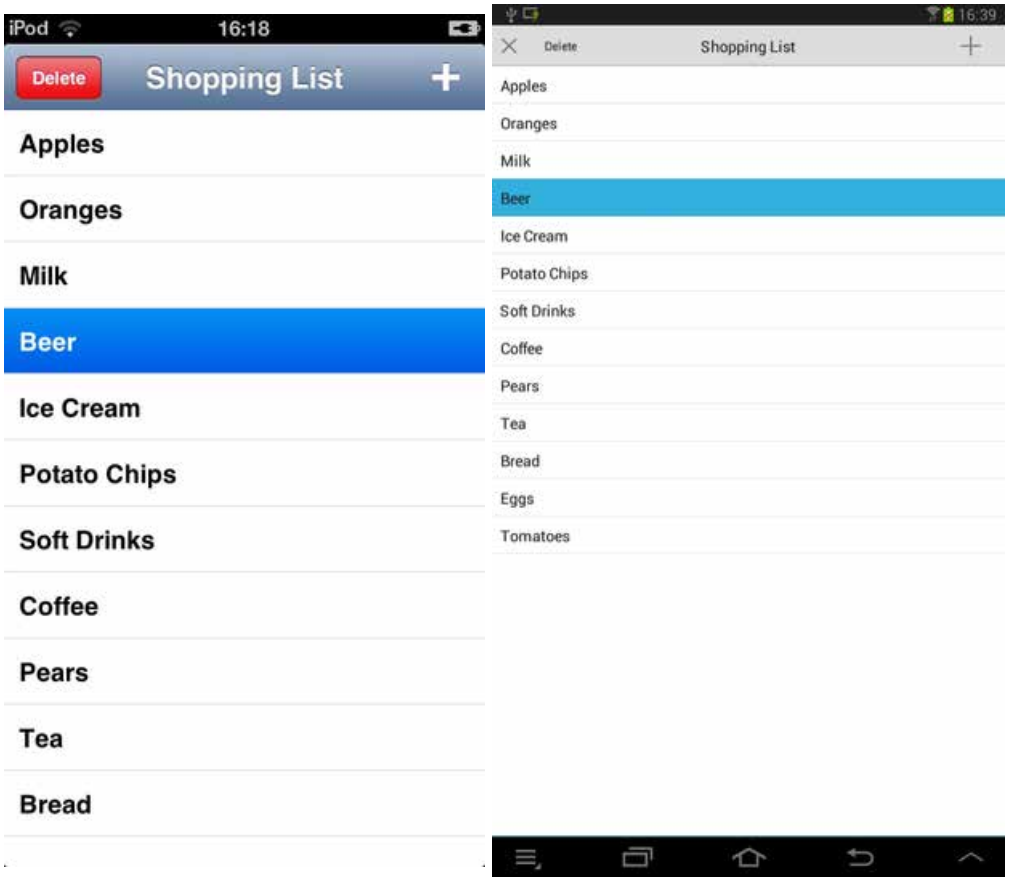
# Mobile Tutorial: Using FireDAC and SQLite (iOS and Android)

Before starting this tutorial, you should read and perform the following tutorial session:

- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

This tutorial describes the basic steps to use SQLite as a local data storage on your mobile device through the FireDAC framework.

**iOS** **Android**




## Using FireDAC to Connect to the Database

FireDAC is a unique set of **Universal Data Access** Components for developing cross-platform database applications for Delphi and C++Builder. With its powerful common architecture, FireDAC enables native high-speed direct access from Delphi to InterBase, SQLite, MySQL,

SQL Server, Oracle, PostgreSQL, IBM DB2, SQL Anywhere, Access, Firebird, Informix, and more.

- For the mobile platforms, **FireDAC** supports **InterBase ToGo** as well as **SQLite**. These database products can run on iOS and Android devices.
- For other databases, such as Oracle, you need to have at least a client library. On Windows platforms, the client library is provided as a DLL to connect to. Therefore, you need to develop applications using middle-tier technologies such as DataSnap REST to connect to these database products from a mobile device.

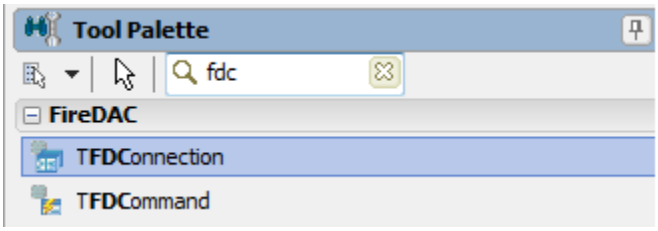
Another tutorial discusses how to connect to Enterprise Database without using a client library on a mobile device; see Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android).

# Creating the Database using FireDAC framework

First, you need to create a SQLite database file on your Windows development platform. Use the following steps, so that you can use the Form Designer to design the user interface of your Mobile App.

**Note:** In the Form Designer, activate the **Master** view for this tutorial.

1. To create an Multi-Device Application, select:
   - For Delphi: **File > New > Multi-Device Application - Delphi > Blank Application**
   - For C++: **File > New > Multi-Device Application - C++Builder > Blank Application**
2. On the Tool Palette, double-click the **TFDConnection** component.



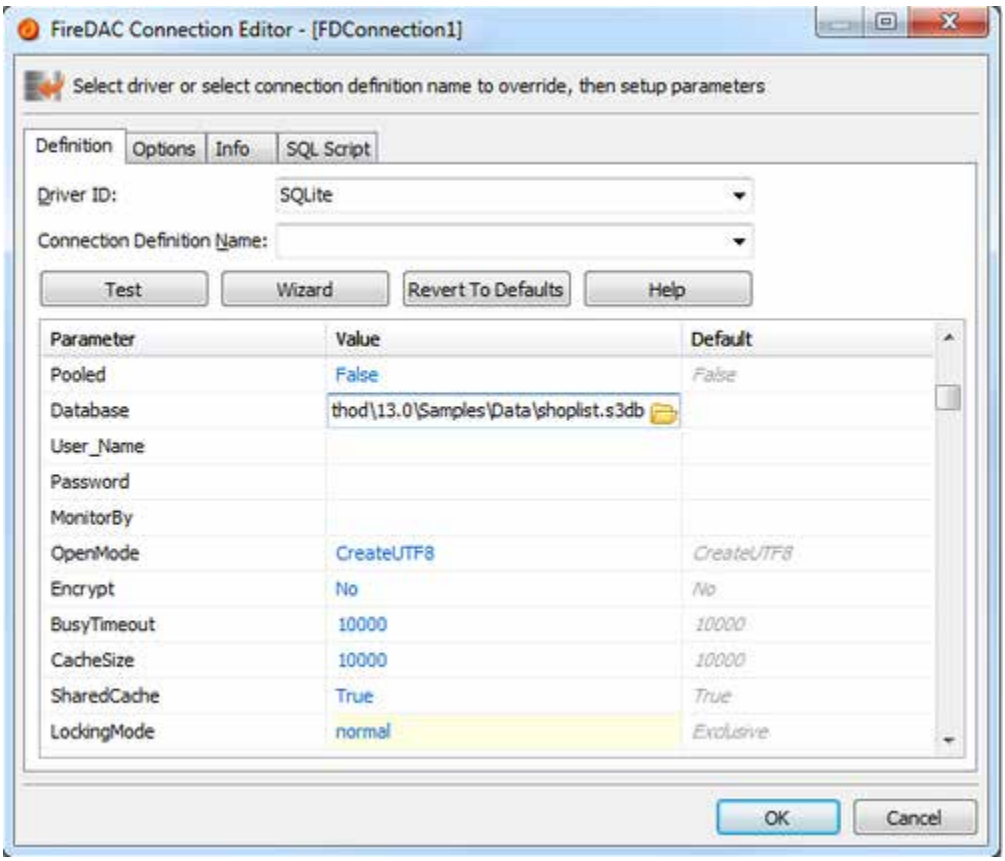

3. Right click the TFDConnection component and choose **Connection Editor**.
4. In the FireDAC Connection Editor, set the following parameters of the TFDConnection:
   - Set the Driver ID property to **SQLite**.
   - Set the **Database** parameter to:

`C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\shoplist.s3db` (location of the database)

and click **Open** in the File Open dialog box.

**Note**: You can set the above **Database** parameter even if **shoplist.s3db** does not exist, RAD Studio creates it automatically. (To display **Employees.s3db** or other ***.s3db** files in the Open dialog, set the **All Files (*.*)** option.)

- Set the **LockingMode** parameter to **normal**.



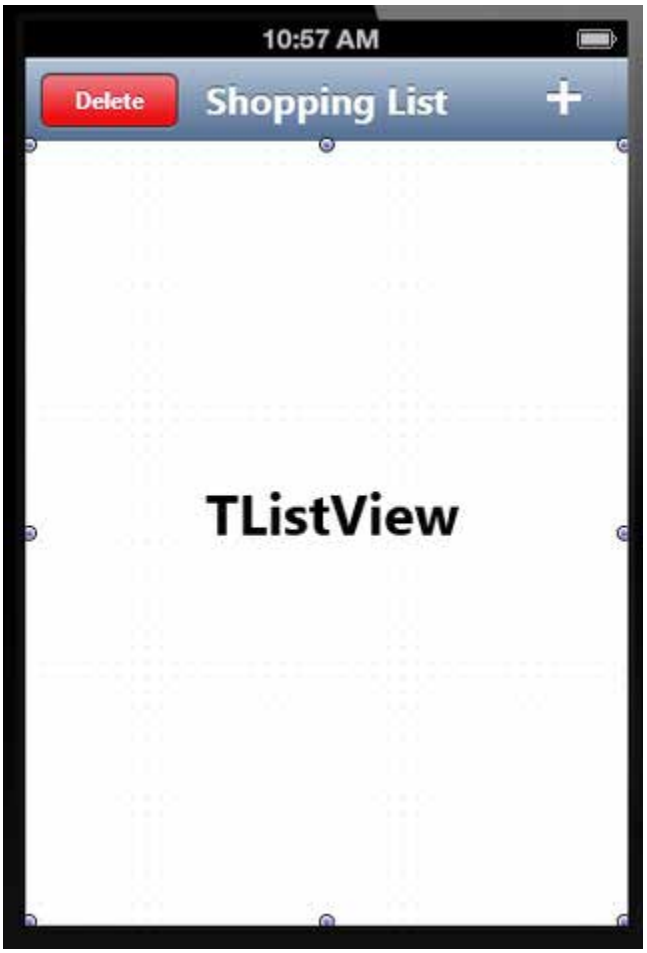

- Click the **Test** button to test the connection.
- Click **OK** to close the Connection Editor.

5. In the Object Inspector, set the following properties of **TFDConnection**:
   - Set the **LoginPrompt** property to **False**, so that the user is not prompted for a login.
   - Set the **Connected** property to **True**.
6. On the Tool Palette, double-click the **TFDQuery** component.
7. In the Object Inspector, set the following properties of **TFDQuery**:
   - Set the Name property to **FDQueryCreateTable**.
   - Set the SQL property as follows:

```
CREATE TABLE IF NOT EXISTS Item (ShopItem TEXT NOT NULL)
```

8. Right click the **FDQueryCreateTable** and choose **Execute**.

# Design and Set Up the User Interface



Visible UI components are loaded on the designer

This tutorial uses one TListView component as the UI element.

**To set up a ListView component and other UI elements, use the following steps:**

1. Drop a **TToolBar** on the form.
2. Drop a **TButton** on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Name property to **ButtonAdd**.
   - Set the StyleLookup to **addtoolbutton**.
   - Set the Align to **Right**.
3. Drop a **TButton** on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Name property to **ButtonDelete**.
   - Set the StyleLookup to **deletetoolbutton**.

- Set the Align to **Left**.
- Set the Text to **Delete**.
- Set the Visible property to **False**.

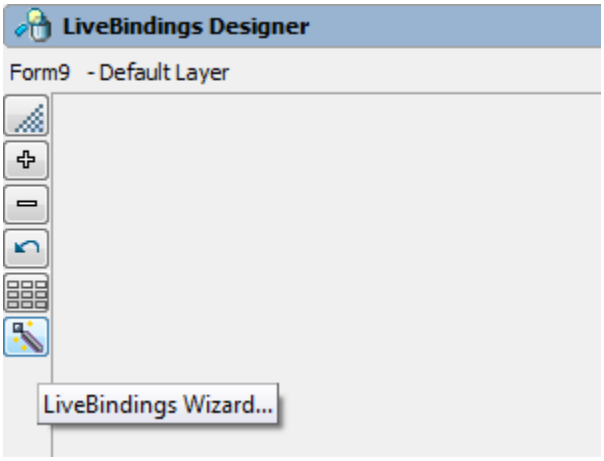
4. Drop a **TLabel** on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Align to **Client**.
   - Set the StyleLookup to **toollabel**.
   - Set the Text to **Shopping List**.
   - Expand the TTextSettings node and set the HorzAlign property to **Center**.
5. Drop a **TListView** component on the form and set the following properties in the Object Inspector:
   - Set the Align property to **Client**, so that the ListView component uses the entire form.

# Using the LiveBindings Wizard

This tutorial uses the **LiveBindings Wizard** to add the LiveBindings components (TBindSourceDB, TBindingsList), and the TFDQuery component.

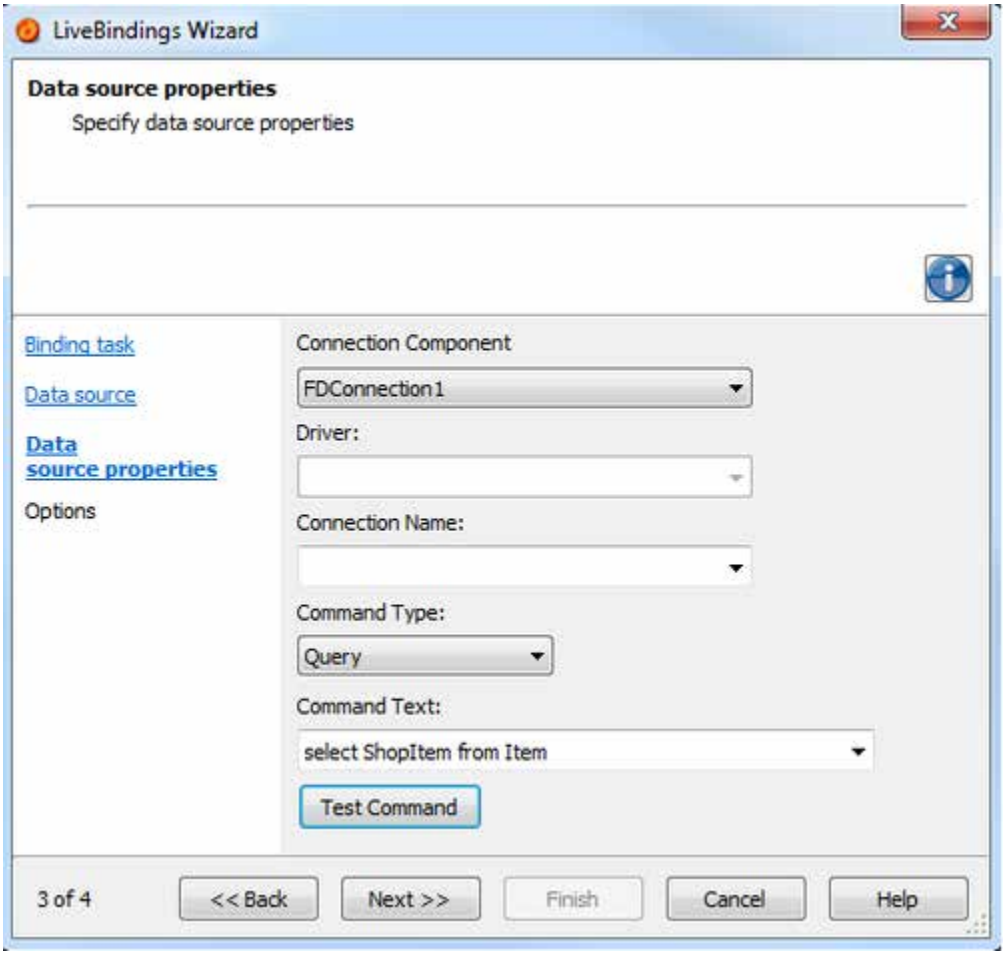
## Add the LiveBinding components

1. Select **View > LiveBindings Designer** and the LiveBindings Designer opens.
2. Select LiveBindings Wizard.




3. Select **Create a data source** binding task.
4. Click the **Next** button.
5. Select **FireDAC** class name.

6. Click the **Next** button.
7. Change the **Command Type** to **Query**.
8. Set the **Command Text** property to `select ShopItem from Item`.

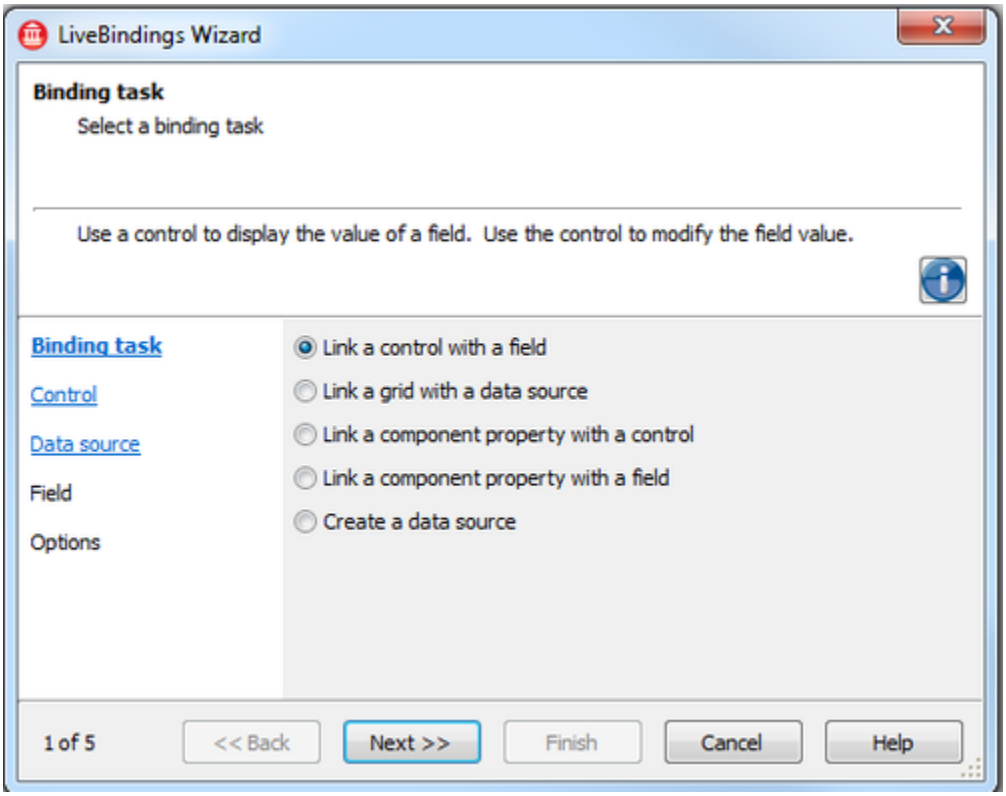



9. Click the **Test Command** button.
10. Click the **Next** button.
11. Click the **Finish** button.

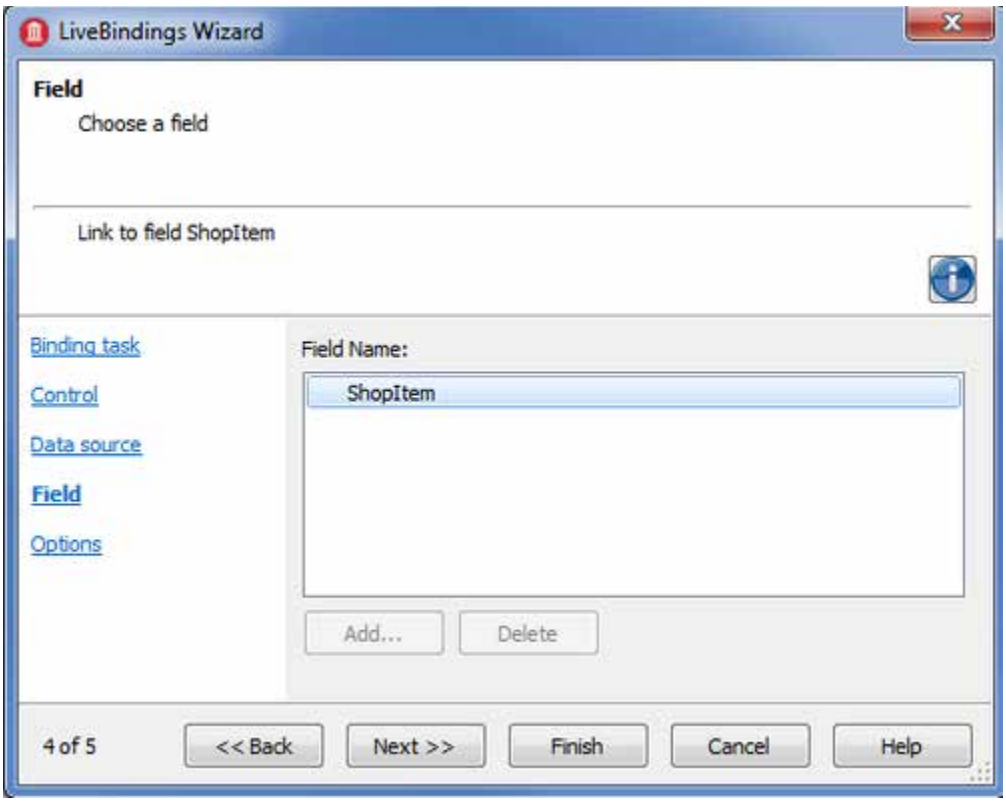
At this point, TBindSourceDB and TFDQuery components were added to your form.

## Connecting to the Data

1. Reopen the LiveBindings Wizard.
2. Select **Link a control with a field** binding task.




3. Click the **Next** button.
4. Select the **Existing Control** tab.
5. Select the **ListView1** component.
6. Click the **Next** button.
7. Select **BindSourceDB1**.
8. Click the **Next** button.
9. Select **ShopItem** Field Name.



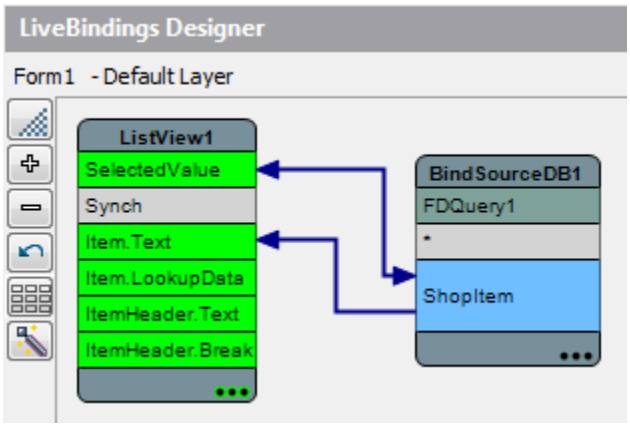

10. Click the **Next** button.
11. Click the **Finish** button to close the wizard.

**Note:** These last steps are not mandatory for this tutorial since there is only one field in **BindSourceDB1**. These steps are useful to link with the selected value if we are managing several fields of a database.

## Displaying ShopItem in the ListView

The following step displays the text of **ShopItem** in the **TListView** component.

1. In the LiveBindings Designer select **ShopItem** in the **BindSourceDB1** component and drag **ShopItem** to **Item.Text** in **ListView1**.



Following these steps connects the user interface of the app with data on a SQLite database. If you used a table with existing data for this tutorial, now you should see actual data within the Form Designer.

# Creating the Event Handler to Make the Delete Button Visible When the User Selects an Item from the List

The Visible property for the **Delete** button is set to **False**. Therefore, by default, the end user does not see this button. You can make it visible when the user selects an item on the list, as follows:

- Select **ListView1** and define the following event handler for the OnItemClick event.

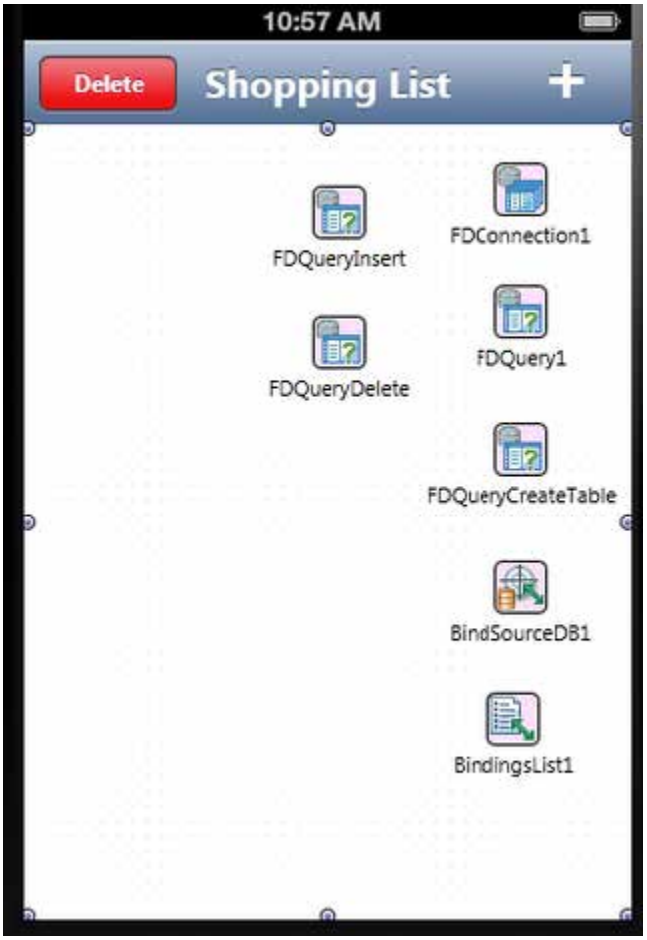
**Delphi:**

```
procedure TForm1.ListView1ItemClick(const Sender: TObject;
  const AItem: TListViewItem);
begin
  ButtonDelete.Visible := ListView1.Selected <> nil;
end;
```

- For **C++**:

```
void __fastcall TForm1::ListView1ItemClick(const TObject *Sender,
    const TListViewItem *AItem) {
    ButtonDelete->Visible = (ListView1->Selected != NULL);
}
```

# Creating the Event Handler for the Add Button to Add an Entry to the List



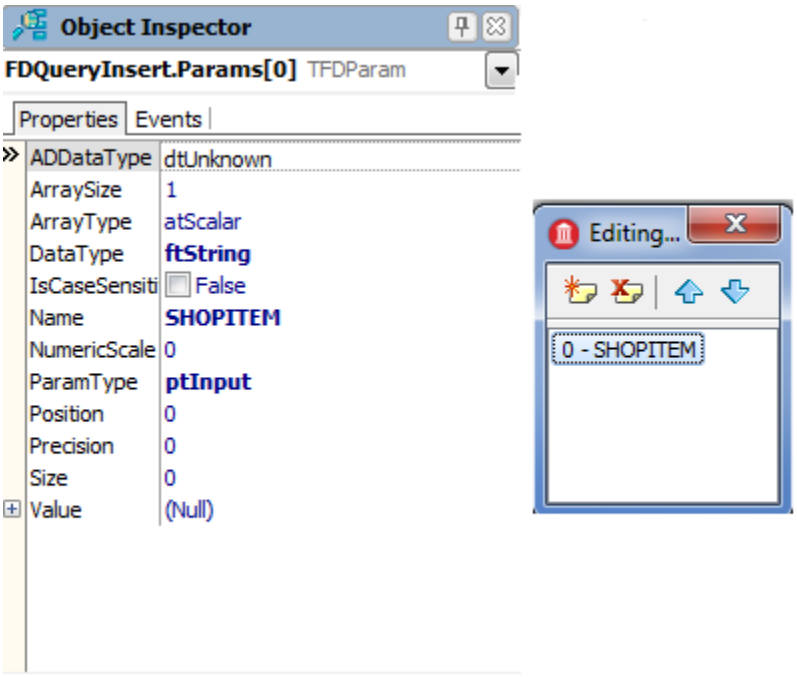

Database connections are also configured

The next step is adding a feature to this application for adding an item to the shopping list.

1. Drop a TFDQuery component on the form.
2. Set the following properties in the Object Inspector:
    - Set the Name property to **FDQueryInsert**.
    - Set the SQL property as follows:

```
INSERT INTO ITEM (ShopItem) VALUES (:ShopItem)
```

    - Select the **Expand** (...) button on the Params property.
    - Select the **ShopItem** parameter and set DataType to **ftString**:




3. In the Structure View, right-click the **ButtonAdd** component and select **Control > Bring to Front**. This brings the button to the visual front of the active form.

- For **Delphi**:
- Declare the following procedure in the private section:

```
private
  procedure OnInputQuery_Close(const AResult: TModalResult; const AValues: array of
string);
```

- Add the following procedure :

```
procedure TForm1.OnInputQuery_Close(const AResult: TModalResult; const AValues: array
of string);
var
  TaskName: String;
begin
 TaskName := string.Empty;
  if AResult <> mrOk then
    Exit;
  TaskName := AValues[0];
  try
    if (TaskName.Trim <> '')
    then
    begin
      FDQueryInsert.ParamByName('ShopItem').AsString := TaskName;
      FDQueryInsert.ExecSQL();
      FDQuery1.Close();
      FDQuery1.Open;
      ButtonDelete.Visible := ListView1.Selected <> nil;
    end;
```

```
  except
    on e: Exception do
    begin
      ShowMessage(e.Message);
    end;
  end;
end;
```

- In the Form Designer, double-click the **ButtonAdd** component. Add the following code to this event handler:

```
procedure TForm1.ButtonAddClick(Sender: TObject);
begin
  TDialogServiceAsync.InputQuery('Enter New Item', ['Name'], [''],
Self.OnInputQuery_Close)
end;
```

- For **C++**:

To replicate the same functionality in C++, additional steps are required.

- Add the following definition after the TForm1 definition (in the .h file of your unit):

```
typedef void __fastcall(__closure * TInputCloseQueryProcEvent)
    (const System::Uitypes::TModalResult AResult,
    System::UnicodeString const *AValues, const int AValues_High);
```

- Add the following class definition (in the .h file of your unit, after the previously defined type):

```
class InputQueryMethod : public TCppInterfacedObject<TInputCloseQueryProc>
{
private:
    TInputCloseQueryProcEvent Event;

public:
    InputQueryMethod(TInputCloseQueryProcEvent _Event) {
        Event = _Event;
    }

    void __fastcall Invoke(const System::Uitypes::TModalResult AResult,
        System::UnicodeString const *AValues, const int AValues_High) {
        Event(AResult, AValues, AValues_High);
    }
};
```

- Add the following declaration under the private section of the form (in the .h file of your unit):

```
private: //User declarations
```

```
    void __fastcall OnInputQuery_Close
        (const System::Uitypes::TModalResult AResult,
        System::UnicodeString const *AValues, const int AValues_High);
```

- Add the following code (in the .cpp file of your unit):

```
void __fastcall TForm1::OnInputQuery_Close(const System::Uitypes::TModalResult
AResult,
        System::UnicodeString const *AValues, const int AValues_High) {
    String TaskName;
    TaskName = "";
    if (AResult != mrOk)
        return;
    TaskName = AValues[0];
try {
        if  (!(Trim(TaskName) == "")) {
            FDQueryInsert->ParamByName("ShopItem")->AsString = TaskName;
            FDQueryInsert->ExecSQL();
            FDQuery1->Close();
            FDQuery1->Open();
            ButtonDelete->Visible = (ListView1->Selected != NULL);
        }
    }
    catch (Exception &e) {
        ShowMessage(e.Message);
    }
}
```

- In the Form Designer, double-click the **ButtonAdd** component. Add the following code to this event handler:

```
void __fastcall TForm1::ButtonAddClick(TObject *Sender) {
    String caption = "Enter New Item";
    String Prompts[1];
    Prompts[0] = "Name:";
    String Defaults[1];
    Defaults[0] = "";
    _di_TInputCloseQueryProc Met = new InputQueryMethod(&OnInputQuery_Close);
    TDialogServiceAsync::InputQuery(
        caption, Prompts, 0, Defaults, 0, (TInputCloseQueryProc *)Met);
}
```

The InputQuery function shows a dialog box asking the end user to enter text. This function returns **True** when the user selects **OK**, so that you can add data to the database only when the user selects **OK** and the text contains some data.

iOS


Android


# Creating the Event Handler for the Delete Button to Remove an Entry from the List

The next step is adding a feature to this application to remove an item from the shopping list:

1. Drop a TFDQuery component on the form.
2. Set the following properties in the Object Inspector:
    - Set the Name property to **FDQueryDelete**.
    - Set the SQL property as follows:

```
delete from Item where ShopItem = :ShopItem
```

    - Select the **Expand** (...) button on the Params property.
    - Select the **ShopItem** parameter and set DataType to **ftString**.
3. In the Structure View, right-click the **ButtonDelete** component and select **Control > Bring to Front**. This brings the button to the visual front of the active form.
4. In the Form Designer, double-click the **ButtonDelete** component. Add the following code to this event handler.

- For **Delphi**:

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
var
  TaskName: String;
begin
  TaskName := TListViewItem(ListView1.Selected).Text;

  try
    FDQueryDelete.ParamByName('ShopItem').AsString := TaskName;
    FDQueryDelete.ExecSQL();
    FDQuery1.Close;
    FDQuery1.Open;
    ButtonDelete.Visible := ListView1.Selected <> nil;
  except
    on e: Exception do
    begin
      SHowMessage(e.Message);
    end;
  end;
end;
```

- For **C++**:

```
void __fastcall TForm1::ButtonDeleteClick(TObject *Sender) {
    String TaskName = ((TListViewItem*)(ListView1->Selected))->Text;
    try {
        FDQueryDelete->ParamByName("ShopItem")->AsString = TaskName;
        FDQueryDelete->ExecSQL();
        FDQuery1->Close();
        FDQuery1->Open();
        ButtonDelete->Visible = (ListView1->Selected != NULL);
    }
    catch (Exception &e) {
        ShowMessage(e.Message);
    }
}
```

# Preparing Your Application for Run Time

FireDAC has a loosely-coupled multilayered architecture, where layers provide services. A service API is defined as a COM interface that other layers can request using the interface factory.

To properly operate FireDAC, you must link the implementation of the **IFDGUIxWaitCursor** and **IFDPhysDriver** interfaces to your application.

For this, drop the TFDGUIxWaitCursor and TFDPhysSQLiteDriverLink components on the form.

# Setting Up Your Database Deployment for mobile

Up to this point, you have used SQLite on your desktop. This means that the actual database is located on your local hard disk drive (for example,

C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\shoplist.s3db). On the mobile Device, applications are sand-boxed, and typically you can only read and write data that is located in the **Documents** folder (for iOS device) and **internal** storage (for Android device) under your application folder.

To connect to a local database on mobile, you need to perform the following actions:

- Deploy the database to the mobile device.
- Check the configuration (to connect to the database file) to a local file under the **Documents** folder (for iOS device) or **internal** storage (for Android device).

## Add and Configure Your Database File in the Deployment Manager

Before you can run your application on mobile, you need to set up the deployment for your database file (shoplist.s3db).

1. You can add the database to your project with one of the following two methods:
   - Right-click the project name in the **Project Manager** and select **Add...** from the context menu (or **Project > Add to Project**) to display the Add to Project dialog box. Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data`, select the database `shoplist.s3db` and click **Open**.
   - Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data` and drag and drop the database `shoplist.s3db` to the project in the Project Manager. Click **Yes** to confirm that you want to add the file to your project.
2. After adding the database file, the **Featured Files** window displays, click **Cancel** to close it.
3. Open the Deployment Manager by selecting **Project > Deployment**.
4. Select **Debug configuration - iOS Device - 32 bit platform**, **Debug configuration - iOS Device - 64 bit platform** or **Debug configuration - Android platform** from the drop-down list of target platforms at the top of the Deployment Manager and see that the database `shoplist.s3db` has been added to the platforms.
5. See how the **Remote Path** of `shoplist.s3db` has been set for iOS and Android platforms:
   - **Remote Path** on iOS Device platform: `StartUp\Documents\`


   - **Remote Path** on Android platform: `assets\internal\`

As you just configured, when you run the app on the mobile device, the database file (shoplist.s3db) is set to be deployed to the **Documents** folder (for iOS platform) or **internal** storage (for Android platform) in the sandbox area of your multi-device application.

# Modifying Your Code to Connect to a Local Database File on mobile

The basic features of this application are now implemented. Following the steps from this tutorial, you created a database file on Windows. The database file is not available on your mobile device unless you copy it to the mobile device or create it on the fly.

You can create a SQLite Database and Table with the following steps:

## Specifying the Location of the SQLite Database on the Mobile Device

1. In the Form Designer, select the **FDConnection1** component.
2. In the Object Inspector, double-click the BeforeConnect event.
3. Add the following code to this event handler:

- For **Delphi**:

```
procedure TForm1.FDConnection1BeforeConnect(Sender: TObject);
begin
  {$IF DEFINED(iOS) or DEFINED(ANDROID)}
  FDConnection1.Params.Values['Database'] :=
      TPath.Combine(TPath.GetDocumentsPath, 'shoplist.s3db');
  {$ENDIF}
end;
```

The TPath record is declared in **System.IOUtils** unit, so you need to add **System.IOUtils** in the uses clause of your unit.

```
implementation

{$R *.fmx}
uses  System.IOUtils;
```

- For **C++**:

```
void __fastcall TForm1::FDConnection1BeforeConnect(TObject *Sender) {
#if defined(_PLAT_IOS) || defined(_PLAT_ANDROID)
    FDConnection1->Params->Values["Database"] =
        System::Ioutils::TPath::Combine(System::Ioutils::TPath::GetDocumentsPath
        (), "shoplist.s3db");
#endif
}
```

You need to add **#include <System.IOUtils.hpp>**.

### Creating a Table if None Exists

With SQLite you can create a table when no table exists, by using the `CREATE TABLE IF NOT EXISTS` statement. You can create a table after the TFDConnection component connects to the database and before the TFDQuery component connects to the table. Use the following steps:

1. In the Form Designer, select the **FDConnection1** component.
2. In the Object Inspector, double-click the AfterConnect event.
3. Add the following code to this event handler:

- For **Delphi**:

```
procedure TForm1.FDConnection1AfterConnect(Sender: TObject);
begin
  FDConnection1.ExecSQL('CREATE TABLE IF NOT EXISTS Item (ShopItem  TEXT NOT NULL)');
end;
```

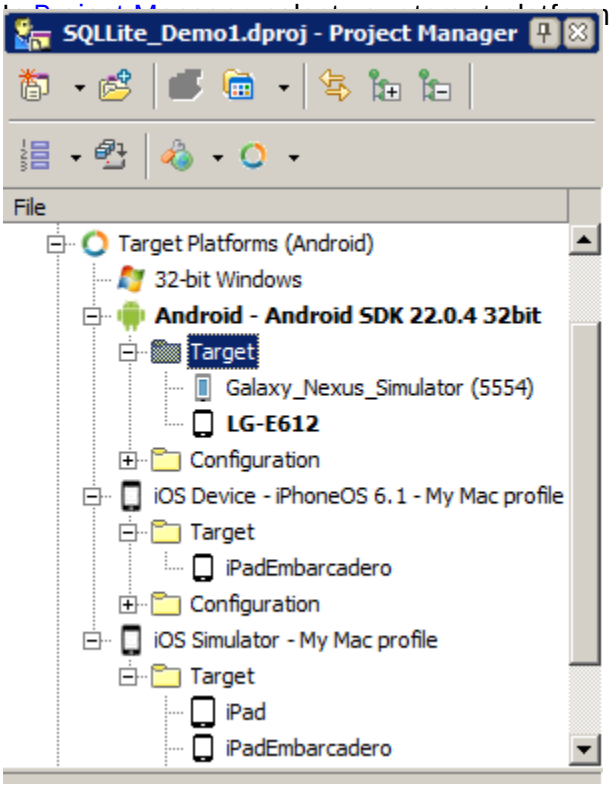
- For **C++**:

```
void __fastcall TForm1::FDConnection1AfterConnect(TObject *Sender) {
    FDConnection1->ExecSQL("CREATE TABLE IF NOT EXISTS Item (ShopItem  TEXT NOT
NULL)");
}
```

## Running Your Application on a Simulator or on a Mobile Device

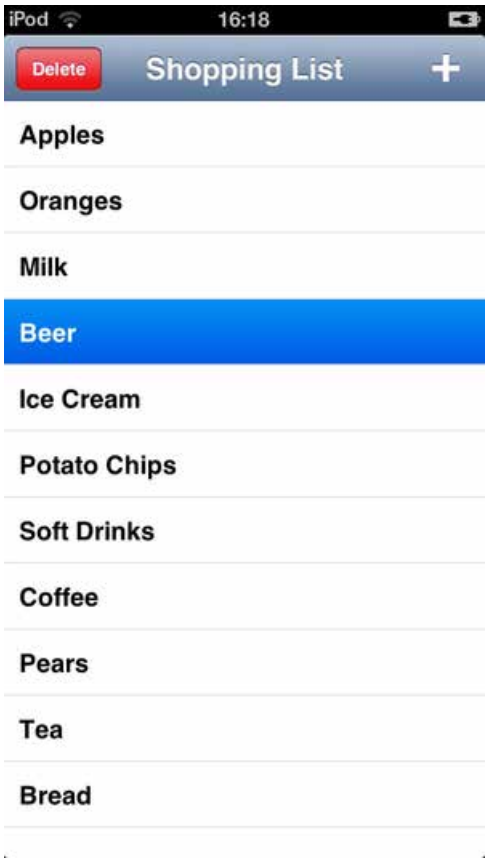
Now your application is ready to run on either a simulator or your connected mobile device.
**To run your application**

1. In Project Manager, select a target platform.

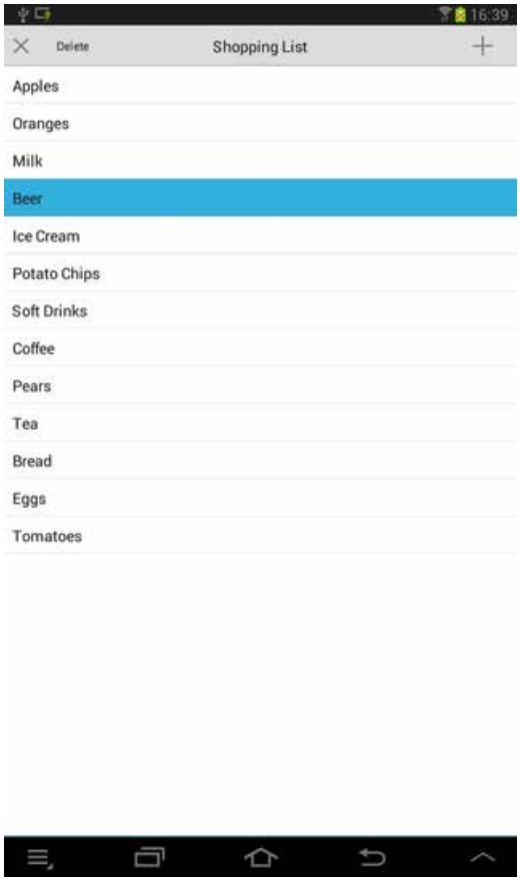



2. Choose either of the following commands:
   - Run > Run
   - Run > Run Without Debugging

iOS




Android




**Note:** If you have an issue with running the application, follow the steps given in **Troubleshooting**.

# See Also

- Mobile Tutorial: Using InterBase ToGo with FireDAC (iOS and Android)
- Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)
- SQLite support in RAD Studio
- Android Mobile Application Development
- iOS Mobile Application Development

# Mobile Tutorial: Using InterBase ToGo with FireDAC (iOS and Android)

Before starting this tutorial, you should read and perform the following tutorial session:

- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

  **Tip:** To follow this tutorial, you need a **license** for **IBToGo** or **IBLite**:

  - If you purchased one of the following RAD Studio versions, you have received in Email a key for an unlimited development and deployment license for IBLite:
    - RAD Studio Tokyo Professional or higherAll Editions
    - Delphi Tokyo Professional or higher with Mobile
  - If you are a trial user, your installation includes a trial license for IBToGo. You can test InterBase on iOS and Android by selecting your test license during the deployment step, as described in this tutorial. The trial licenses are installed with your trial product, in `C:\Users\Public\Documents\Embarcadero\InterBase\redist\InterBaseXE7`.

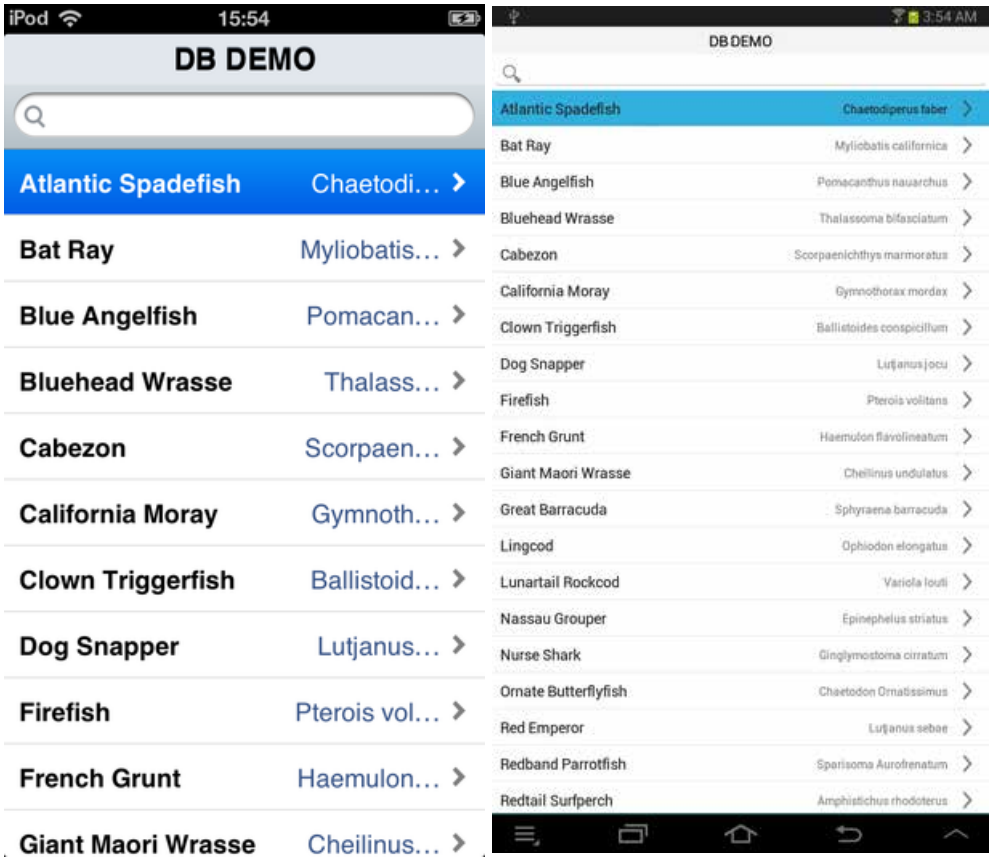
  Follow the steps at IBLite and IBToGo Test Deployment Licensing to obtain and install your license file.

**Note:** On Android devices, InterBase ToGo apps require specific permissions to be set, specifically:

- **Read external storage** (the database is placed in the external memory)
- **Write external storage** (the database is placed in the external memory)
- **Internet** (you need to connect with a remote server)

This tutorial describes the basic steps to browse data managed by InterBase ToGo on your iOS and Android devices through the FireDAC framework.

iOS Android




**Note:** You can use FireDAC, dbExpress, and Interbase Express (IBX) components to build **Interbase ToGo** applications. For a detailed discussion on Interbase Express components usage in a Delphi application, read the Getting Started with InterBase Express article. For this tutorial, we will connect to **Interbase ToGo** using FireDAC.

# Using FireDAC to Connect to the Database

FireDAC is a unique set of **Universal Data Access Components** for developing cross-platform database applications for Delphi and C++Builder. With its powerful common architecture, FireDAC enables native high-speed direct access from Delphi to InterBase, SQLite, MySQL, SQL Server, Oracle, PostgreSQL, IBM DB2, SQL Anywhere, Access, Firebird, Informix, and more.

- For the mobile platforms, **FireDAC** supports **InterBase ToGo** as well as **SQLite**. These database products can run on iOS and Android devices.
- For other databases, such as Oracle, you need to have at least a client library. On Windows platforms, the client library is provided as a DLL to connect to. Therefore, you need to develop applications using middle-tier technologies such as DataSnap to connect to these database products from a mobile device.

Another tutorial discusses how to connect to Enterprise Database without using a client library on a mobile device; see Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android).

# Design and Set Up the User Interface

This tutorial uses TListView and TPanel components as the UI elements.

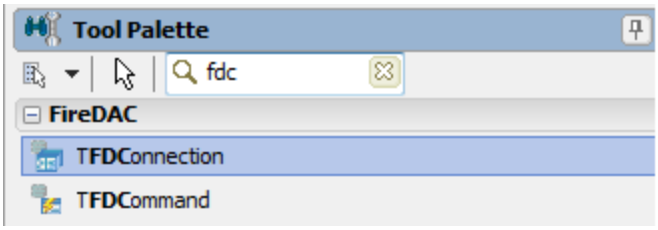
**To set up a ListView and a Panel component, use the following steps:**

1. To create an HD Multi-Device Application, select either of the following:
   - **File > New > Multi-Device Application - Delphi > Blank Application**
   - **File > New > Multi-Device Application - C++Builder > Blank Application**
2. Drop a TListView component on the form.
3. In the Object Inspector, set the following properties of the ListView:
   - Set the Align property to **Client**, so that the ListView component uses the entire form.
   - Set the ItemAppearance to **ListItemRightDetail**.
   - Set the SearchVisible to **true**.
4. Add a TPanel component to the form, and set the following properties in the Object Inspector:
   - Set the Align property for the **TPanel** component to **Top**.
5. Add a TLabel component to the Panel, and set the following properties in the Object Inspector:
   - Set the Align property for the **TLabel** component to **Client**.
   - Set the **Views** to either iOS or Android.
   - Set the StyleLookup property to **listboxitemlabel**.
   - Set back the **Views** to **Master**.
   - Set the HorzAlign property in TextSettings to **Center**.
   - Set the Text property to **DB DEMO**.

# Connecting to the Data

Following are the basic steps to connect to data in a database using FireDAC:

1. On the Tool Palette, double-click the **TFDConnection** component.

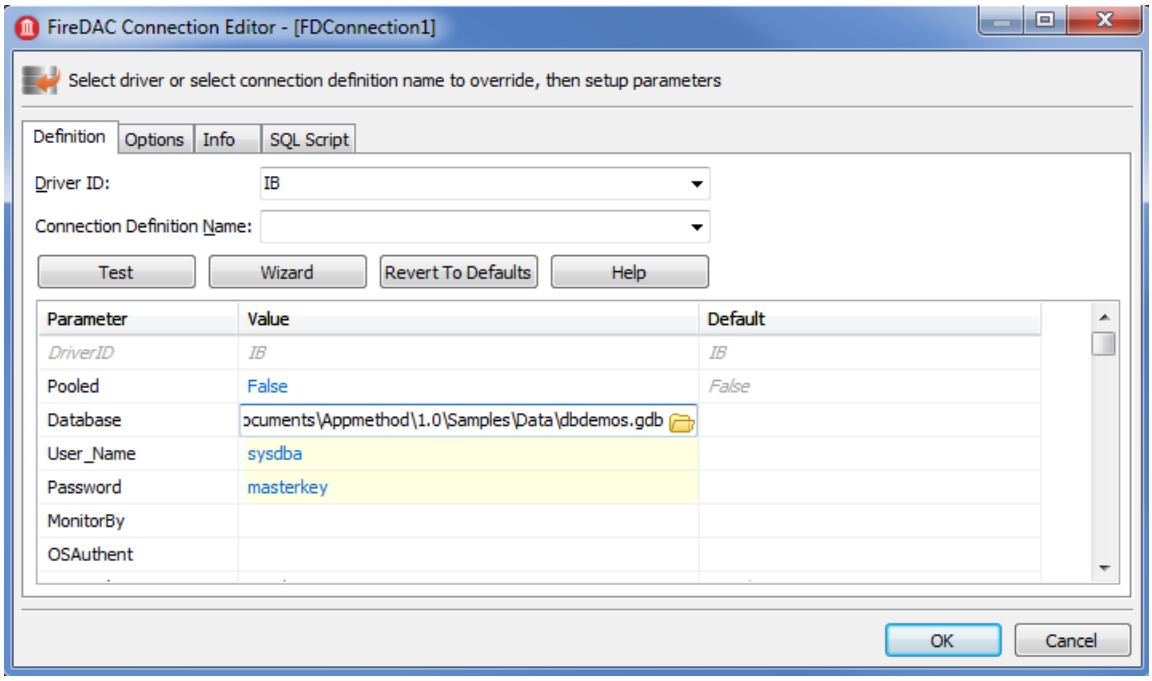



2. Right-click the TFDConnection component and choose **Connection Editor**.
3. In the FireDAC Connection Editor, set the following parameters of the TFDConnection:
    1. Set the Driver ID property to **IB**.
    2. Set the **Database** parameter to:

`C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\dbdemos.gdb` (location of the database)

and click **Open** in the File Open dialog box.

    3. Set the **User_name** parameter to **sysdba**.
    4. Set the **Password** parameter to **masterkey**.
    5. Set the **Protocol** parameter to **TCPIP**.

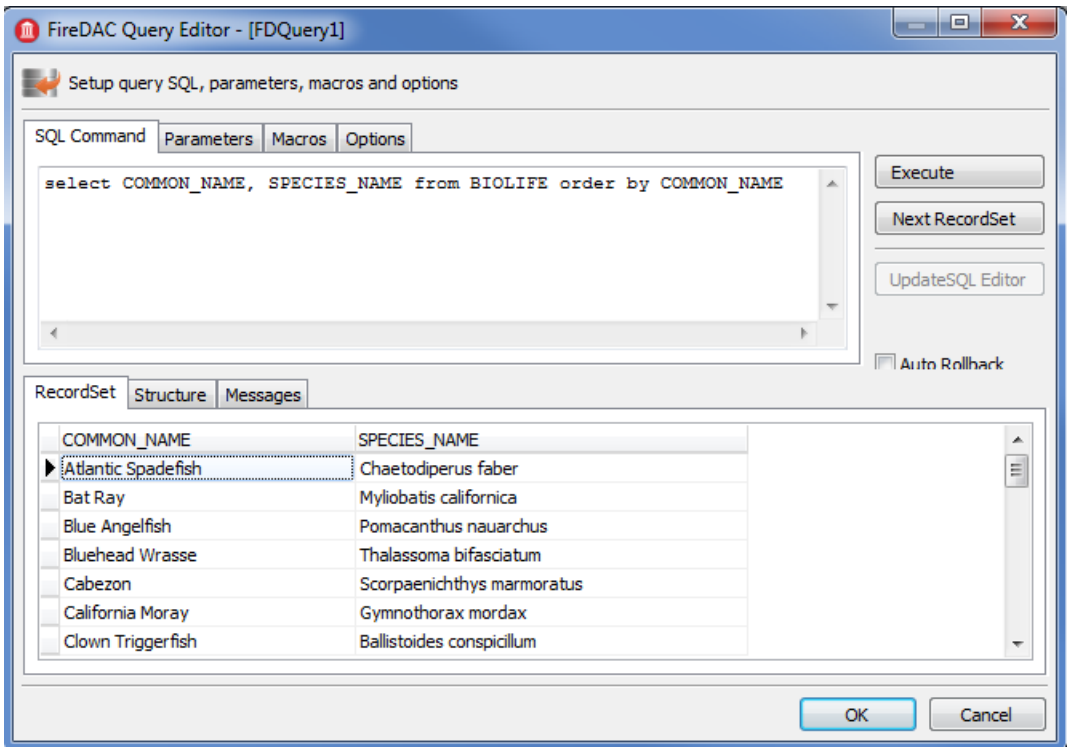



    6. Click the **Test** button to test the connection.
    7. Click **OK** to close the Connection Editor.
2. In the Object Inspector, set the following properties of **TFDConnection**:

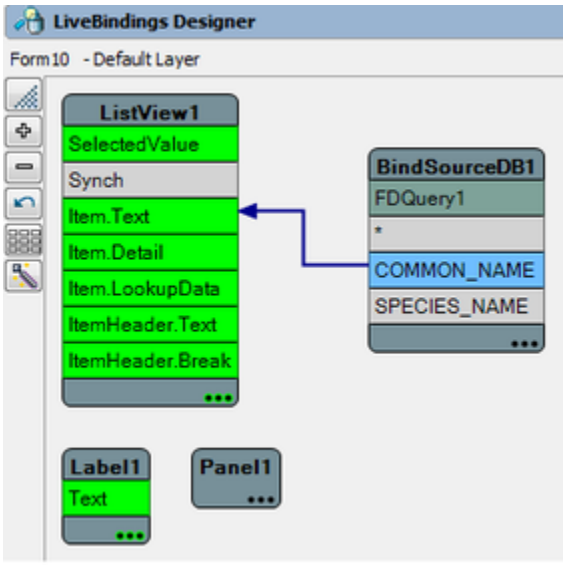
1. Set the LoginPrompt property to **False**, so that the user is not prompted for a login.
2. Set the Connected property to **True**.

**Note:** If you get an error ("unavailable database") in the development environment, this means you do not have a current license for InterBase. The license of InterBase Developer Edition is included as part of the product for some product editions. For more information, see Troubleshooting.

3. Add a TFDQuery component to the form.
4. Right-click the TFDQuery component and choose **Query Editor**.
    1. Write in the **SQL Command** Text editor `select COMMON_NAME, SPECIES_NAME from BIOLIFE order by COMMON_NAME`.
    2. Click the **Execute** button to see the command results.

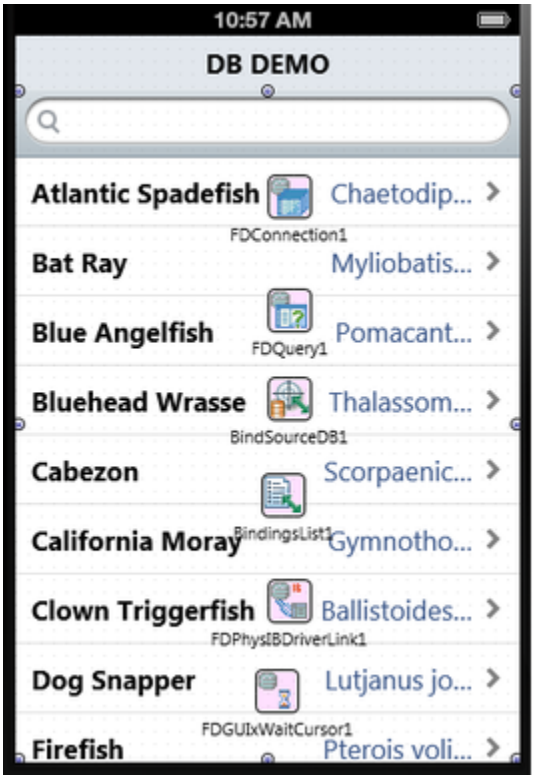


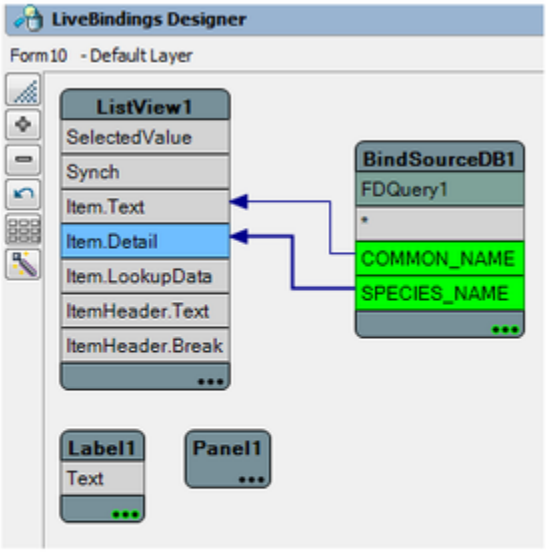

    3. Click **OK** to close the Query Editor.
5. In the Object Inspector, set the Active property of the TFDQuery component to **True**.
6. Open the LiveBindings Designer and connect the data and the user interface as follows:
    1. Click **COMMON_NAME** in FDQuery1, and drag the mouse cursor to **Item.Text** in ListView1.




At this point, TBindSourceDB and TBindingsList components were added to the form.

2. Click **SPECIES_NAME** in BindSourceDB1, and drag the mouse cursor to **Item.Detail** in ListView1.




7. Add a TFDPhysIBDriverLink component to the form.
8. Add a TFDGUIxWaitCursor component to the form.

**Note:** The Preparing a FireDAC Application for Run Time topic explains the use of the TFDGUIxWaitCursor and TFDPhysIBDriverLink components in a FireDAC application.

# Deploying your Application to Mobile

Up to this point, you have used InterBase on your desktop. This means that the actual database is located at your local hard disk drive (for example, `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\dbdemos.gdb`). On the mobile Device, the application is sand-boxed, and typically you can only read and write data that is located in the **Documents** folder (for iOS device) and **internal** storage (for Android device) under your application folder.

To connect to a local database on mobile, you need to perform the following actions:

- Deploy the database to the mobile device.
- Check the configuration (to connect to the database file) to a local file under the **Documents** folder (for iOS device) or **internal** storage (for Android device).

## Deploying InterBase ToGo Required Files and the Database File to Mobile

To execute your application on mobile, you need to deploy the following files:

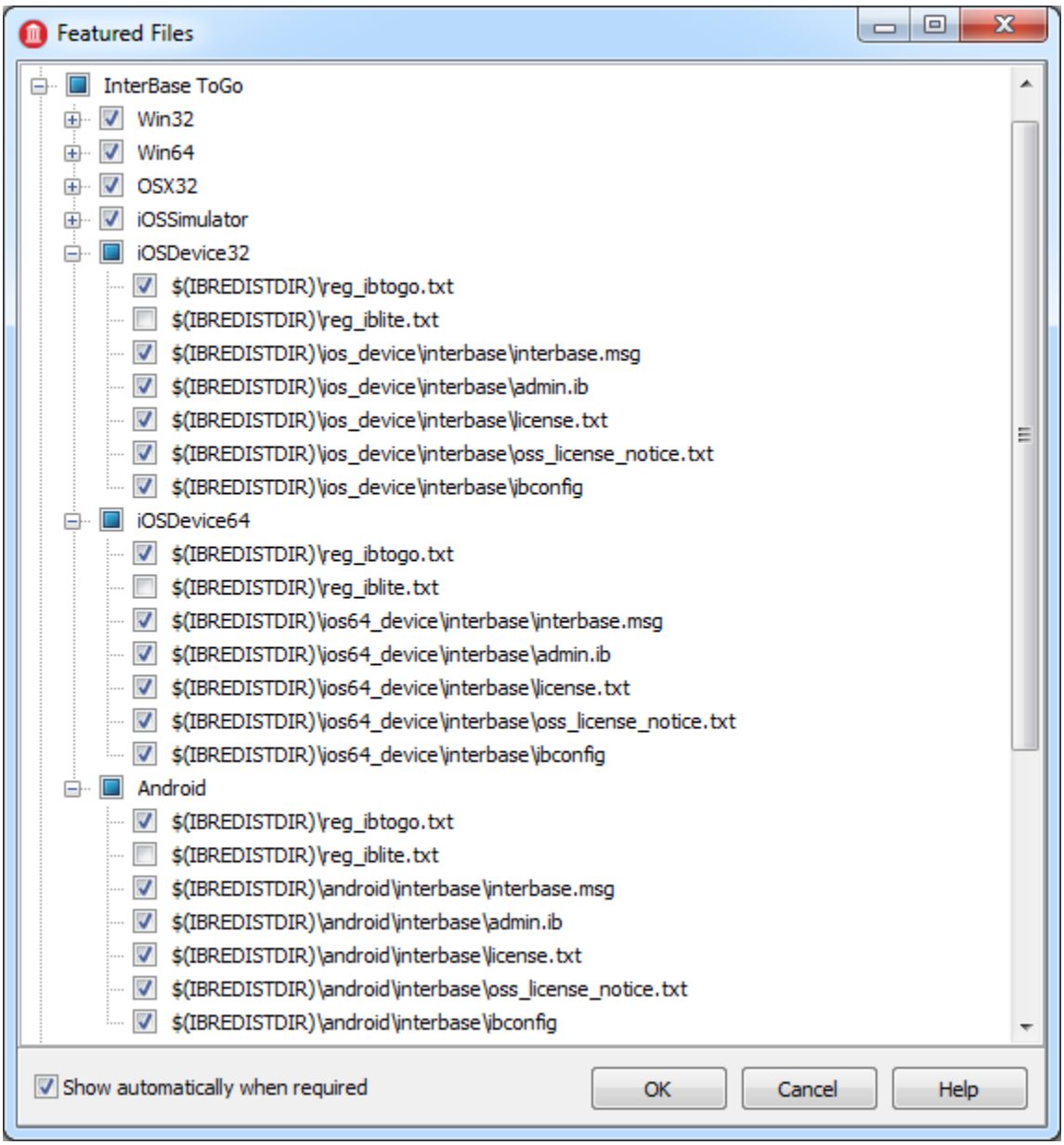
- Interbase ToGo required file (the license file and other configuration files)
- The database file (dbdemos.gdb)

Deploy these files to your application as follows:

1. You can add the database to your project with one of the following two methods:
    - Right-click the project name in the **Project Manager** and select **Add...** from the context menu (or **Project > Add to Project**) to display the Add to Project dialog box. Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data`, select the database `dbdemos.gdb` and click **Open**.
    - Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data` and drag and drop the database `dbdemos.gdb` to the project in the Project Manager. Click **Yes** to confirm that you want to add the file to your project.
2. After adding the database file, the **Featured Files** window displays. Select **InterBase ToGo** in the **Feature Files**, and then click **OK** to close the Featured Files dialog box.
    - Under the node **InterBase ToGo** you need to select the license to be used when deploying the application on the device.
        - The **Tip** at the beginning of this tutorial describes how to activate an InterBase license.
        - The suggested names for the license files available are listed in the Featured Files dialog, under the following name pattern: **reg_*.txt**.

As you can see in the image below, the **reg_ibtogo.txt** license file is selected for this tutorial.

- You might have received from Embarcadero a license file for IBToGo or IBLite that has a pattern of **reg_nnnnnnn.txt**, where **nnnnnnn** is a generated number:
  - If you have saved that file over **reg_ibtogo.txt** or **reg_iblite.txt** in the location below (for example, `C:\Users\Public\Documents\Embarcadero\InterBase\redist\InterBaseXE7`), you can just select the desired license.
  - If you have saved the file with its original name, then select **Add Files** (shown in the next step) and include the license file in the list of files that need to be deployed with the application.



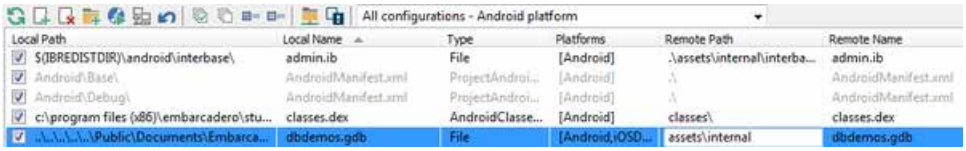

3. Open the Deployment Manager by selecting **Project > Deployment**.

4. Select **Debug configuration - iOS Device - 32 bit platform**, **Debug configuration - iOS Device - 64 bit platform** or **Debug configuration - Android platform** from the drop-down list of target platforms at the top of the Deployment Manager and see that the database `dbdemos.gdb` has been added to the platforms.

5. See how the **Remote Path** of `dbdemos.gdb` has been set for iOS and Android platforms:

    - **Remote Path** on iOS Device platform: `StartUp\Documents\`


    - **Remote Path** on Android platform: `assets\internal\`




As you just configured, when you run the app on the mobile device, the database file (dbdemos.gdb) is to be deployed to the **Documents** folder (for iOS platform) or **internal** storage (for Android platform) in the sandbox area of your multi-device application.

For the TFDConnection, create the BeforeConnect event handler as follows:

1. In the Form Designer, select the TFDConnection component.
2. In the Object Inspector, go to the Events tab and double-click the BeforeConnect event.
3. Implement the event handler for the BeforeConnect event by adding the following code:

    - For **Delphi:**

```
procedure TForm1.FDConnection1BeforeConnect(Sender: TObject);
begin
  {$IF DEFINED(iOS) or DEFINED(ANDROID)}
   FDConnection1.Params.Values['Protocol'] := 'Local';
   FDConnection1.Params.Values['Database'] :=
TPath.Combine(TPath.GetDocumentsPath, 'dbdemos.gdb');
  {$ENDIF}
end;
```

The TPath record is declared in **System.IOUtils** unit, so you need to add **System.IOUtils** in the uses clause, as follows:

```
implementation

{$R *.fmx}
{$R *.NmXhdpiPh.fmx ANDROID}
{$R *.iPhone4in.fmx IOS}
```

```
uses System.IOUtils;

procedure TForm1.FDConnection1BeforeConnect(Sender: TObject);
//  previous code goes here
```

- For **C++**:

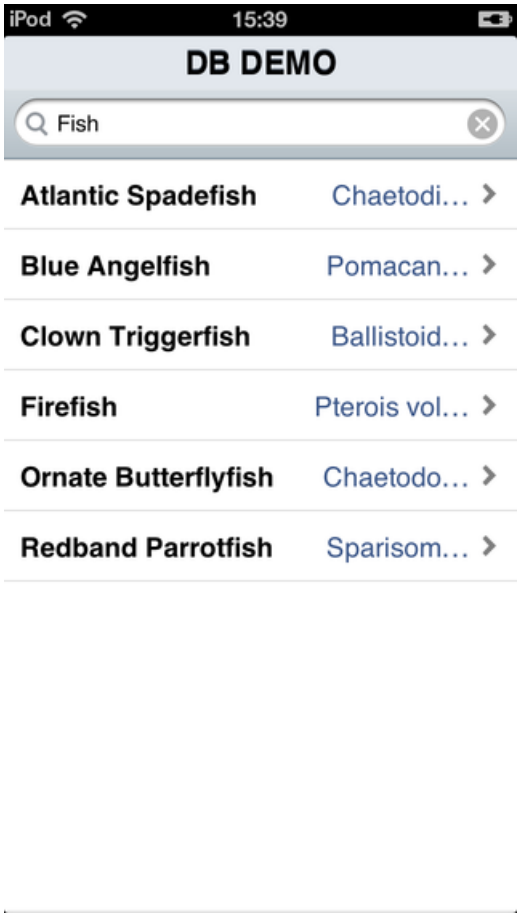
```
void __fastcall TForm1::SQLConnection1BeforeConnect(TObject *Sender)
{
  #if defined(_PLAT_IOS) || defined(_PLAT_ANDROID)
      FDConnection1->Params->Values["Protocol"] = "Local";
      FDConnection1->Params->Values["Database"] =
System::Ioutils::TPath::Combine(System::Ioutils::TPath::GetDocumentsPath(),
"dbdemos.gdb");
  #endif
}
```

You need to add **#include <System.IOUtils.hpp>** in the unit (.cpp file).

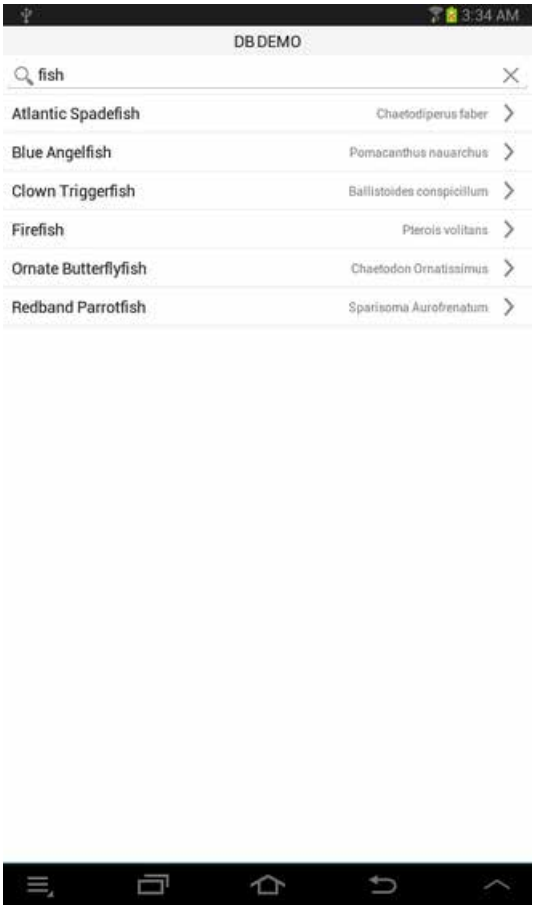
# Run Your Application on a Simulator or on a Mobile Device

Now your application is ready to run (**Run > Run** or press F9). You should be able to browse data just as you can in the IDE. You can narrow down the list using the Search Box.

iOS




Android




# Troubleshooting

## InterBase Issues

See the following section with detailed information about Interbase License Issues.

**Note**: Follow the steps at IBLite and IBToGo Test Deployment Licensing to obtain a valid license file.

## Exception Handling Issues

If your application raises an exception without having proper exception handling code, your multi-device application simply crashes (disappears) at run time.

If you encounter a crash, you might want to connect manually to the database while you troubleshoot the issue using the following steps:

1. Select the **FDConnection1** component, and change the Connected property to **False**.

2. Drop a button on the form, and create the following event handler to manually connect to the database:

   **Delphi:**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    FDConnection1.Connected := True;
    FDQuery1.Active := True;
  except
    on e: Exception do
    begin
      ShowMessage(e.Message);
    end;
  end;
end;
```

   **C++:**

```
void __fastcall TForm1::Button1Click(TObject *Sender) {
    try {
        FDConnection1->Connected = true;
        FDQuery1->Active = true;
    }
    catch(Exception &e) {
        ShowMessage(e.Message);
    }
}
```

3. Check the error message.

# See Also

- InterBase ToGo
- IBLite and IBToGo Licensing
- Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)
- http://www.embarcadero.com/products/interbase/product-editions
- Mobile Tutorial: Using FireDAC and SQLite (iOS and Android)
- Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)

## Samples

- FireDAC InterBase sample

# Mobile Tutorial: Using dbExpress and SQLite (iOS and Android)

**Caution:** dbExpress, which is described in this tutorial, is being **deprecated**. This means that dbExpress will be removed from RAD Studio in an upcoming release.

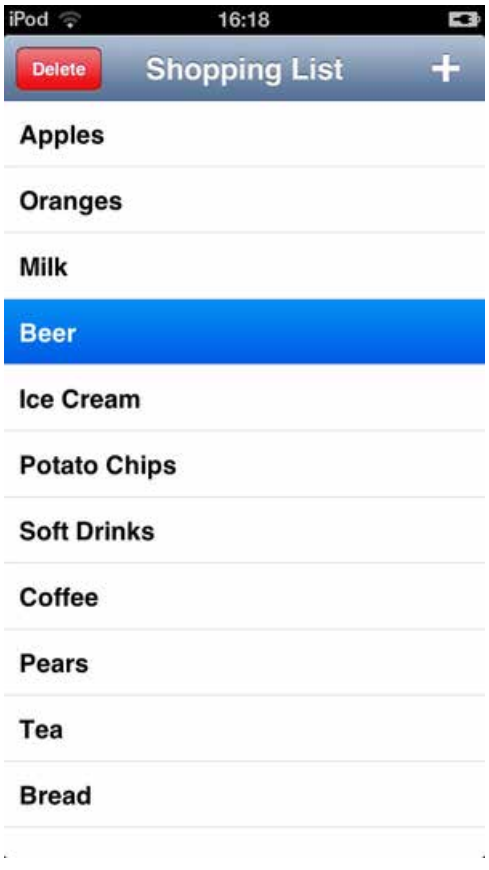
Instead of dbExpress, we recommend that you use our newer database solution, **FireDAC**, which is described in a similar tutorial, here:
Mobile Tutorial: Using FireDAC and SQLite (iOS and Android).

Before starting this tutorial, you should read and perform the following tutorial session:

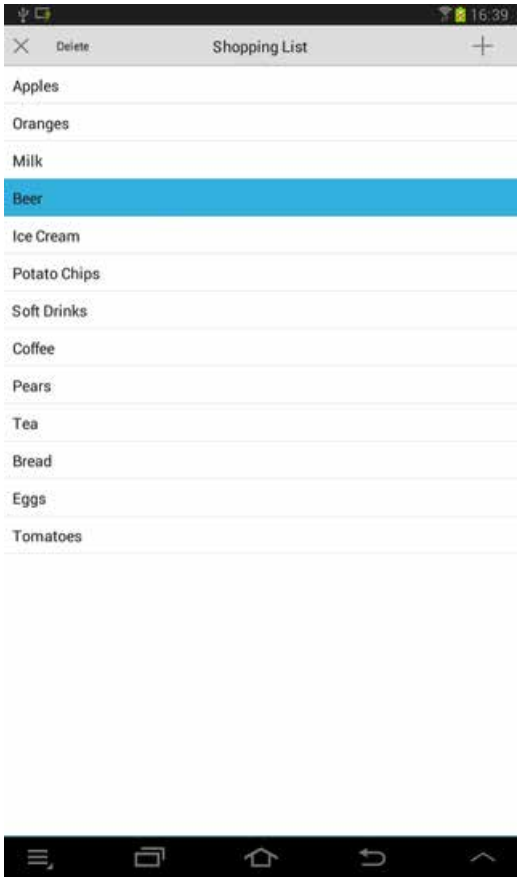
- Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

This tutorial describes the basic steps to use SQLite as a local data storage on your mobile device through the dbExpress framework.

iOS




Android




# Using dbExpress to Connect to the Database

dbExpress is a very fast database access framework, written in Delphi. RAD Studio provides drivers for most major databases, such as InterBase, Oracle, DB2, SQL Server, MySQL, Firebird, SQLite, and ODBC. You can access these different databases using procedures similar to the procedure described here.

- For the mobile platforms, dbExpress supports **InterBase ToGo** as well as **SQLite**. These database products can run on iOS and Android devices.
- For other databases, such as Oracle, you need to have at least a client library. On Windows platforms, the client library is provided as a DLL to connect to. Therefore, you need to develop applications using middle-tier technologies such as DataSnap to connect to these database products from a mobile device.

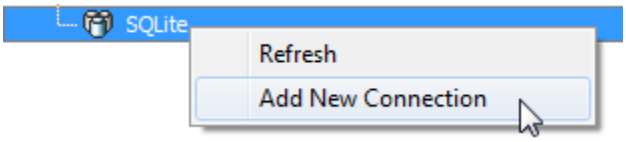
Another tutorial discusses how to connect to Enterprise Database without using a client library on a mobile device; see Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android).

# Creating the Database in the Windows Environment for Development Purposes

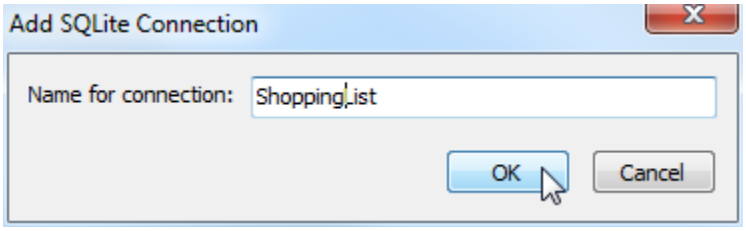
First, you need to create a SQLite database file on your Windows development platform. Use the following steps, so that you can use the Form Designer to design the user interface of your application.

## Create the Database in the Data Explorer

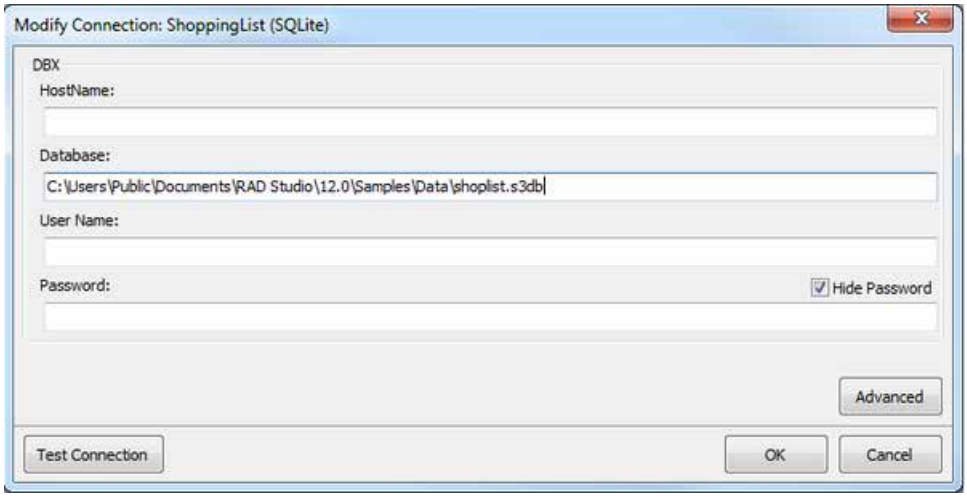
1. Go to Data Explorer, right-click the **SQLite** node and select **Add New Connection**:



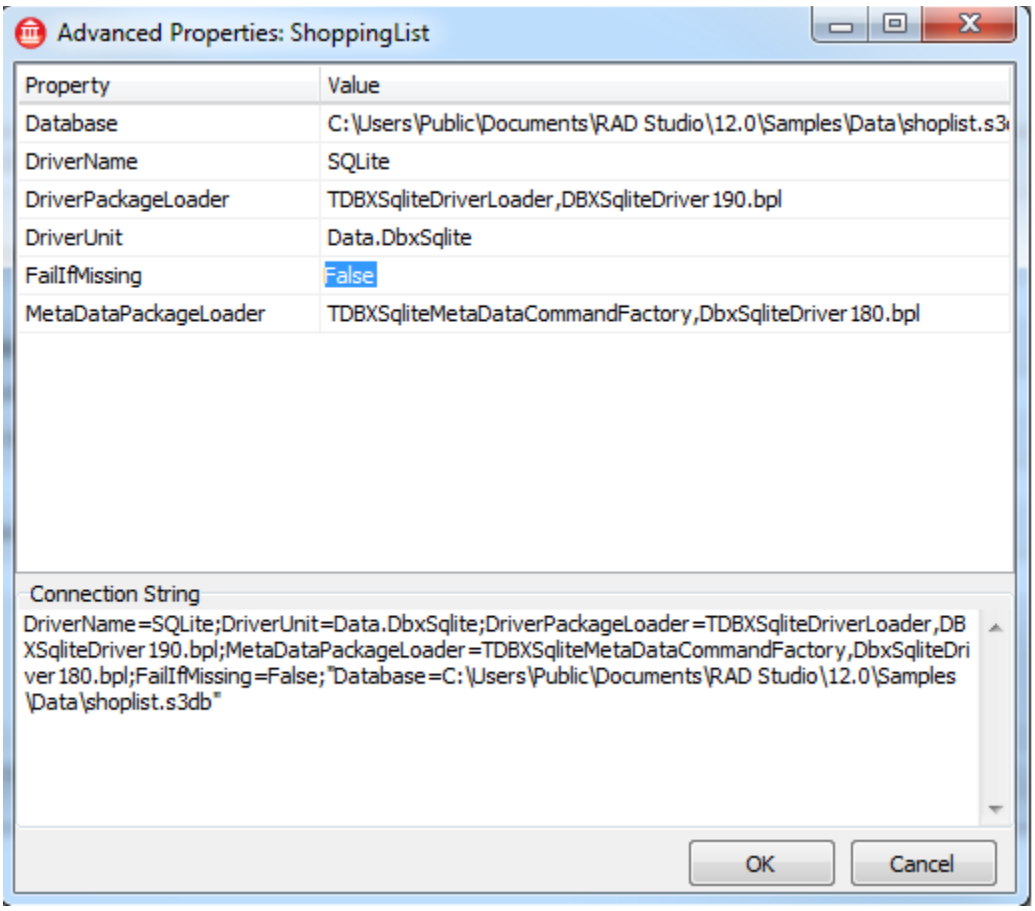

2. Define the name of the connection, such as **ShoppingList**.




3. Specify the location of the database file:

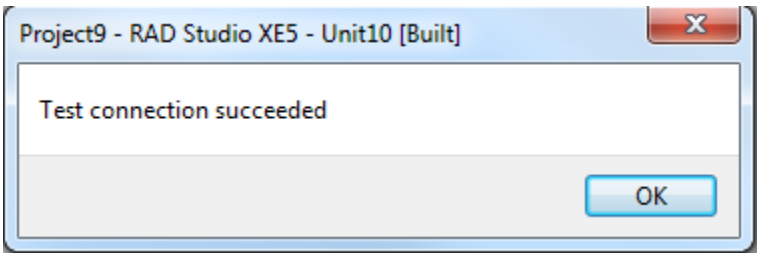



4. Click the **Advanced** button and open the **Advanced Properties** dialog box.
5. Change the **FailIfMissing** property to **False** and click OK to close the **Advanced Properties** dialog box:




**Note:** Setting **FailIfMissing** to **False** instructs the Data Explorer to create a new database file if the file is not available.

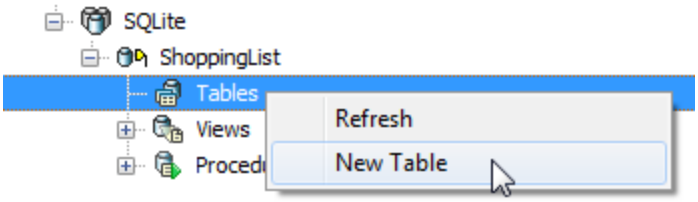
6. Back on the **Modify Connection** dialog box, click the **Test Connection** button. With this operation, the new database file is created if no file existed:



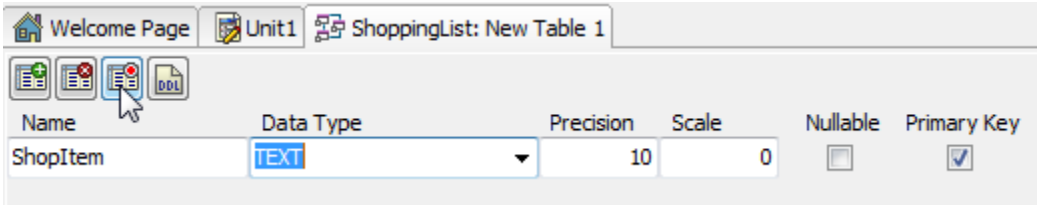

**Note:** Ensure that **sqlite3.dll** is present on your development system. If this file is not present, download sqlite3.dll from http://www.sqlite.org/download.html to your system path (such as **C:\Windows\SysWOW64** for 64-bit Windows).

## Create Table on DataExplorer

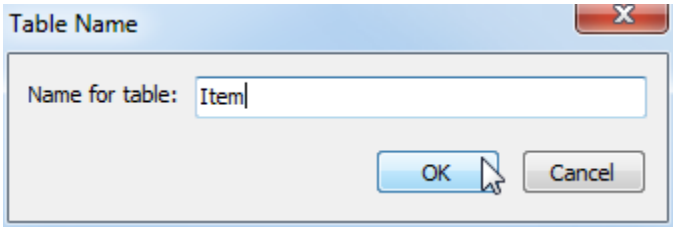
1. On the Data Explorer, double-click the **ShoppingList** node under the SQLite section, right-click **Tables**, and then select **New Table** from the context menu.




2. Set **Data Type** for a **ShopItem** column to **TEXT**.




3. Click the **Save** button and specify a table name (for example, **Item**.)



# Design and Set Up the User Interface

Visible UI components are loaded on the designer

This tutorial uses one TListBox component as the UI element.

**To set up a ListBox component and other UI elements, use the following steps:**

1. Create a multi-device application using **File > New > Multi-Device Application - Delphi** or **File > New > Multi-Device Application - C++Builder**.
2. Drop a TToolBar on the form.
3. Drop a **TButton** on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Align to **Right**.
   - Set the Name property to **ButtonAdd**.
   - Set the StyleLookup to **addtoolbutton**.
4. Drop a **TButton** on the ToolBar component and set the following properties in the Object Inspector:

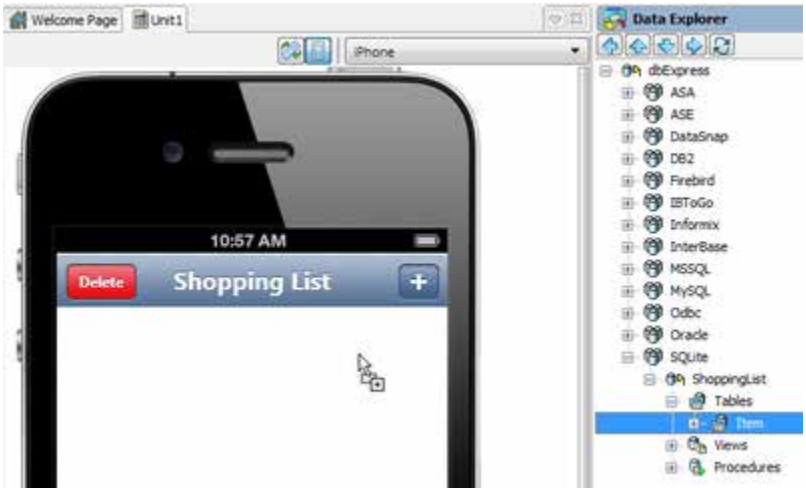
- Set the Align to **Left**.
- Set the Name property to **ButtonDelete**.
- Set the StyleLookup to **deletetoolbutton**.
- Set the Text to **Delete**.
- Set the Visible to **False**.

5. Drop a **TLabel** on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Align to **Client**.
   - Set the StyleLookup to **toollabel**.
   - Set the Text to **Shopping List**.
   - Set the TextSettings.HorzAlign to **Center**.
6. Drop a **TListBox** component on the form and set the following properties in the Object Inspector:
   - Set the Align property to **Client**, so that the ListBox component uses the entire form.

# Connecting to the Data

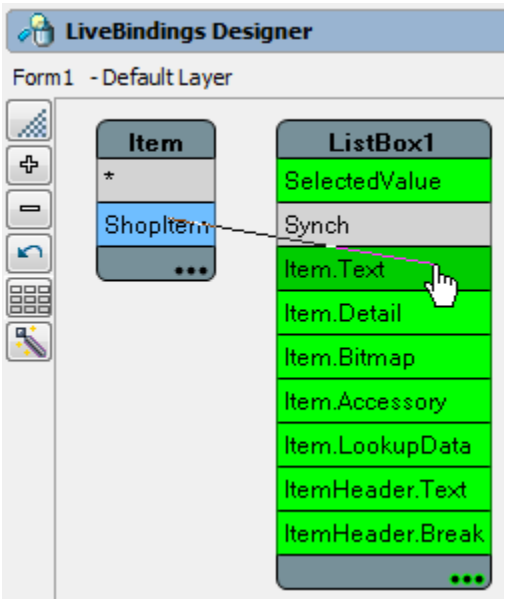
Following are the basic steps to connect to data in a database which is already defined in the Data Explorer:

1. Select the **Item** table on the Data Explorer and drag it to the Form Designer.




**Note:** This creates two components (ShoppinglistConnection: TSQLConnection and ItemTable: TSQLDataSet) on the form.


2. Select the **ShoppinglistConnection** component on the form, and then change the Connected property to **True**.
3. Select the **ItemTable** component on the form, and then change the Active property to **True**.
4. Select **View > LiveBindings Designer** and the LiveBindings Designer opens.
5. Select **ShopItem** in the **ItemTable** component and drag **ShopItem** to **Item.Text** of **ListBox1**.




Following these steps connects the user interface of the app with data on a SQLite database. If you used a table with existing data for this tutorial, now you should see actual data within the Form Designer.

# Creating the Event Handler to Make the Delete Button Visible When the User Selects an Item from the List

The Visible property for the **Delete** button is set to **False**. Therefore, by default, the end user does not see this button. You can make it visible when the user selects an item on the list, as follows:

- Select **ListBox1** and define the following event handler for the OnItemClick event.

**Delphi:**

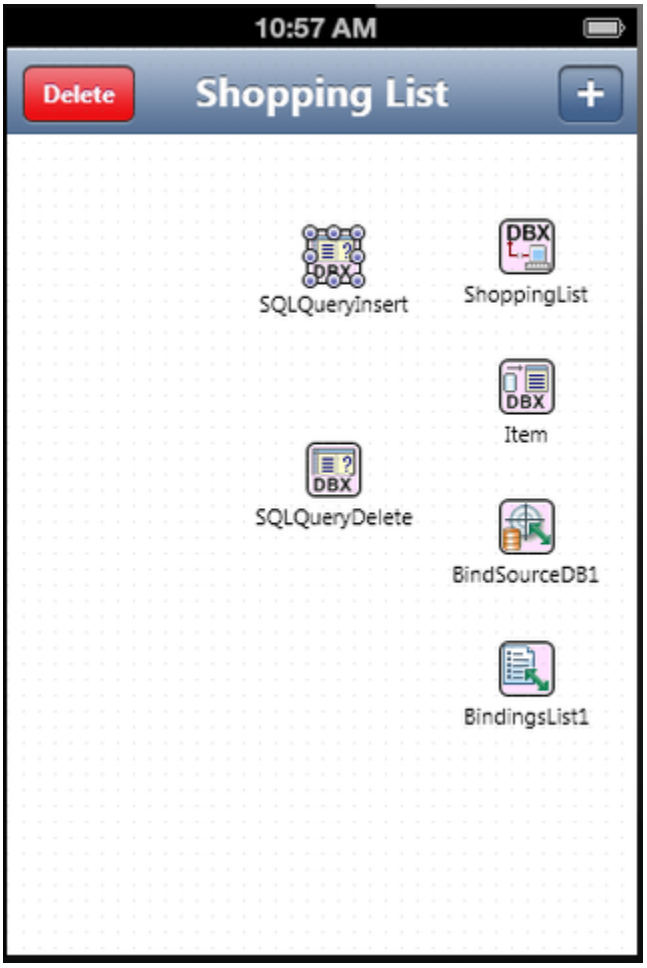
```
```

```
procedure TForm1.ListBox1ItemClick(const Sender: TCustomListBox;
  const Item: TListBoxItem);
begin
  if ListBox1.Selected <> nil then
    ButtonDelete.Visible := True
  else
    ButtonDelete.Visible := False;
end;
```

### C++Builder:

```
void __fastcall TForm1::ListBox1ItemClick(const TCustomListBox *Sender,const
TListBoxItem *Item)
{
        if (ListBox1->Selected)
                ButtonDelete->Visible = True;
        else
                ButtonDelete->Visible = False;
}
```

# Creating the Event Handler for the Add Button to Add an Entry to the List



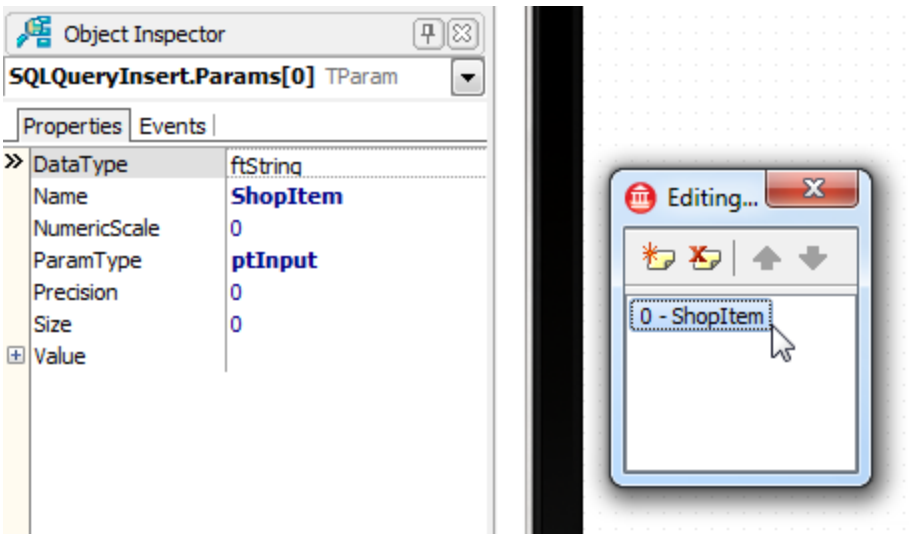

Database connections are also configured

The next step is adding a feature to this application for adding an item to the shopping list.

1. Drop a TSQLQuery component to the form.
2. Set the following properties in the Object Inspector:
    - Set the Name property to **SQLQueryInsert**.
    - Set the SQLConnection property to **ShoppinglistConnection**.
    - Set the SQL property as follows:

```
INSERT INTO ITEM (ShopItem) VALUES (:ShopItem)
```

    - Select the **Expand** (...) button on the Params property.
    - Select the **ShopItem** parameter and set DataType to **ftString**:




3. In the Form Designer, double-click the **ButtonAdd** component. Add the following code to this event handler:

**Delphi:**

```
procedure TForm1.ButtonAddClick(Sender: TObject);
begin
  TDialogServiceAsync.InputQuery('Enter New Item', ['Name'], [''],
Self.OnInputQuery_Close);
end;

procedure TForm1.OnInputQuery_Close(const AResult: TModalResult; const AValues: array
of string);
var
  TaskName: string;
begin
  TaskName := string.Empty;
  if AResult <> mrOk then
    Exit;
  TaskName := AValues[0];
  try
    if not (TaskName.Trim = string.Empty) then
    begin
      SQLQueryInsert.ParamByName('ShopItem').AsString := TaskName;
      SQLQueryInsert.ExecSQL();
      ItemTable.Refresh;
      LinkFillControlToField1.BindList.FillList;
      if ListBox1.Selected <> nil then
        ButtonDelete.Visible := True
      else
        ButtonDelete.Visible := False;
    end;
  except
    on Ex: Exception do
      ShowMessage('Error: ' + Ex.Message);
  end;
end;
```

Declare this procedure prototype under the private section of the Form class:

```
private
    procedure OnInputQuery_Close(const AResult: TModalResult; const AValues: array of
string);
```

### C++Builder:

To replicate the same functionality in C++, additional steps are required:

1. Add the next type definition after the TForm1 definition:

```
typedef void __fastcall (__closure *TInputCloseQueryProcEvent)(const
System::Uitypes::TModalResult AResult,
        System::UnicodeString const *AValues, const int AValues_High);
```

2. Add the next class definition:

```
class InputQueryMethod : public TCppInterfacedObject<TInputCloseQueryProc>
{
private:
    TInputCloseQueryProcEvent Event;

public:
    InputQueryMethod(TInputCloseQueryProcEvent _Event) {
        Event = _Event;
    }

    void __fastcall Invoke(const System::Uitypes::TModalResult AResult,
        System::UnicodeString const *AValues, const int AValues_High) {
        Event(AResult, AValues, AValues_High);
    }
};
```

3. Add the next declaration under the private section of the form:

```
void __fastcall OnInputQuery_Close(const System::Uitypes::TModalResult AResult,
    System::UnicodeString const *AValues, const int AValues_High);
```

4. Add the actual functions:

```
void __fastcall TForm1::ButtonAddClick(TObject *Sender) {
    String caption = "Caption";
    String Prompts[1];
    Prompts[0] = "Prompt 0";
    String Defaults[1];
    Defaults[0] = "Default 0";
    _di_TInputCloseQueryProc Met = new InputQueryMethod(&OnInputQuery_Close);
    TDialogServiceAsync::InputQuery(
```

```
        "caption", Prompts, 0, Defaults, 0, (TInputCloseQueryProc *)Met);
}
void __fastcall TForm1::OnInputQuery_Close(const System::Uitypes::TModalResult
AResult,
        System::UnicodeString const *AValues, const int AValues_High) {
    String TaskName;
    TaskName = "";
    if (AResult != mrOk)
        return;
    TaskName = AValues[0];
    try {
        if (TaskName.Trim() != "")
            SQLQueryInsert->ParamByName("ShopItem")->AsString = TaskName;
        SQLQueryInsert->ExecSQL();
        ItemTable->Refresh();
        LinkFillControlToField1->BindList->FillList();
        if (ListBox1->Selected != NULL)
            ButtonDelete->Visible = True;
        else
            ButtonDelete->Visible = False;
    }
    catch (Exception& Ex) {
        ShowMessage("Error: " + Ex.Message);
    }
}
```

The InputQuery function shows a dialog box asking the end user to enter a text. This function returns **True** when the user selects **OK**, so that you can add data to the database only when the user selects **OK** and the text contains some data.

iOS


Android


# Creating the Event Handler for the Delete Button to Remove an Entry from the List

The next step is adding a feature to this application to remove an item from the shopping list:

1. Drop a TSQLQuery component to the form.
2. Set the following properties in the Object Inspector:
   - Set the Name property to **SQLQueryDelete**.
   - Set the SQLConnection property to **ShoppinglistConnection**.
   - Set the SQL property as follows:

```
delete from Item where ShopItem = :ShopItem
```

   - Select the **Expand** (...) button on the Params property.
   - Select the **ShopItem** parameter and set DataType to **ftString**.
3. In the Structure View, select the **ButtonDelete** component. Add the following code to this event handler.

**Delphi:**

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
var
  TaskName: String;
begin
  TaskName := ListBox1.Selected.Text;

  try
    SQLQueryDelete.ParamByName('ShopItem').AsString := TaskName;
    SQLQueryDelete.ExecSQL();
    ItemTable.Refresh;
    LinkFillControlToField1.BindList.FillList;
    if ListBox1.Selected <> nil then
      ButtonDelete.Visible := True
    else
      ButtonDelete.Visible := False;
  except
    on e: Exception do
    begin
      SHowMessage(e.Message);
    end;
  end;
end;
```

**C++Builder:**

```
void __fastcall TForm1::ButtonDeleteClick(TObject *Sender) {
        String TaskName = ListBox1->Selected->Text;
        try {
                SQLQueryDelete->ParamByName("ShopItem")->AsString = TaskName;
                SQLQueryDelete->ExecSQL();
                ItemTable->Refresh();
                LinkFillControlToField1->BindList->FillList();
                if (ListBox1->Selected)
                        ButtonDelete->Visible = True;
                else
                        ButtonDelete->Visible = False;
        }
        catch (Exception &e) {
                ShowMessage(e.Message);
        }
}
```

# Setting Up Your Database Deployment for Mobile Platforms

Up to this point, you have used SQLite on your desktop. This means that the actual database is located on your local hard disk drive (for example, C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\shoplist.s3db). On the mobile device, applications are sand-boxed, and typically you can only read and write data that is located in the **Documents** folder (for iOS device) and **internal** storage (for Android device) under your application folder.

To connect to a local database on mobile, you need to perform the following actions:

- Deploy the database to the mobile device.
- Check the configuration (to connect to the database file) to a local file under the **Documents** folder (for iOS device) or **internal** storage (for Android device).

## Add and Configure Your Database File in the Deployment Manager

Before you can run your application on mobile, you need to set up the deployment for your database file (shoplist.s3db).

1. You can add the database to your project with one of the following two methods:
   - Right-click the project name in the **Project Manager** and select **Add...** from the context menu (or **Project > Add to Project**) to display the Add to Project dialog box. Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data`, select the database `shoplist.s3db` and click **Open**.
   - Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data` and drag and drop the database `shoplist.s3db` to the project in the Project Manager. Click **Yes** to confirm that you want to add the file to your project.
2. After adding the database file, the **Featured Files** window displays, click **Cancel** to close it.
3. Open the Deployment Manager by selecting **Project > Deployment**.
4. Select **Debug configuration - iOS Device - 32 bit platform**, **Debug configuration - iOS Device - 64 bit platform** or **Debug configuration - Android platform** from the drop-down list of target platforms at the top of the Deployment Manager and see that the database `shoplist.s3db` has been added to the platforms.
5. See how the **Remote Path** of `shoplist.s3db` has been set for iOS and Android platforms:
   - **Remote Path** on iOS Device platform: `StartUp\Documents\`


   - **Remote Path** on Android platform: `assets\internal\`


As you just configured, when you run the app on the mobile device, the database file (shoplist.s3db) is set to be deployed to the **Documents** folder (for iOS platform) or **internal** storage (for Android platform) in the sandbox area of your multi-device application.

# Modifying Your Code to Connect to a Local Database File on Mobile Platforms

The basic features of this application are now implemented. As you worked in the Data Explorer, you created a database file on Windows. The database file is not available on your mobile device unless you copy it to the mobile device or create it on the fly.

You can create a SQLite Database and Table with the following steps:

## Specifying the Location of the SQLite Database on the Mobile Device

1. In the Form Designer, select the **ShoppinglistConnection** component.
2. In the Object Inspector, double-click the BeforeConnect event.
3. Add the following code to this event handler:

**Delphi:**

```
procedure TForm1.ShoppinglistConnectionBeforeConnect(Sender: TObject);
begin
  {$IF DEFINED(iOS) or DEFINED(ANDROID)}
  ShoppinglistConnection.Params.Values['ColumnMetadataSupported'] := 'False';
  ShoppinglistConnection.Params.Values['Database'] :=
      TPath.Combine(TPath.GetDocumentsPath, 'shoplist.s3db');
  {$ENDIF}
end;
```

The TPath record is declared in **System.IOUtils** unit, so you need to add **System.IOUtils** in the uses clause of your unit.

**C++Builder:**

```
void __fastcall TForm1::ShoppinglistConnectionBeforeConnect(TObject *Sender) {
        #if defined(_PLAT_IOS) || defined(_PLAT_ANDROID)
        ShoppinglistConnection->Params->Values["ColumnMetadataSupported"] = "False";
        ShoppinglistConnection->Params->Values["Database"] =
System::Ioutils::TPath::Combine(System::Ioutils::TPath::GetDocumentsPath(),
"shoplist.s3db");
        #endif
}
```

The TPath record is declared in **System.IOUtils** library, so you need to add **#include <System.IOUtils.hpp>** in your header unit.

## Creating a Table if None Exists

With SQLite you can create a table when no table exists, by using the `CREATE TABLE IF NOT EXISTS` statement. You can create a table after the TSQLConnection component connects to

the database and before the TSQLDataSet component connects to the table. Use the following steps:

1. In the Form Designer, select the **ShoppinglistConnection** component.
2. In the Object Inspector, double-click the AfterConnect event.
3. Add the following code to this event handler:

**Delphi:**

```
procedure TForm1.ShoppinglistConnectionAfterConnect(Sender: TObject);
begin
  ShoppinglistConnection.ExecuteDirect('CREATE TABLE IF NOT EXISTS Item (ShopItem
TEXT NOT NULL)');
end;
```

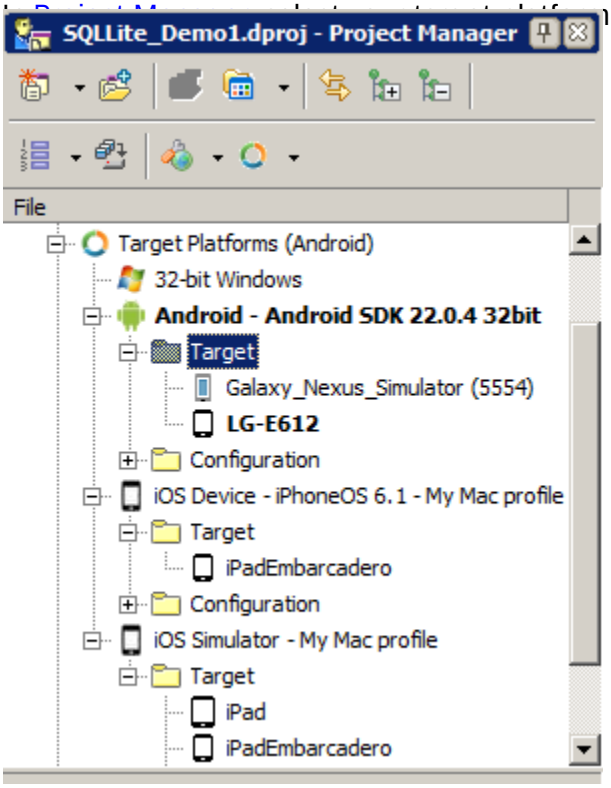
**C++Builder:**

```
void __fastcall TForm1::ShoppinglistConnectionAfterConnect(TObject *Sender){
        ShoppinglistConnection->ExecuteDirect("CREATE TABLE IF NOT EXISTS Item
(ShopItem  TEXT NOT NULL)");
}
```

# Running Your Application on a Mobile Device

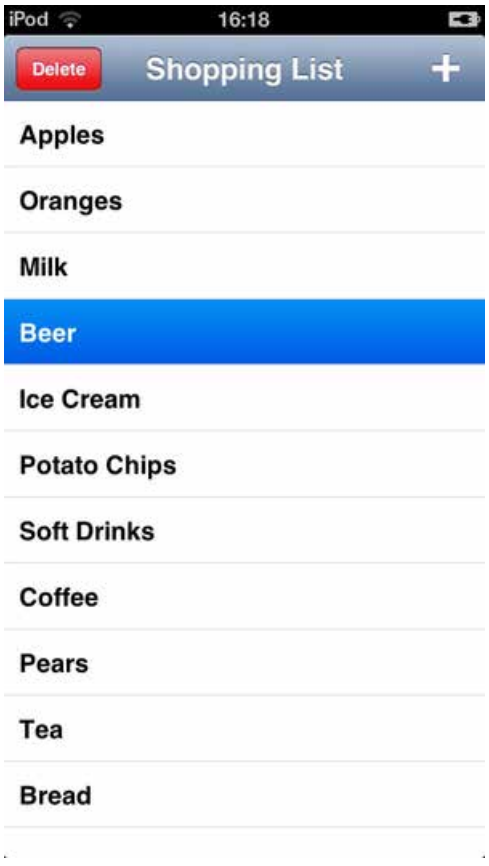
Now your application is ready to run on either a simulator or your connected mobile device.
**To run your application**

1. In Project Manager, select a target platform.

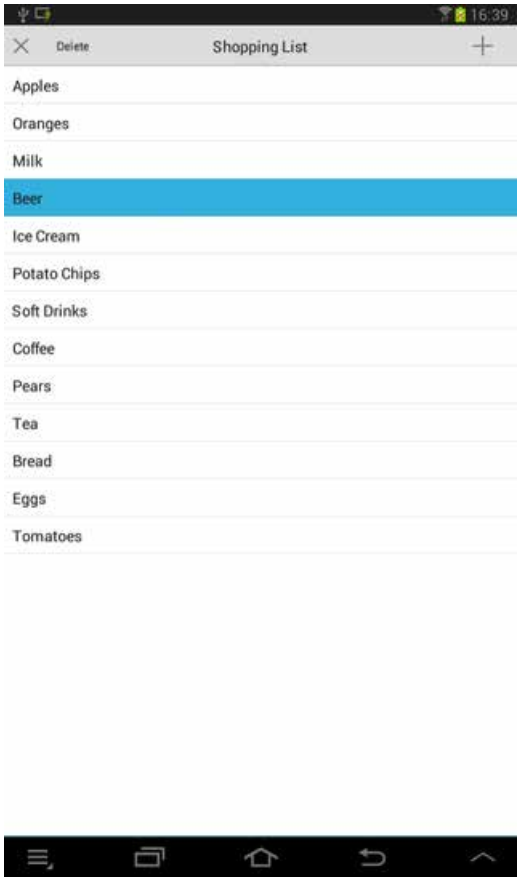



2. Choose either of the following commands:
   - Run > Run
   - Run > Run Without Debugging

iOS




Android




**Note:** If you have an issue with running the application, follow the steps given in **Troubleshooting**.

# See Also

- Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)
- Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)
- SQLite support in RAD Studio
- Android Mobile Application Development
- iOS Mobile Application Development

# Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)

**Caution:** dbExpress, which is described in this tutorial, is being **deprecated**. This means that dbExpress will be removed from RAD Studio in an upcoming release.

Instead of dbExpress, we recommend that you use our newer database solution, **FireDAC**, which is described in a similar tutorial, here:
Mobile Tutorial: Using InterBase ToGo with FireDAC (iOS and Android).

Before starting this tutorial, you should read and perform the following tutorial session:

- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

  **Tip:** Following this tutorial requires a **license** for **IBToGo** or **IBLite**:

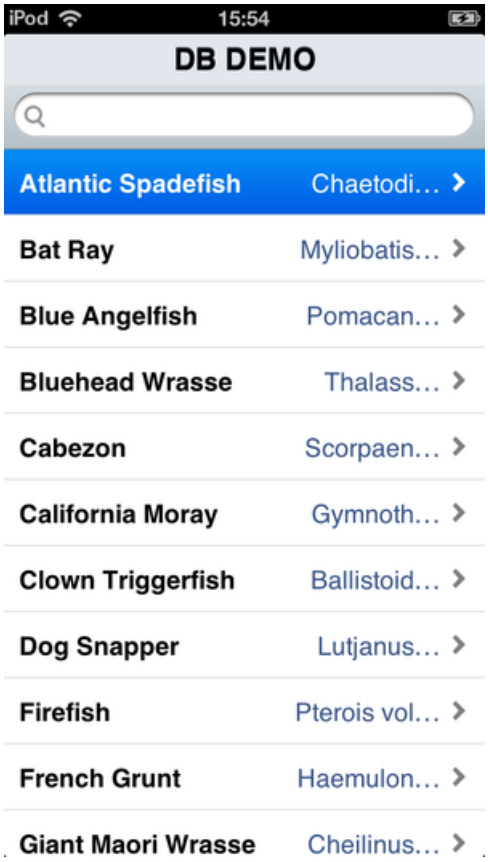
  - If you purchased one of the following RAD Studio versions, you have received in Email a key for an unlimited development and deployment license for IBLite:
    - RAD Studio Tokyo Professional or higherAll Editions
    - Delphi Tokyo Professional or higher with Mobile
  - If you are a trial user, your installation includes a trial license for IBToGo. You can test InterBase on iOS and Android by selecting your test license during the deployment step, as described in this tutorial. The trial licenses are installed with your trial product, in `C:\Users\Public\Documents\Embarcadero\InterBase\redist\InterBaseXE7`.

**Note:** On Android devices, InterBase ToGo apps require specific permissions to be set, specifically:

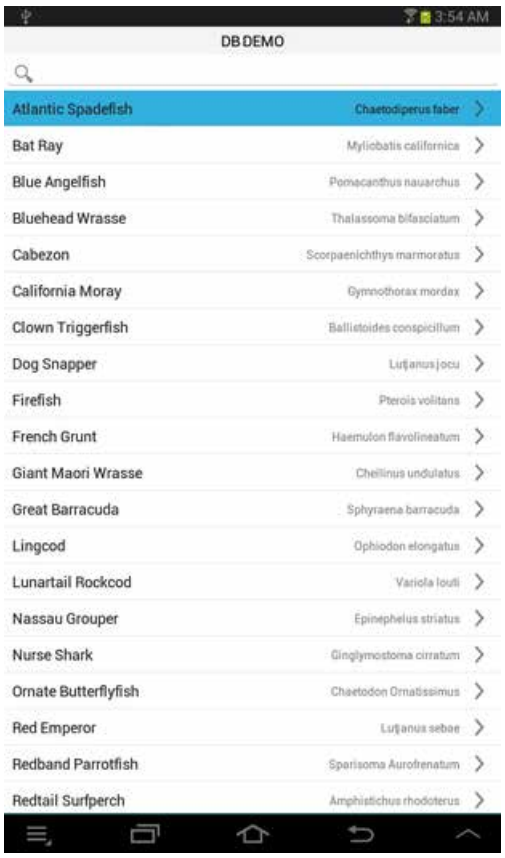
- **Read external storage** (the database is placed in the external memory)
- **Write external storage** (the database is placed in the external memory)
- **Internet** (you need to connect with a remote server)

This tutorial describes the basic steps to browse data managed by InterBase ToGo on your iOS and Android devices through the dbExpress framework.

iOS




Android




**Note:** You can use FireDAC, dbExpress, and Interbase Express (IBX) components to build **Interbase ToGo** applications. For a detailed discussion on Interbase Express components usage in a Delphi application, read the Getting Started with InterBase Express article. For this tutorial, we will connect to **Interbase ToGo** using dbExpress framework.

# Using dbExpress to Connect to the Database

dbExpress is a very fast database access framework, written in Delphi. RAD Studio provides drivers for most major databases, such as InterBase, Oracle, DB2, SQL Server, MySQL, Firebird, SQLite and ODBC. You can access these different databases using procedures similar to the procedure described here.

- For the mobile platforms, dbExpress supports **InterBase ToGo** as well as **SQLite**. These database products can run on iOS and Android devices.
- For other databases, such as Oracle, you need to have at least a client library. On Windows platforms, the client library is provided as a DLL to connect to. Therefore, you need to develop applications using middle-tier technologies such as DataSnap to connect to these database products from a mobile device.

Another tutorial discusses how to connect to Enterprise Database without using a client library on a mobile device; see Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android).

# Design and Set Up the User Interface

This tutorial uses TListView and TPanel components as the UI elements.

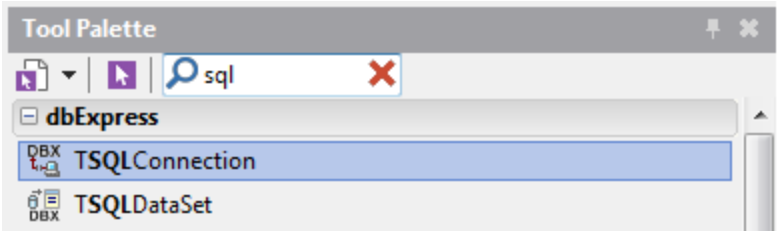
**To set up a ListView and a Panel component, use the following steps:**

1. To create an HD Multi-Device Application, select either of the following:
   - **File > New > Multi-Device Application - Delphi > Blank Application**
   - **File > New > Multi-Device Application - C++Builder > Blank Application**
2. Drop a TListView component on the form.
3. In the Object Inspector, set the following properties of the **ListView**:
   - Set the Align property to **Client**, so that the ListView component uses the entire form.
   - Set the ItemAppearance to **ListItemRightDetail**.
   - Set the SearchVisible to **true**.
4. Add a TPanel component to the form, and set the following properties in the Object Inspector:
   - Set the Align property for the **TPanel** component to **Top**.
5. Add a TLabel component to the Panel, and set the following properties in the Object Inspector:
   - Set the Align property for the **TLabel** component to **Client**.
   - Set the StyleLookup property to **toollabel**.
   - Set the HorzAlign property from TextSettings to **Center**.
   - Set the Text property to **DB DEMO**.

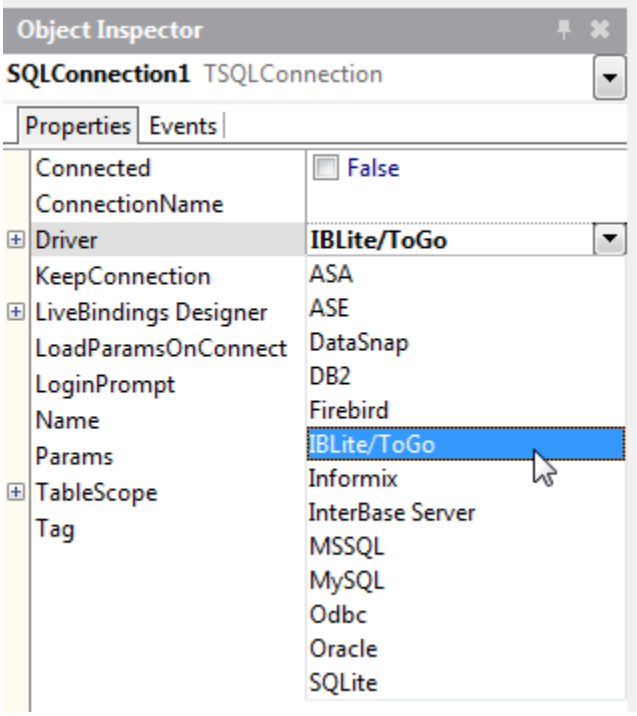
# Connecting to the Data

Following are the basic steps to connect to data in a database using dbExpress:

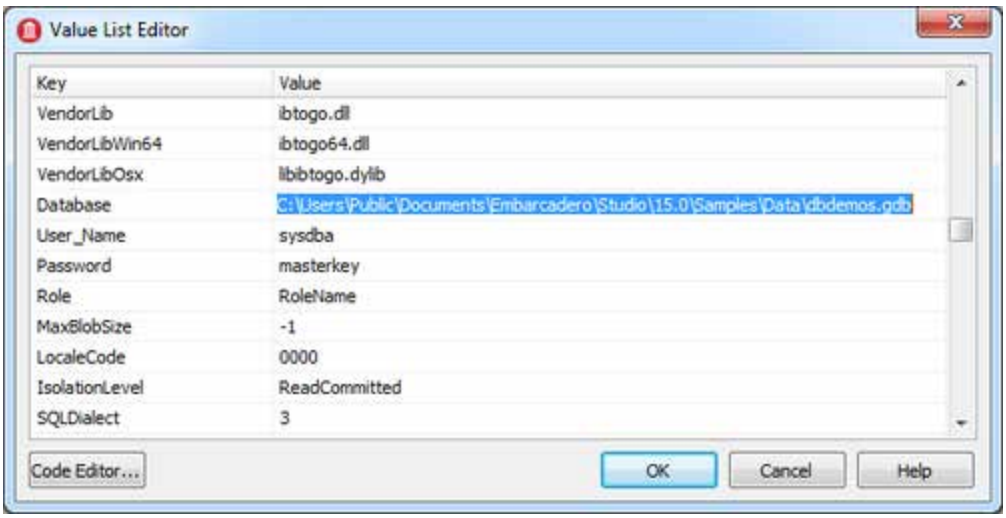
1. On the Tool Palette, double-click the **TSQLConnection** component.




2. In the Object Inspector, set the following properties for **TSQLConnection**:
    1. This app uses InterBase ToGo, so set the **Driver** property to **IBLite/ToGo**.



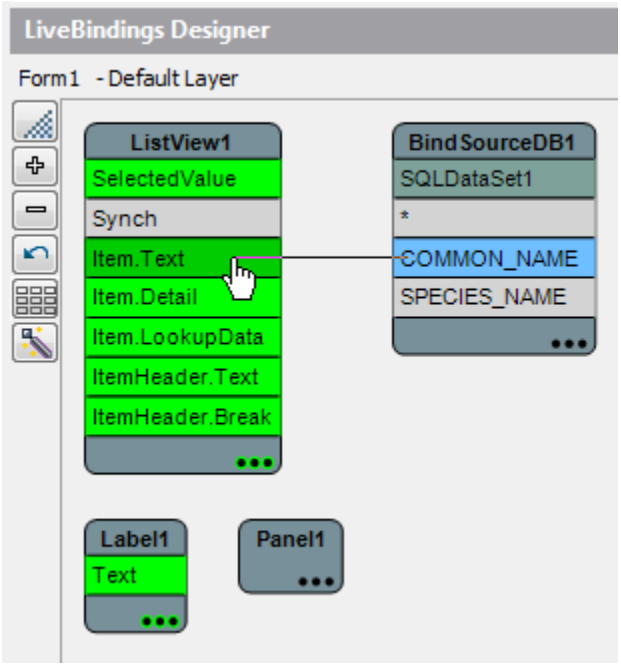

    2. Set the LoginPrompt property to **False**, so that the user is not prompted for a login.
    3. Click the ellipsis [...] for the Params property, and set the **Database** value to `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\dbdemos.gdb` (location of the database).
    4. Ensure the **VendorLib** value is set to `ibtogo.dll` (client software library); then click ok to close the dialog box:




5. Set the **Connected** property to **True**.

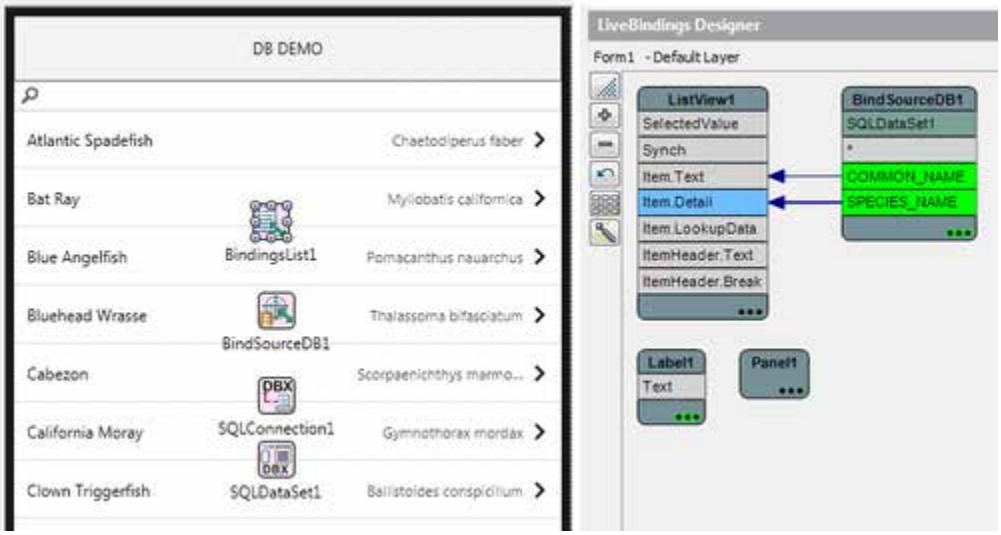
**Note:** If you get an error ("unavailable database") on development environment, this means you do not have a current license for InterBase. The license of InterBase Developer Edition is included as part of the product for some product editions. For more information, see Troubleshooting.

2. Add a **TSQLDataSet** component to the form, and set the following properties:
    1. Set the SQLConnection property to **SQLConnection1** (the one that you added in a previous step).
    2. Set the CommandText property to `select COMMON_NAME, SPECIES_NAME from BIOLIFE order by COMMON_NAME`.
    3. Set the Active property to **True**.
3. Open the LiveBindings Designer and connect the data and the user interface as follows:
    1. Click **COMMON_NAME** in SQLDataSet1, and drag the mouse cursor to **Item.Text** in ListView1.




At this point, TBindSourceDB and TBindingsList components were added to the form.

2. Click **SPECIES_NAME** in BindSourceDB1, and drag the mouse cursor to **Item.Detail** in ListView1.




# Deploying Your Application to Mobile

Up to this point, you have used InterBase on your desktop. This means that the actual database is located at your local hard disk drive (for example, C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\dbdemos.gdb). On the mobile Device, the application is sand-boxed, and typically you can only read and write data that is located in the **Documents** folder (for iOS device) and **internal** storage (for Android device) under your application folder.

To connect to a local database on mobile, you need to perform the following actions:

- Deploy the database to the mobile Device.
- Check the configuration (to connect to the database file) to a local file under the **Documents** folder (for iOS device) or **internal** storage (for Android device).

## Deploy InterBase ToGo, dbExpress Driver, and the Database File to Mobile

To execute your application on mobile, you need to deploy the following files:

- Interbase ToGo
- dbExpress Driver to InterBase (for iOS Simulator)
- The database file (dbdemos.gdb)

1. You can add the database to your project with one of the following two methods:
   - Right-click the project name in the **Project Manager** and select **Add...** from the context menu (or **Project > Add to Project**) to display the Add to Project dialog box. Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data`, select the database `dbdemos.gdb` and click **Open**.
   - Navigate to the database location `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data` and drag and drop the database `dbdemos.gdb` to the project in the Project Manager. Click **Yes** to confirm that you want to add the file to your project.
2. After adding the database file, the **Featured Files** window displays. Select the following database modules, and then click **OK** to close the Featured Files dialog box:
   - **InterBase ToGo**. You need to select the license to be used when deploying the application on the device.
     - The **Tip** at the beginning of this tutorial describes how to activate an InterBase license.
     - The suggested names for the license files available are listed in the Featured Files dialog, under the following name pattern: **reg_*.txt**.

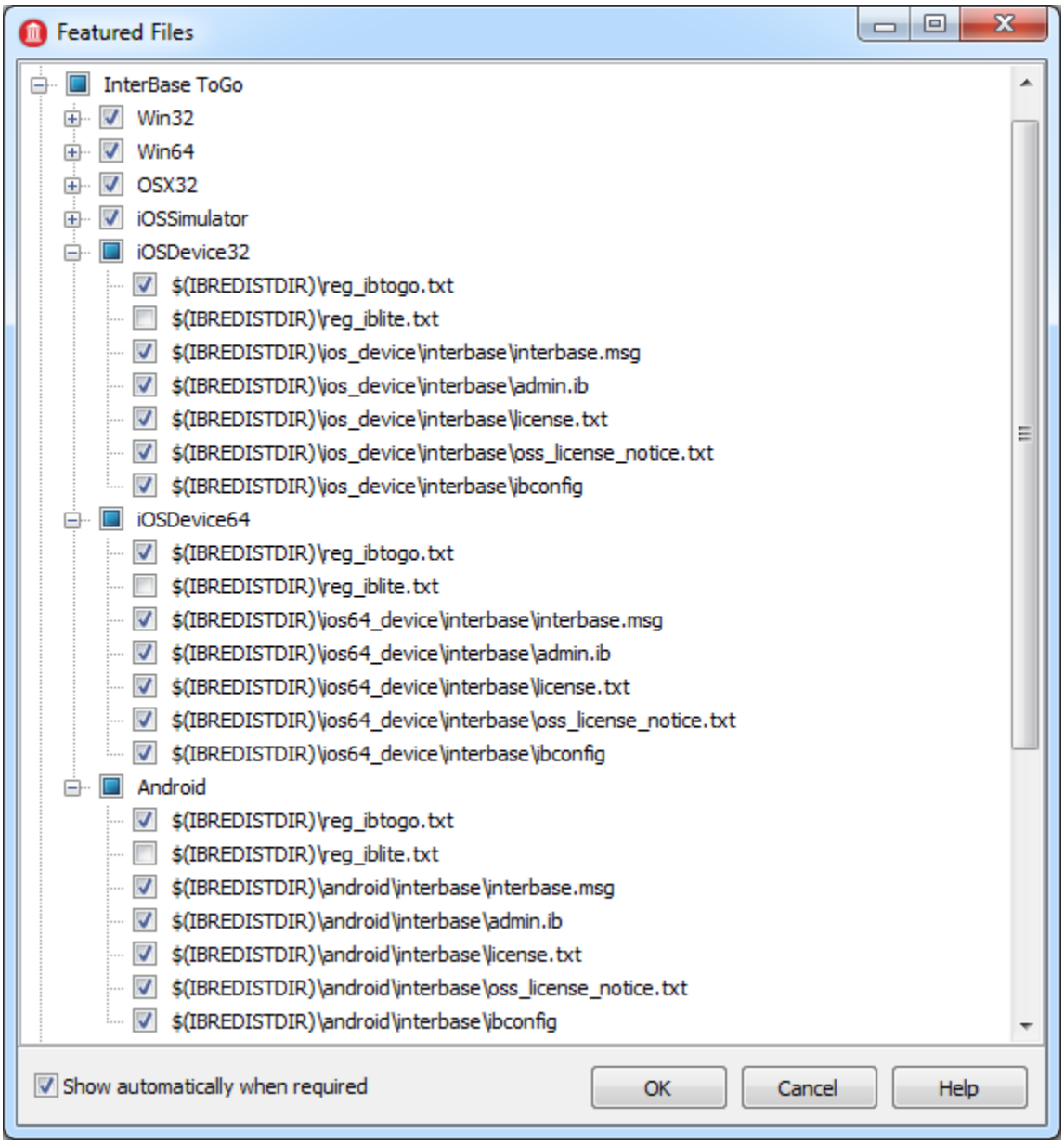
As you can see in the image below, the **reg_ibtogo.txt** license file is selected for this tutorial.

- You might have received from Embarcadero a license file for IBToGo or IBLite that has a pattern of **reg_nnnnnnn.txt**, where **nnnnnnn** is a generated number:
  - If you have saved that file over **reg_ibtogo.txt** or **reg_iblite.txt** in the location below (for example, `C:\Users\Public\Documents\Embarcadero\InterBase\redist\InterBaseXE7`), you can just select the desired license.

- If you have saved the file with its original name, then select **Add Files** (shown in the next step) and include the license file in the list of files that need to be deployed with the application.

- **DBExpress InterBase Driver**

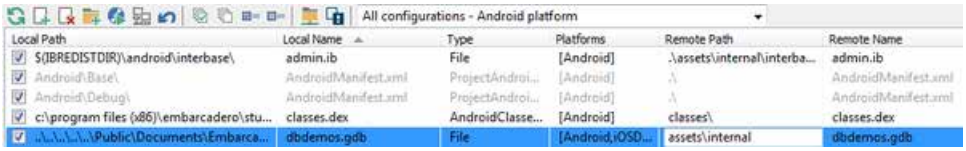
**Tip:** If you plan to test your application on the iOS Simulator, then you also have to select the **DBExpress InterBase Driver** for **iOS Simulator**.




3. Open the Deployment Manager by selecting **Project > Deployment**.
4. Select **Debug configuration - iOS Device platform** or **Debug configuration - Android platform** from the drop-down list of target platforms at the top of the Deployment Manager and see that the database `dbdemos.gdb` has been added to the platforms.

5. See how the **Remote Path** of `dbdemos.gdb` has been set for iOS and Android platforms:
    - **Remote Path** on iOS Device platform: `StartUp\Documents\`


    - **Remote Path** on Android platform: `assets\internal\`




As you just configured, when you run the app on the mobile device, the database file (dbdemos.gdb) is to be deployed to the **Documents** folder (for iOS platform) or **internal** storage (for Android platform) in the sandbox area of your multi-device application.

## Modify Your Code to Connect to a Local Database File on Mobile

As described in the previous step, the TSQLConnection component is connected to a database on your local file system with an absolute path. So you need to replace the location of the file before connecting to the database, as follows:

1. In the Form Designer, select the SQLConnection1 component.
2. In the Object Inspector, double-click the Value field of the BeforeConnect event.
3. Add the following code to this event handler:

**Delphi:**

```
procedure TForm1.SQLConnection1BeforeConnect(Sender: TObject);
begin
  {$IF DEFINED(iOS) or DEFINED(ANDROID)}
   SQLConnection1.Params.Values['Database'] :=
      TPath.Combine(TPath.GetDocumentsPath, 'dbdemos.gdb');
  {$ENDIF}
end;
```

**C++:**

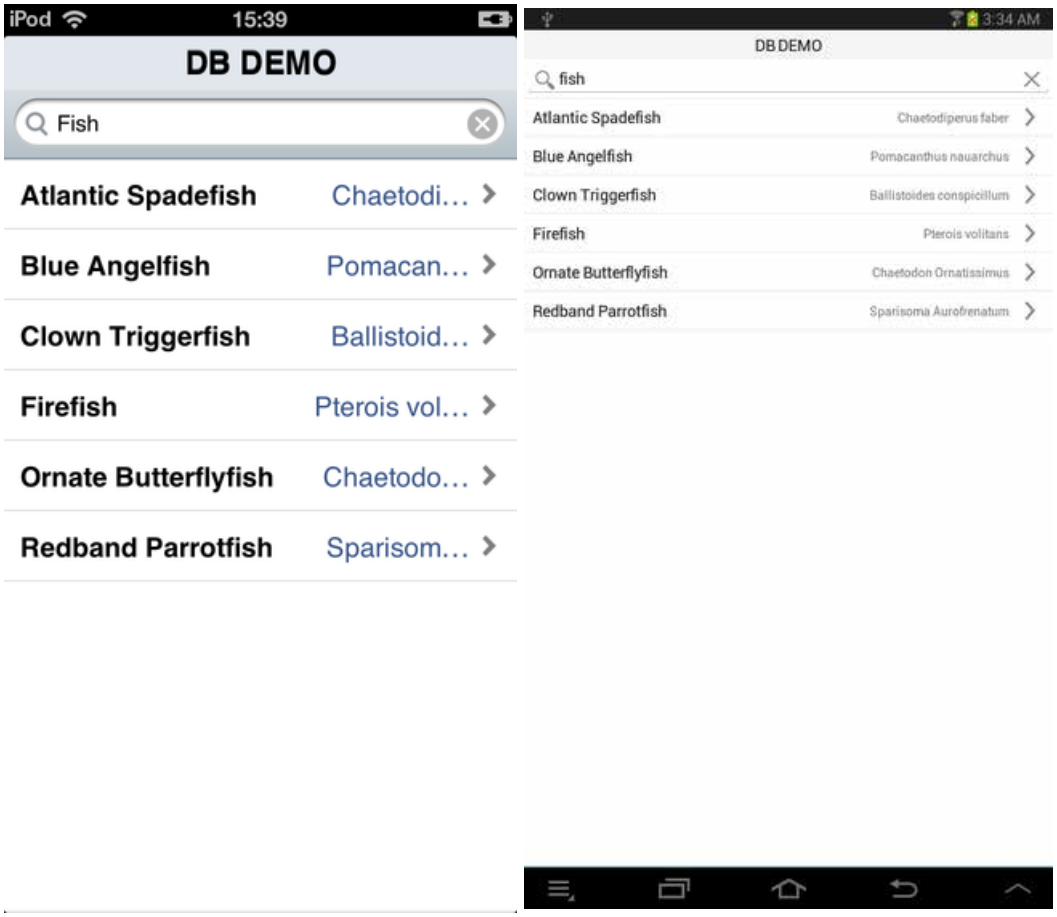
```
void __fastcall TForm1::SQLConnection1BeforeConnect(TObject *Sender)
{
  #if defined(_PLAT_IOS) || defined(_PLAT_ANDROID)
  SQLConnection1->Params->Values["Database"] =
System::Ioutils::TPath::Combine(System::Ioutils::TPath::GetDocumentsPath(),
"dbdemos.gdb");
  #endif
}
```

The TPath record is declared in **System.IOUtils** unit, so you need to add **System.IOUtils** in the uses clause.

# Run Your Application on a Simulator or on a Mobile Device

Now your application is ready to run. You should be able to browse data just as you can in the IDE. You can narrow down the list using the Search Box.

**iOS** **Android**




# Troubleshooting

## InterBase Issues

See the following section with detailed information about Interbase License Issues.

**Note**: Follow the steps at IBLite and IBToGo Test Deployment Licensing to obtain a valid license file.

### Exception Handling Issues

If your application raises an exception without having proper exception handling code, your multi-device application simply crashes (disappears) at run time.

If you encounter a crash, you might want to connect manually to the database while you troubleshoot the issue using the following steps:

1. Select the **SQLConnection1** component, and change the Connected property to **False**.
2. Drop a button on the form, and create the following event handler to manually connect to the database:

   **Delphi:**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    SQLConnection1.Connected := True;
    SQLDataSet1.Active := True;
  except
    on e: Exception do
    begin
      ShowMessage(e.Message);
    end;
  end;
end;
```

   **C++:**

```
void __fastcall TForm1::Button1Click(TObject *Sender) {
        try {
        SQLConnection1->Connected = True;
        SQLDataSet1->Active = True;
        }
        catch(Exception *e) {
        ShowMessage(e->Message);
}
```

3. Check the error message.

## See Also

- InterBase ToGo with dbExpress
- IBLite and IBToGo Licensing in RAD Studio
- http://www.embarcadero.com/products/interbase/product-editions
- Getting Started with InterBase Express
- Mobile Tutorial: Using dbExpress and SQLite (iOS and Android)

- Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)
- Mobile Tutorial: Using FireDAC in Mobile Applications (iOS and Android)

# Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)

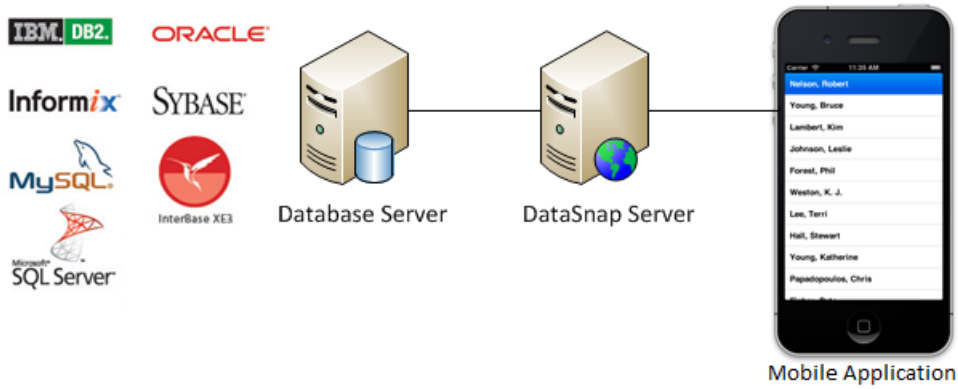
Before starting this tutorial:

- Read and perform the following tutorial session:
  - Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)
  - Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)
- Choose **Start | Programs | Embarcadero InterBase XE7 | <xx>-bit instance = gd_db | InterBase Server Manager** and verify that the InterBase server is running. The InterBase server must be running before you create or run the example.

This tutorial describes how to connect to an Enterprise database from a mobile client application.

To connect to an Enterprise Database, you need to have a **client library**. In most cases, the client library is provided by the database vendor in DLL format. This strategy does not work well for mobile devices because no client library is available. To resolve this issue, you can develop a **middle tier** to connect to an Enterprise Database, and your application can communicate with the middle tier.

RAD Studio provides the **DataSnap** framework with which you can develop the middle tier (and access the middle tier) with almost no coding required. This tutorial describes the steps to develop the middle tier and then develop the mobile client.



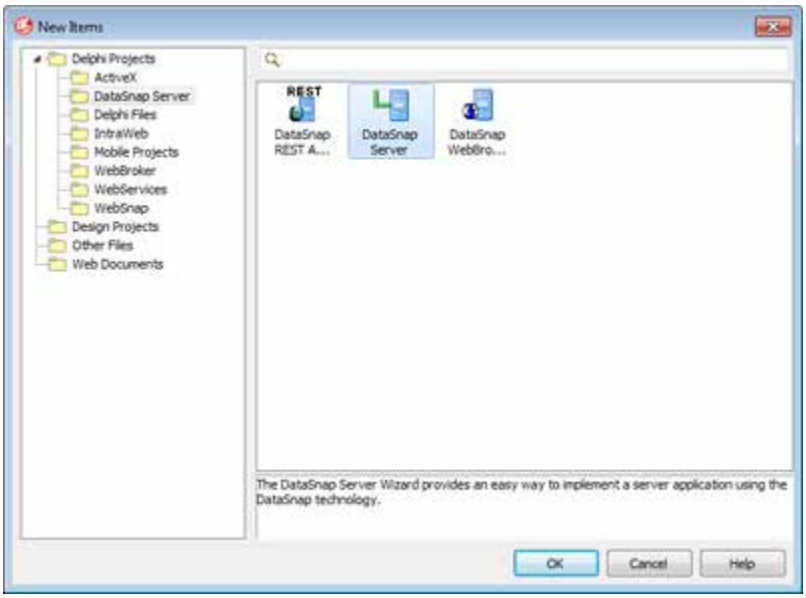

## Creating the Middle Tier, a DataSnap Server

First, create a DataSnap server that exposes a table from a database server. This tutorial uses a **DataSnap Server VCL Forms Application** as a DataSnap server.

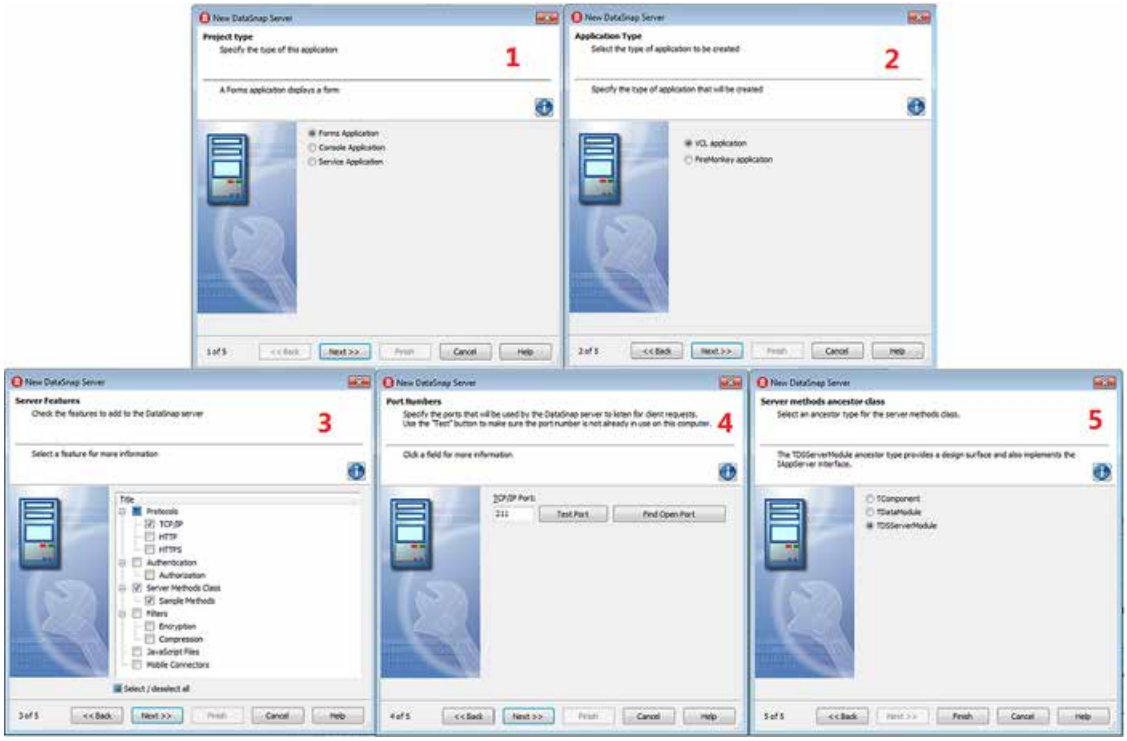
**Note:** In this tutorial, the DataSnap server (a VCL application) functions as the middle tier in a multi-tiered database application. You can easily create and later delete an instance of a DataSnap server. After you understand the basic steps, you can convert the middle tier to a Windows service application.

## Create a DataSnap Server VCL Application

1. In order to create a new Delphi or C++ project, choose **File > New > Other** and from the **New Items** dialog select:
   - **For Delphi:Delphi Projects > DataSnap Server > DataSnap Server**
   - **For C++:C++Builder Projects > DataSnap Server > DataSnap Server**

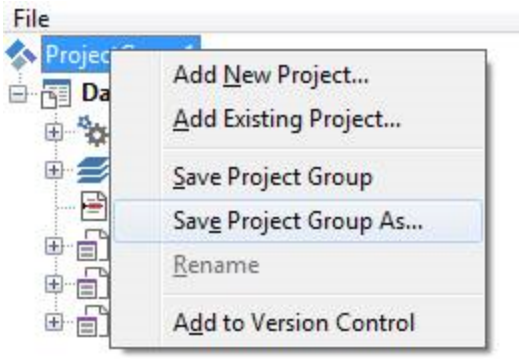



2. The New DataSnap Server wizard appears:




1. At first step, choose **Forms Application** as project type.
2. At the second step, check **VCL Application** as application type.
3. At the third step, choose the **TCP/IP** protocol, **Server Methods Class** and **Sample Methods** from the Server Features list.
4. At the fourth step, leave the default TCP/IP communications port to **211**. This will ensure that the communication between the client and the server will pass through the default DataSnap port.
5. At the final step (number five) select **TDSServerModule** as the ancestor for the Server Methods.

2. Save the form unit as **DataSnapServerUnit**.
3. Switch to **DataSnapServerUnit**, and change the **Name** property of the Form to **DSServerForm**.
4. Save the server methods unit (by default as created by the Wizard: **ServerMethodsUnit1**) as **ServerModuleUnit**.
5. Save the server container unit (by default as created by the Wizard: **ServerContainerUnit1**) as **ServerContainerUnit**.
6. Save the new project as **DataSnapServerProject**.
7. Select ProjectGroup1 in the Project Manager, and save the project as **DataSnapTutorialProjectGroup.groupproj**.




## Define a DataSet on the DataSnap Server

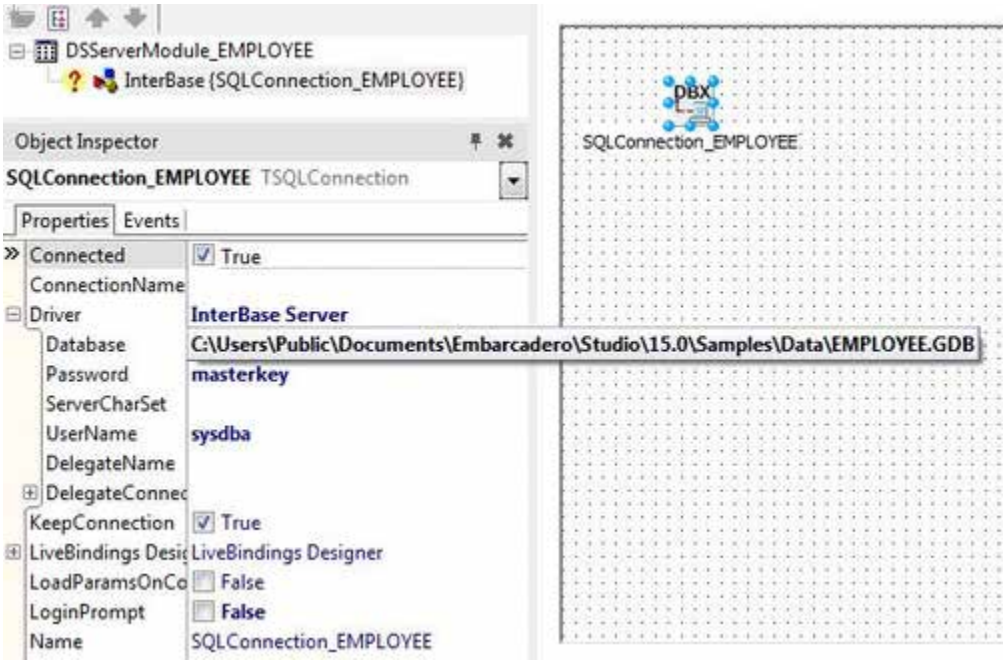
1. Switch to the **ServerContainerUnit.pas** file and replace the uses clause in the implementation with: `uses Winapi.Windows, ServerModuleUnit`, for Delphi, and replace **#include "ServerMethodsUnit.h"** with `#include "ServerModuleUnit.h"` in **ServerContainerUnit.cpp**, for C++;
2. Switch to the **ServerModuleUnit.pas** file.
3. In the Form Designer, change the **Name** property of the Server Module to **DSServerModule_EMPLOYEE**.
4. Configure the following components on the Server Module:
   - Drop a TSQLConnection component on the Server Module, and set the following properties:

**TSQLConnection** encapsulates a dbExpress connection to a database server.

- Set the Name property to **SQLConnection_EMPLOYEE**.
- Set the LoginPrompt property to **False**.
- Set **Driver** to **InterBase Server**.

**Note:** Make sure that the **InterBase Server** is running.

- Expand the **Driver** node, and set the **DataBase** property to `C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\EMPLOYEE.GDB`.
- Change the Connected property to **True**. If you get an error, double-check the **Driver** properties:




- Drop a TSQLDataSet component on the Server Module, and set the following properties:

**TSQLDataSet** represents the data retrieved using dbExpress.

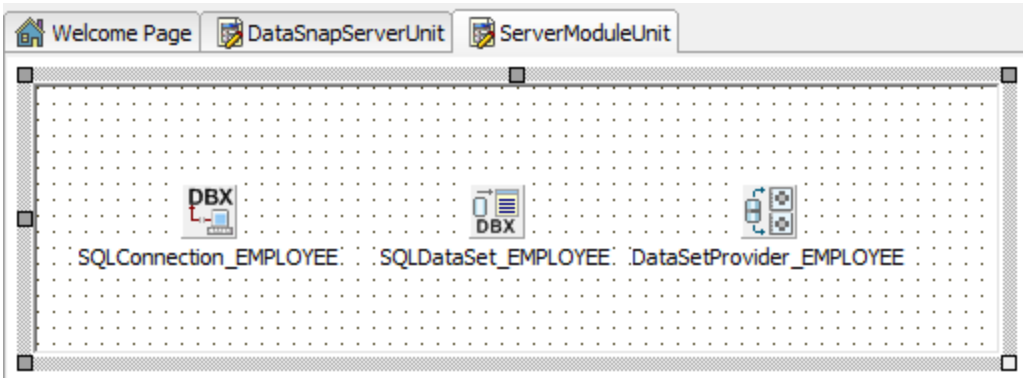
- Set the **Name** property to **SQLDataSet_EMPLOYEE**.
- Set the **SQLConnection** property to **SQLConnection_EMPLOYEE**.
- Set the **CommandType** property to **ctTable**.
- Set the **CommandText** property to **EMPLOYEE**.
- Change the **Active** property to **True**. If you get an error, double-check the properties you just configured.

- Drop a TDataSetProvider component on the Server Module, and set the following properties:

**TDataSetProvider** packages data from a dataset and passes one or more transportable data packets to the DataSnap client.

- Set the **Name** property to **DataSetProvider_EMPLOYEE**.
- Set the **DataSet** property to **SQLDataSet_EMPLOYEE**:

§




**Note:** This tutorial uses InterBase as an example. However, you can connect to any database server using the same steps. Select the proper driver, and other properties to point to your database.

## Expose the DataSet from the DataSnap Server

You have just created a new Server Module that contains a DataSet and a DataSetProvider that packages data to the next layer. The next step is to expose the Server Module to the DataSnap client.

1. In the Form Designer, open **ServerContainerUnit**.
2. Select **DSServerClass1**, and update the existing event handler for the **OnGetClass** event. Add the following code to the **DSServerClass1** event handler:

**Delphi:**

```
procedure TServerContainer1.DSServerClass1GetClass(DSServerClass: TDSServerClass;
  var PersistentClass: TPersistentClass);
begin
  PersistentClass := TDSServerModule_EMPLOYEE;
end;
```

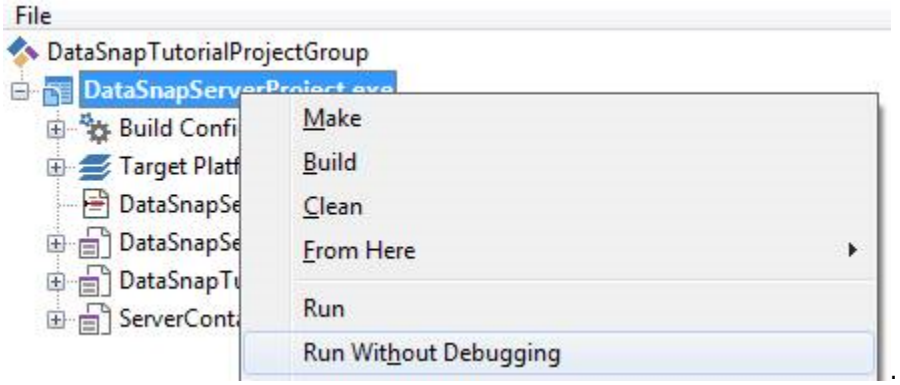
**C++ (only for iOS):**

```
void __fastcall TServerContainer1::DSServerClass1GetClass(TDSServerClass
*DSServerClass,
          TPersistentClass &PersistentClass)
{
        PersistentClass =  __classid(TDSServerModule_EMPLOYEE);
}
```

With this event handler, the DataSnap Server exposes providers as well as public methods in this class to a DataSnap client. Based on the steps in the previous section, now you are going to expose the **DataSetProvider_EMPLOYEE DataSetProvider** component to your DataSnap client.

### Run the DataSnap Server

Implementation of the DataSnap Server is complete. Right-click **DataSnapServerProject.exe** and select **Run Without Debugging**.

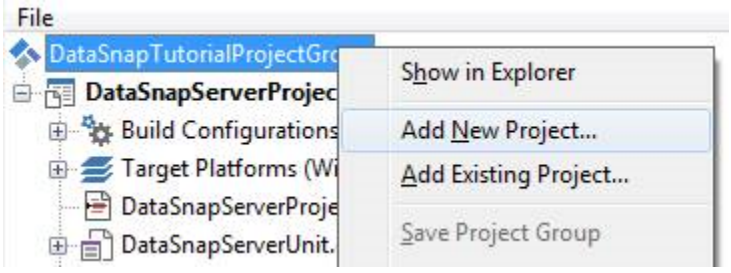



Now you can see the DataSnap server running on your Windows machine. Because this DataSnap server has no UI element, it looks like a blank form, and this is as expected at this point.

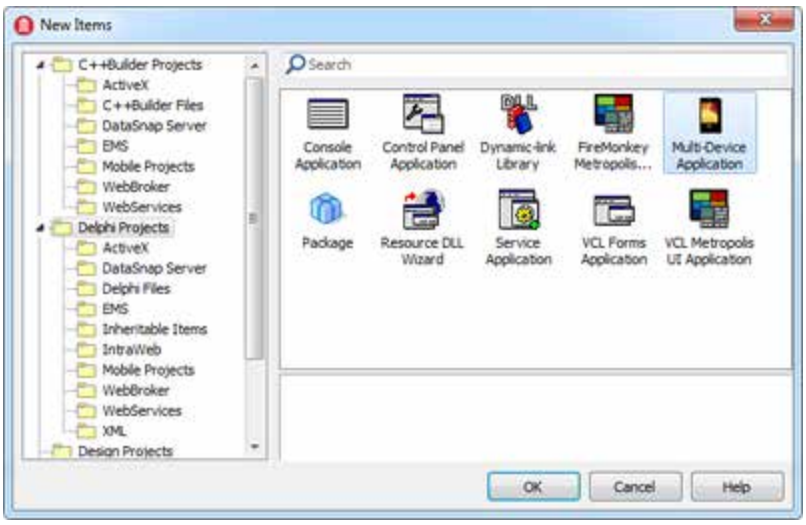

## Creating a Mobile Application that Connects to the DataSnap Server

The next step is creating the mobile client application.

1. In the Project Manager, right-click **DataSnapTutorialProjectGroup**, and select **Add New Project**.




2. Select **Multi-Device Application** on the **Delphi Projects** page:




3. Save the new Unit as **DataSnapClientUnit**.
4. Save the new Project as **DataSnapClientProject**.
5. Open **DataSnapClientUnit**, and change the **Name** property of the Form to **DSClientForm**.
6. Drop the following components on the **Form Designer**:
    - § TSQLConnection component (SQLConnection1)

**TSQLConnection** encapsulates a dbExpress connection to a database server. Also, it supports the DataSnap server.

- § Set the **Driver** property to **DataSnap**.
- § Expand the **Driver** property, and set the **HostName** property to the host name or IP address of the DataSnap server.
- § Set the LoginPrompt property to **False**.
- § Set the Connected property to **True**.

If you see an error, please double-check the properties you have just set.

- § TDSProviderConnection component (DSProviderConnection1)

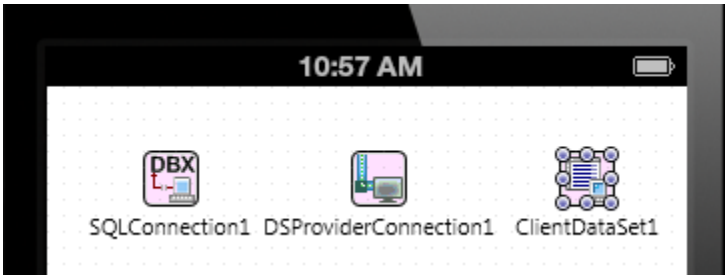
The TDSProviderConnection component provides connectivity to the DataSnap server using dbExpress.

- § Set the SQLConnection property to **SQLConnection1**.
- § Set **ServerClassName** to **TDSServerModule_EMPLOYEE**. This name needs to match the name of the class of the Server Module of the DataSnap server.
- § Set the Connected property to **True**.

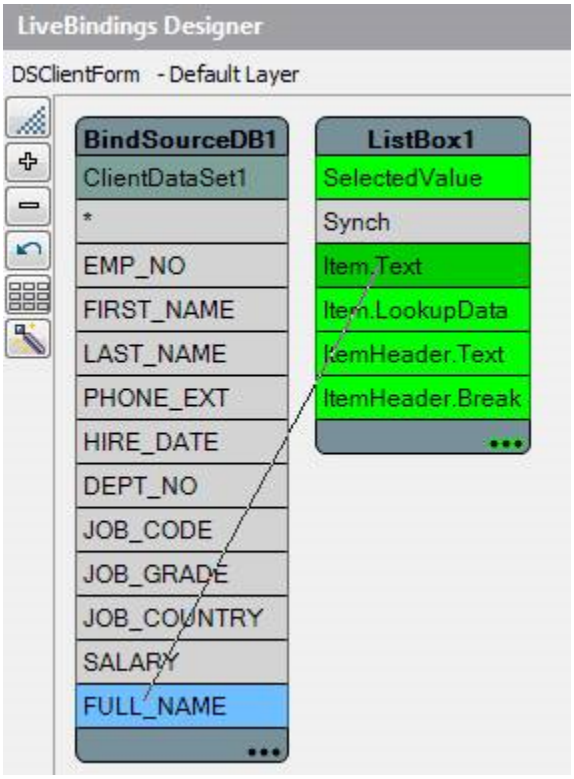
- TClientDataSet component (ClientDataSet1)

TClientDataSet implements a database-independent dataset, and this can be used as a local in-memory buffer of the records from another dataset.

  - Set the RemoteServer property to **DSProviderConnection1**.
  - Set the ProviderName property to **DataSetProvider_EMPLOYEE**. This name needs to match the name of the provider for the DataSnap server.
  - Set the Active property to **True**.
- TListBox component
  - Set the Align property to **Client**:




7. Open the LiveBindings Designer and connect the data and user interface as follows:
   1. Click **FULL_NAME** in BindSourceDB1, and drag the mouse cursor to **Item.Text** in ListBox1:



2. Now you have created and configured the DataSnap Client on the mobile platform. You should be able to see the data coming from the DataSnap server in the IDE:

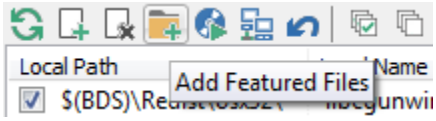
**iOS** **Android**


# Deploy the MIDAS Library to iOS Simulator

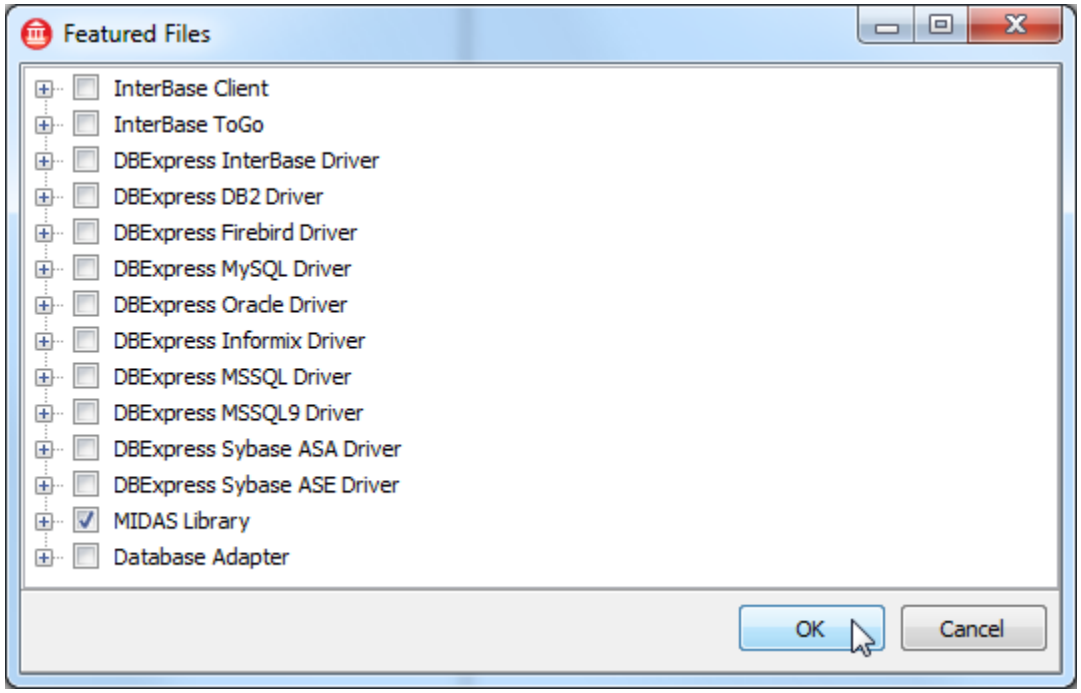
To execute your application on the iOS Simulator, you need to deploy the following files:

- MIDAS Library

1. Open the Deployment Manager by selecting **Project > Deployment**.
2. Select Add Featured Files ( ):



3. Select the following module, and then click **OK** to close the Deployment Manager:
    - MIDAS Library




# Run Your Application on the mobile platform

Now your application is ready to run.

In the Project Manager, select the mobile target platform, and run your application. You should be able to browse data just as you do within the IDE.

# See Also

- Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)
- Mobile Tutorial: Using dbExpress and SQLite (iOS and Android)
- Mobile Tutorial: Using FireDAC in Mobile Applications (iOS and Android)
- Developing DataSnap Applications
- Understanding Multi-tiered Database Applications
- Datasnap.DSServer.TDSServer
- Start the InterBase Server

# Mobile Tutorials: Table of Components Used

The following table lists the FireMonkey components that are used in each of the mobile tutorials:

| Mobile Tutorial | FireMonkey Components, Properties and Methods Used |
| --- | --- |
| Creating a Multi-Device Application (iOS and Android) | TEdit, TButton, TLabel<br>KillFocusByReturn, Text, Name |
| Using a Button Component with Different Styles (iOS and Android) | TButton, TSpeedButton<br>Position, Height, Width, StyleLookup, TintColor, IconTintColor, GroupName, IsPressed |
| Using a Calendar Component to Pick a Date (iOS and Android) | TDateEdit<br>ShowCheckBox |
| Using Combo Box Components to Pick Items from a List (iOS and Android) | TComboBox<br>ItemIndex, IndexOf, OnChange |
| Using a MultiView Component to Display Alternate Views of Information (iOS and Android) | TMultiView, TDrawerAppearance<br>Visible, Mode, MasterButton, DoOpen, DoClose, GetDisplayName, DoInstall, TargetControl, DrawerOptions, ShadowOptions |
| Using the Web Browser Component (iOS and Android) | TWebBrowser, TButton, TVirtualKeyboard<br>ReturnKeyType, GoBack, KeyboardType, KillFocusByReturn |
| Using Tab Components to Display Pages (iOS and Android) | TTabControl, TTabItem, MultiResBitmap Editor, ActionList, TButton |
| Using LiveBindings to Populate a ListView (iOS and Android) | TListView, TPrototypeBindSource, TextButton |
| Using ListBox Components to Display a Table View (iOS and Android) | TListBox, TLabel, TToolBar, TSpeedButton, TSearchBox, TListBoxHeader, TListBoxGroupHeader, TListBoxGroupFooter<br>Align, StyleLookup, HorzAlign, Text, Accessory, TListBoxItemData.Detail, Bitmap, StyleLookup, TListBoxItemData.Detail |
| Using LiveBindings to Populate a ListBox in Mobile Applications (iOS and Android) | TListBox, DefaultItemStyles.ItemStyle, TRectangle, TPrototypeBindSource, LiveBindings Designer, |
| Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android) | TLayout and its properties: Align, Margins, Padding, and Anchors<br>TVertScrollBox |

| | |
|---|---|
| Taking and Sharing a Picture, and Sharing Text (iOS and Android) | TToolBar, TImage, TActionList, FMX.MediaLibrary, TTakePhotoFromCameraAction, OnDidFinishTaking, OnBeforeExecute,TShowShareSheetAction, TToolBar, TButton, TMemo |
| Using Location Sensors (iOS and Android) | TListBox with added items and groupheaders, TWebBrowser, TLocationSensor, OnLocationChanged, TSwitch, TGeocoder and its methods |
| Using Notifications (iOS and Android) | TNotificationCenter, TNotification, TNotification.Number, ScheduleNotification, RepeatInterval |
| Using InterBase ToGo with dbExpress (iOS and Android) | TListView, TPanel, TLabel, TSQLDataSet, LiveBindings Designer, Deployment Manager |
| Using SQLite and dbExpress (iOS and Android) | Data Explorer, TListBox, TButton, TLabel, TSQLQuery, InputQuery, Deployment Manager |
| Connecting to an Enterprise Database from a Mobile Client (iOS and Android) | DataSnap Server Wizard, TDSServerModule, TSQLConnection, TSQLDataSet, TDataSetProvider, TDSProviderConnection, TClientDataSet, TListBox |
| Using InterBase ToGo with FireDac (iOS and Android) | TFDConnection, TBindSourceDB, TFDQuery, TListView, TPanel, TLabel, LiveBindings Designer, TFDGUIxWaitCursor, FTFDPhysSQLiteDriverLink, Deployment Manager |
| Using FireDAC and SQLite (iOS and Android) | TFDConnection, LiveBindings Wizard, TBindSourceDB, TFDQuery, TListView, LiveBindings Designer, TFDGUIxWaitCursor, FTFDPhysSQLiteDriverLink, Deployment Manager |
| Using Remote Notifications (iOS and Android) | TPushEvents, AutoActivate, TKinveyProvider, AppKey, AppSecret, MasterSecret, TParseProvider, ApplicationID, RestApiKey,Provider, LiveBindings Designer |
| Using BaaS for Backend Storage (iOS and Android) | TListView, TLabel, TButton, TBackendStorage, TKinveyProvider and its properties AppKey, AppSecret, MasterSecret, and the TParseProvider component and its properties ApplicationID, MasterKey and RestApiKey |

**Note:** This list of components is intended for informational purposes only, and might not be entirely accurate or complete.