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 San Francisco, with offices located around the world. Embarcadero is online at www.embarcadero.com.

November, 2014
# CONTENTS

Mobile Tutorials: Mobile Application Development (iOS and Android) ........................................... 9  
Setup .................................................................................................................................................. 10  
Using Basic User Interface Elements ................................................................................................. 10  
Using Device Functionality .................................................................................................................. 12  
Accessing a Database .......................................................................................................................... 13  
Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS) .................................. 15  
  Requirements on the Mac and iOS ................................................................................................... 15  
  Step 1: Configure Your Mac to Run Your iOS Application ............................................................... 16  
  Step 1: Install the Platform Assistant .............................................................................................. 17  
  Step 2: Run the Platform Assistant .................................................................................................. 17  
  Step 3: Install Xcode on the Mac ....................................................................................................... 19  
  Next Steps ......................................................................................................................................... 19  
Additional Steps to Configure Your Mac to Run Your iOS Application on Your iOS Device ............ 20  
  Step 1: Make Sure that the Xcode Command Line Tools Are Installed on Your Mac ...................... 20  
  Step 2: Sign Up for a Developer Account ......................................................................................... 22  
  Step 3: Request, Download and Install Your Development Certificate ........................................... 22  
  Request, Download and Install Your Certificate ................................................................................ 23  
  Step 4: Register Your Device for Deployment .................................................................................. 24  
  Step 5: Create and Install a Provisioning Profile .............................................................................. 25  
Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS) .......................... 28  
Setting Up Your RAD Studio Environment ....................................................................................... 29  
  Create a Connection Profile for the Mac ......................................................................................... 29  
  Add an SDK to the Development System for the iOS Device Connected to the Mac ..................... 32  
Mobile Tutorial: Set Up Your Development Environment on Windows PC (Android) ................... 35  
Setting Up Your RAD Studio Environment ....................................................................................... 35  
  Android Development Tools Are Installed with RAD Studio ....................................................... 36  
  Adding Your Android SDK in Tools Options > SDK Manager ...................................................... 37  
  Installing the USB Driver for Your Android Device (Required) .................................................... 43  
  Creating an Android Emulator (Optional) ....................................................................................... 45  
Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android) ......................... 46  
  Before You Start .............................................................................................................................. 46  
  Step 1: Create a New FireMonkey Application for Android or iOS ................................................. 47  
  Step 2: Select a Style ....................................................................................................................... 48  
  Step 3: Place Components on the Multi-Device Form .................................................................. 49  
  Step 4: Adding Views to Your Project ............................................................................................. 52  
  Step 5: Write an Event Handler for a Button Click by the User ...................................................... 52
Step 6: Test Your Mobile Application................................................................. 54
  Test Your Android Application........................................................................ 55
  Test Your iOS Application................................................................................ 57
Mobile Tutorial: Using a Button Component with Different Styles (iOS and
Android)............................................................................................................ 60
  Buttons in Mobile Platforms........................................................................... 60
  Define the Look and Feel for a Button Component........................................ 61
    UsingTintColor on Buttons........................................................................... 61
    Placing an Image over a Button.................................................................... 64
  Create a Segmented Control Using Button Components............................... 65
  Create a Scope Bar on a Toolbar Component.................................................. 67
Mobile Tutorial: Using a Calendar Component to Pick a Date (iOS and Android) ............................ 69
  Calendar in Mobile Platforms......................................................................... 69
  Implementing an Event Handler for User Changes to the Date....................... 71
Mobile Tutorial: Using ComboBox Components to Pick Items from a List (iOS and
Android)............................................................................................................ 74
  Implementing a Picker in Multi-Device Applications........................................ 74
  Building a List of Items Using Code............................................................... 77
  Displaying a Specific Item.............................................................................. 78
  Implementing an Event Handler for the User's Selection................................. 79
Mobile Tutorial: Using a MultiView Component to Display Alternate Views of
Information (iOS and Android)........................................................................... 83
  About the TMultiviewComponent................................................................. 83
    Master Pane Presentation Modes............................................................... 85
  Designing the User Interface........................................................................... 89
    Designing the Master Pane.......................................................................... 89
    Designing the Detail Pane........................................................................... 90
  Setting the TMultiviewComponent Properties............................................... 91
  Final Steps...................................................................................................... 91
  Running the Example Application................................................................... 95
Mobile Product Samples that Use TMultiview..................................................... 96
Mobile Tutorial: Using the Web Browser Component (iOS and Android) ............ 97
  Design the User Interface.............................................................................. 99
  Write an Event Handler to Open a Web Page when the User Changes the URL
  in the Edit Control......................................................................................... 102
    Implement a Common Method to Open a Web Page................................... 103
    Implement an Event Handler for the OnChange Event............................... 104
    Implement an Event Handler for the Back Button...................................... 105
  Selecting the Proper Virtual Keyboard for the Web Browser Application...... 106
  WebBrowser Mobile Code Snippet.................................................................. 107
Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android) .... 109
  Using the Native Style for Tabs on iOS and Android..................................... 110
  Designing Tab Pages Using the Form Designer.............................................. 111
Comparing the Tab Settings on iOS and Android .................................................. 116
Using Custom Multi-Resolution Icons for Your Tabs ............................................. 121
Displaying Multi-Resolution Custom Icons on Tabs .............................................. 122
Using a Single-Resolution Bitmap for a Custom Icon ........................................... 126
Defining Controls within a TabControl ................................................................... 127
Changing the Page at Run Time ............................................................................ 130
By the User Tapping the Tab ................................................................................ 130
By Actions and an ActionList .............................................................................. 130
By Source Code .................................................................................................. 135
Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android) ... 137
  Step 1: Creating the Project ............................................................................... 137
  Step 2: Adding Fields ........................................................................................ 139
  Step 3: Creating LiveBindings .......................................................................... 140
  Step 4: Adding More Fields (Bitmaps, Currency) .............................................. 143
  Step 5: Adding the onButtonClick Event Handler .......................................... 145
The Results ........................................................................................................... 146
Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android) ............................................................................................................. 149
  Using ListBox Components to Display a Table View in Mobile Platforms ....... 149
    Plain List .......................................................................................................... 150
    Grouped List .................................................................................................... 151
    Search Box ...................................................................................................... 152
  Create Items on the ListBox Component .......................................................... 153
    Add a Header .................................................................................................... 155
    Add a Group Header/Footer to the List ............................................................ 156
    Show List Items as Separate Grouped Items ................................................. 158
    Add a Check Box or Other Accessory to a ListBox Item ................................ 159
    Add an Icon to a ListBox Item ........................................................................ 159
    Add Detail Information to an Item .................................................................. 160
    Running Your Application .............................................................................. 160
  Create Your ListBox Application ....................................................................... 161
    Add Items to a ListBox from Your Code ......................................................... 161
    Create an Overflow Menu .............................................................................. 165
    Creating the Event Handler for the Overflow Button ..................................... 167
    Add a Search Box ............................................................................................ 168
    Running Your Application .............................................................................. 169
Mobile Tutorial: Using LiveBindings to Populate a ListBox in Mobile Applications
  (iOS and Android) .............................................................................................. 171
  Step 1: Creating the Project ............................................................................. 171
  Step 2: Creating the LiveBindings .................................................................... 174
  The Results ....................................................................................................... 176
Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS
  and Android) ....................................................................................................... 178
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.
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 a Button Component with Different Styles (iOS and Android)

- Using a Calendar Component to Pick a Date (iOS and Android)
Using Basic User Interface Elements

- Using Combo Box Components to Pick Items from a List (iOS and Android)
- Using the Multi View 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 Device Functionality

- 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)

- Using Remote Notifications (iOS and Android)
Accessing a Database

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

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
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

- 10.8 Mountain Lion
- 10.9 Mavericks
- 10.10 Yosemite

(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 Mac OS X development, the latest version of **Xcode**
- For iOS development, the iOS SDK and the appropriate version of **Xcode** installed (see the table below), along with **the Xcode command line tools**:

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

**iOS version** | **Related Xcode version**
--- | ---
iOS 7.0 | Xcode 5.0
iOS 7.1 | Xcode 5.1
iOS 8.0 | Xcode 6.0

- Membership in one of several Apple Developer Programs, which are described in this topic under [Sign up for a Developer account](#).
- 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.

![Deployment and Testing Diagram](image)

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 Mac 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 Mac 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 Mac OS X installer for the Platform Assistant is named PAServer15.0.pkg and it is available in two places:

- Inside the RAD Studio installation directory on your PC:
  C:\Program Files (x86)\Embarcadero\Studio\15.0\PAServer\PAServer15.0.pkg

- On the Web, for download to the Mac:
  http://altd.embarcadero.com/release/studio/15.0/PAServer/PAServer15.0.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 15.0.app) as follows:

1. Navigate to the top-level Applications folder.
2. Double-click PAServer 15.0.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:
Step 3: Install Xcode on the Mac

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

You can install Xcode from any of the following sources:

- On your "Mac OS X Install" DVD, under **Optional Installs**, double-click **Xcode.mpkg**
  to install Xcode on your system.
- At the [Mac App Store](https://appstore.com), 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:

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

- **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**.

![](image)

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**

**Notice:** Beginning from the 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** does 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 one of the iOS developer programs is a requirement for building, running, debugging, and deploying applications for iOS.

You can join a **developer program** in either of the following ways:

- As an individual developer.
- As a member (or leader) of a team in an enterprise (business) program or university program.

For more details, see [Joining an iOS 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**:

   ![Keychain Access](image.png)

   - 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* or *Window > Organizer > Devices tab* in Xcode 5)*.
3. Click on your device.
4. Next to the **Identifier** label is a string of characters:

   ![](image)

   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:

- **Xcode5:**
  1. Open Xcode on the Mac and go to the Organizer (Window > Organizer).
  2. In the Library section, select Provisioning Profiles and click Refresh.
  3. Xcode asks you to sign in with your Apple ID. Enter your credentials and select Log in.

The provisioning profiles available to you are installed into your Xcode:

4. Select a valid iOS provisioning profile and drag-and-drop it into the Provisioning profiles of your test device.
**Xcode 6:**

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

- Troubleshooting: Cannot Deploy to the iOS Device
- Useful Apple Web Pages:
  - Apple Developer Program
  - iOS Developer Program
  - 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**:

![Connection Profile Manager](image)
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).

![Remote machine information](image)

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:

![SDK Manager](image)

2. Click Add.
3. On the **Add a New SDK** dialog box, select **iOS Device** 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:

Click **OK** 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
- Mac 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)

The largest part of the testing process of a FireMonkey application destined for the Android target platform uses the Android Device target platform and requires a test Android device connected by USB cable to your development system. You can optionally test your application on an Android Emulator running on the Windows development platform. You do not need to use the Platform Assistant in developing apps for the Android target platform. (PAServer is, however, required for developing Mac OS X and iOS apps.)

Note: If you install the Android development tools during your RAD Studio installation, Installing the USB Driver for Your Android Device is the only setup step described here that you need to perform on your PC.

After you complete either this tutorial or the one required step, your next steps are to:

- Enable USB debugging on your Android device
- Configure your system to detect your Android device

Setting Up Your RAD Studio Environment

The following Android development tools are installed on your development system during RAD Studio installation:

- Java Development Kit (JDK)
- Android Software Development Kit (SDK)
- Android Native Development Kit (NDK)
We recommend that you install the Android development tools using our product installer. However, if you already have the Android development tools installed on your system, or if you want to install the tools yourself, you can choose to skip this step in the installer. For specific instructions on installing the Android development tools yourself, see Installing the Android Development Tools.
### Default Installed Locations of the Android Development Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Default Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Development Kit (JDK)</td>
<td>C:\Program Files\Java\jdk1.7.0_25</td>
</tr>
<tr>
<td>Android Software Development Kit (SDK)</td>
<td>C:\Users\Public\Documents\Embarcadero\Studio\15.0\Platform SDKs\adt-bundle-windows-x86-20131030\sdk</td>
</tr>
<tr>
<td>Android Native Development Kit (NDK)</td>
<td>C:\Users\Public\Documents\Embarcadero\Studio\15.0\Platform SDKs\android-ndk-r9c</td>
</tr>
</tbody>
</table>

### Adding Your Android SDK in Tools Options > SDK Manager

**Important:**

- If the Android SDK and NDK are installed during the RAD Studio installation, your Android SDK should be automatically discovered by the RAD Studio **SDK Manager**, and you do not need to perform this step. Proceed to the next step: [Installing the USB Driver for Your Android Device](#).

- If you installed the Android SDK and NDK yourself, you need to perform this step so that RAD Studio can build apps that target Android devices.

1. Select **Tools > Options > Environment Options > SDK Manager**.
Here is the SDK Manager when it is fully populated with the recommended Android SDK:

![SDK Manager](image)

**Note:**

- The **SDK Manager** fields are blank if the SDK Manager cannot detect the default installed Android SDK and you have not yet added an Android SDK to the SDK Manager.

- Fields that are marked with ![warning] do not have the correct path. Click the [...] in that field and browse to the installed location of the library files.

2. Click **Add**.
3. On the Add a New SDK dialog box, click the down-arrow in the Select an SDK version field, and select Add New... from the drop-down list:

![Add a New SDK dialog box](image)

**Note:** If your installed Android SDK is listed in the Select an SDK version list, select your Android SDK.
4. The **Create a new Android SDK** wizard is displayed.

- Complete the first two fields on this wizard: **Android SDK Base Path** and **Android NDK Base Path** (their installed locations). Either enter the paths of the installed locations or click the ellipsis [...] and navigate to the installed locations.

- The symbol indicates that the base paths are missing, and the **Next** button is enabled only after these fields are filled.

- Here is the wizard when no SDKs have been added and the base path fields are empty:

![Create a new Android SDK wizard](image)

This wizard verifies the base paths you specify and then auto-populates the paths to the various other libraries (that is, the tools on page 2 of the wizard).

Thus, you might need to enter only the base paths for the NDK and SDK, on page 1 of the wizard. Then the second page will auto-populate, if given a little time. Or you might need to verify the auto-populated addresses (if they are present).
Here is the first page of the wizard after you browse to (or enter) the paths to the Android SDK and NDK.
There are no icons because the paths have been verified.
5. **Click Next** to go to the next page of the wizard, where the wizard will attempt to prefill the fields by discovery.
   - Here is the second page of the wizard with all its fields prefilled:

![Second page of the wizard with prefilled fields](image)

   - For any fields that do not prefill, click the ellipsis [...] button and browse to the installed location.

6. **Click Finish** to close the wizard.

You can view and manage your installed Android SDKs by opening the **SDK Manager**.
Installing the USB Driver for Your Android Device (Required)

Instructions and download sites are available for various Android devices here:

- [https://developer.amazon.com/sdk/fire/connect-adb.html](https://developer.amazon.com/sdk/fire/connect-adb.html) (Kindle Fire USB Drivers)

For example, for a Nexus 7 or Nexus 10 tablet, you install the Google USB Driver using the Android SDK Manager, as follows:

1. Start the **SDK Manager.exe** by selecting **Start | Programs | Embarcadero RAD Studio | Android SDKs | Android Tools**:

   Tip: If the Android SDK Manager does not start, run **android.bat** from the **\tools** directory inside your installed SDK directory.
2. In the Android SDK Manager, install the USB driver for your Nexus 7 or Nexus 10:

1. Select **Google USB Driver**, clear all other checkboxes, and click **Install 1 Package**:

2. Power up your Android device and connect it by USB cable to your development system.

3. In the Control Panel **Device Manager** on your development system, right-click your Nexus 7 or Nexus 10, and select **Update Driver Software**.

4. In the **Update Driver Software** dialog box, browse to the \*extras* folder in your Android SDK directory, and select the folder that matches your Nexus 7 or Nexus 10. (Select **google** for the Nexus 7 or Nexus 10.)

5. Click **OK**.
Creating an Android Emulator (Optional)

An Android emulator enables you to run an Android application on your development system instead of on a live Android device. An emulator can be useful during development cycles, but typically has significant performance problems not seen on live Android devices.

For steps to follow in creating and running an emulator, see Creating an Android Emulator.

See Also

- Installing the USB Driver for Your Android Device
- Enabling USB Debugging on an Android Device
- Configuring Your System to Detect Your Android Device
- Mobile Tutorial: Creating an Application for Mobile Platforms (iOS and Android)
- Installing the Android Development Tools
- Android Mobile Application Development
- Creating an Android Emulator
- Running Your Android Application on an Android Emulator
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. If you do not have an Android device, you can alternatively [create an Android emulator](#).

   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 reading this tutorial before starting 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:

![Structure View](image)

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:

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

![Code Editor comparison between Delphi and 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 your 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:
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:**

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

**C++ code:**

```cpp
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:

**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:
Test Your Android Application

Testing on the Android Emulator

Before running your Android app, confirm the target platform in the Project Manager:
Testing 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 target platform so that the Platform Assistant deploys the application to the connected iOS Device:

After you select the 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
- Mac OS X Application Development
- Mobile Code Snippets
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:**
  
<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>[ ]</td>
</tr>
<tr>
<td></td>
<td>[ ]</td>
</tr>
<tr>
<td></td>
<td>[ ]</td>
</tr>
</tbody>
</table>

-
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.
- 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.
Using TintColor 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.

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.

For the iOS target platform:

- FireMonkey provides buttons that correspond to the Apple Style Guide, and some buttons might not support the tint feature.
- For example, on iOS 7, segmented buttons have the TintColor property. (On iOS 6, the segmented buttons do not support the tint feature.)
When you change a button’s `StyleLookup` property, the Object Inspector automatically displays or hides the `TintColor` and `IconTintColor` properties as appropriate. The following image shows three TSpeedButtons on an Android app:

![Image showing three TSpeedButtons on an Android app]

**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.

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Segmented Control iOS" /></td>
<td><img src="image2.png" alt="Segmented Control Android" /></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Segmented Control iOS" /></td>
<td><img src="image4.png" alt="Segmented Control Android" /></td>
</tr>
</tbody>
</table>
2. Select the first component, and change its `StyleLookup` property to `segmentedbuttonleft`:

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="SegmentedButton Left iOS" /></td>
<td><img src="image2.png" alt="SegmentedButton Left Android" /></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="SegmentedButton Middle iOS" /></td>
<td><img src="image4.png" alt="SegmentedButton Middle Android" /></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.png" alt="SegmentedButton Text iOS" /></td>
<td><img src="image6.png" alt="SegmentedButton Text Android" /></td>
</tr>
</tbody>
</table>

6. Use the mouse to select these three buttons:

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image7.png" alt="Mouse Selection iOS" /></td>
<td><img src="image8.png" alt="Mouse Selection Android" /></td>
</tr>
</tbody>
</table>
7. Set the **GroupName** property to a unique name such as **LocationSegments**:

![Image of GroupName set to 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:

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

iOS

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

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)
- Android Mobile Application Development
- FMX.Mobile.Controls Sample (Delphi)
- 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

Android

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/emulator or your connected mobile device. After you tap TDateEdit, the calendar control appears, and you can select a date.

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:**

```delphi
procedure TForm25.DateEdit1Change(Sender: TObject);
begin
  ShowMessage(DateFormatTime('ddddd', DateEdit1.Date));
end;
```

**C++Builder:**

```cpp
void __fastcall TForm25::DateEdit1Change(TObject *Sender)
{
  ShowMessage(DateFormatTime("ddddd", DateEdit1->Date));
}
```
Mobile Tutorial: Using a Calendar Component to Pick a Date (iOS and Android)

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`
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:

<table>
<thead>
<tr>
<th>iOS 7 (iPad2)</th>
<th>Android (LG-E612)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arizona</td>
<td></td>
</tr>
<tr>
<td>Arkansas</td>
<td></td>
</tr>
<tr>
<td>California</td>
<td>○</td>
</tr>
<tr>
<td>Colorado</td>
<td></td>
</tr>
<tr>
<td>Connecticut</td>
<td>○</td>
</tr>
</tbody>
</table>

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:

![Image showing Object Inspector and TComboBox in Delphi](image)

7. Edit other items as well, such as Alaska, Arizona, Arkansas, California, Colorado, and so forth.

8. Run the application on your chosen mobile target platform (iOS Simulator (for Delphi only), iOS Device, Android Emulator, or Android Device). After you tap 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:**

```delphi
procedure TForm27.FormCreate(Sender: TObject);
begin
  ComboBox1.Items.Add('Alabama');
  ComboBox1.Items.Add('Alaska');
  ComboBox1.Items.Add('Arizona');
  ComboBox1.Items.Add('Arkansas');
  ComboBox1.Items.Add('California');
  // Other states can be listed here
  ComboBox1.Items.Add('Virginia');
  ComboBox1.Items.Add('Washington');
  ComboBox1.Items.Add('West Virginia');
  ComboBox1.Items.Add('Wisconsin');
  ComboBox1.Items.Add('Wyoming');
end;
```

**C++Builder:**

```cpp
void __fastcall TForm27::FormCreate(TObject *Sender)
{
  ComboBox1->Items->Add("Alabama");
  ComboBox1->Items->Add("Alaska");
  ComboBox1->Items->Add("Arizona");
  ComboBox1->Items->Add("Arkansas");
  ComboBox1->Items->Add("California");
  // Other states can be listed here
  ComboBox1->Items->Add("Virginia");
  ComboBox1->Items->Add("Washington");
  ComboBox1->Items->Add("West Virginia");
  ComboBox1->Items->Add("Wisconsin");
  ComboBox1->Items->Add("Wyoming");
}
```
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" in the following sample code), specify ItemIndex as follows:

**Delphi:**

```delphi
procedure TForm27.FormCreate(Sender: TObject);
begin
    ComboBox1.Items.Add('Alabama');
    ComboBox1.Items.Add('Alaska');
    ComboBox1.Items.Add('Arizona');
    ComboBox1.Items.Add('Arkansas');
    ComboBox1.Items.Add('California');
    // Other states can be listed here

    // Index of 5th item is "4"
    ComboBox1.ItemIndex := 4;
end;
```

**C++Builder:**

```cpp
void __fastcall TForm27::FormCreate(TObject *Sender)
{
    ComboBox1->Items->Add("Alabama");
    ComboBox1->Items->Add("Alaska");
    ComboBox1->Items->Add("Arizona");
    ComboBox1->Items->Add("Arkansas");
    ComboBox1->Items->Add("California");
    // Other states can be listed here

    // Index of 5th item is "4"
    ComboBox1->ItemIndex = 4;
}
```
If you do not know the index value, you can find the value by using the `IndexOf` method as follows:

**Delphi:**

```delphi
procedure TForm27.FormCreate(Sender: TObject);
begin
  ComboBox1.Items.Add('Alabama');
  ComboBox1.Items.Add('Alaska');
  ComboBox1.Items.Add('Arizona');
  ComboBox1.Items.Add('Arkansas');
  ComboBox1.Items.Add('California');
  // Other states can be listed here

  ComboBox1.ItemIndex := ComboBox1.Items.IndexOf('California');
end;
```

**C++Builder:**

```cpp
void __fastcall TForm27::FormCreate(TObject *Sender)
{
    ComboBox1->Items->Add("Alabama");
    ComboBox1->Items->Add("Alaska");
    ComboBox1->Items->Add("Arizona");
    ComboBox1->Items->Add("Arkansas");
    ComboBox1->Items->Add("California");
    // Other states can be listed here

    ComboBox1->ItemIndex = ComboBox1->Items->IndexOf("California");
}
```

### 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 **Fit**.
To implement an OnChange event handler:

1. Select the TComboBox 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:**

```delphi
procedure TForm27.ComboBox1Change(Sender: TObject);
begin
  Memo1.Lines.Insert(0, Format('Item %s at Index %d was selected.',
                                 [ComboBox1.Selected.Text,
                                  ComboBox1.ItemIndex]));
end;
```

**C++Builder:**

```cpp
void __fastcall TForm27::ComboBox1Change(TObject *Sender)
{
  Memo1->Lines->Insert(0, "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)**

```
Item California at Index 4 was selected.
```

**iOS6 (iPad)**

```
Item California at Index 4 was selected.
```

Options:

- Arizona
- Arkansas
- California
- Virginia
- Washington
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
- Mac OS X Application Development
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:

<table>
<thead>
<tr>
<th><code>TMultiView.Mode</code></th>
<th>Master Pane Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawer</td>
<td>Drawer (Push/Overlap)</td>
</tr>
<tr>
<td>Panel</td>
<td>Docked panel</td>
</tr>
<tr>
<td>PlatformBehaviour</td>
<td>(see the following table)</td>
</tr>
<tr>
<td>Popover</td>
<td>Popup menu</td>
</tr>
<tr>
<td>Custom</td>
<td>For details, see the <a href="#">Custom Mode</a> subsection.</td>
</tr>
</tbody>
</table>

**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`.

**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 popup 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.
**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:

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Device Orientation</th>
<th>Master Pane Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone</td>
<td>Landscape, Portrait</td>
<td>Drawer (push/overlap)</td>
</tr>
<tr>
<td>Tablet</td>
<td>Landscape</td>
<td>Docked panel</td>
</tr>
<tr>
<td>Tablet</td>
<td>Portrait</td>
<td>Drawer (push/overlap)</td>
</tr>
</tbody>
</table>
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:

   Delphi:

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

   C++Builder:

   ```cpp
   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


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.

Note: It is not recommended to put native controls on the detail pane because they do not support the Z-order of the form: a native control is always on top of other controls available on your form.
Setting the TMultiView Component Properties

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

- **TargetControl** = Panell
- **Mode** = Drawer
- 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.)

**Final Steps**

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:

```delphi
procedure TForm7.bStartCameraClick(Sender: TObject);
begin
    CameraComponent1.Active := true;
end;
```

```cpp
void __fastcall TForm7::bStartCameraClick(TObject *Sender)
{
    CameraComponent1->Active = true;
}
```
2. Double-click the **Stop Camera** button, and insert the following code:

**Delphi:**

```delphi
procedure TForm7.bStopCameraClick(Sender: TObject);
begin
  CameraComponent1.Active := false;
end;
```

**C++Builder:**

```cpp
void __fastcall TForm7::bStopCameraClick(TObject *Sender)
{
  CameraComponent1->Active = false;
}
```

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

**Delphi:**

```delphi
procedure TForm7.bFrontCameraClick(Sender: TObject);
begin
  CameraComponent1.Active := False;
  CameraComponent1.Active := True;
end;
```

**C++Builder:**

```cpp
void __fastcall TForm7::bFrontCameraClick(TObject *Sender) {
  // select Front Camera
  CameraComponent1->Active = false;
  CameraComponent1->Kind = TCameraKind::ckFrontCamera;
  CameraComponent1->Active = true;
}
```
4. Double-click the **Back** button, and insert the following code:

**Delphi:**

```delphi
procedure TForm7.bBackCameraClick(Sender: TObject);
begin
  CameraComponent1.Active := False;
  CameraComponent1.Active := True;
end;
```

**C++Builder:**

```cpp
void __fastcall TForm7::bBackCameraClick(TObject *Sender) {
  // select Back Camera
  CameraComponent1->Active = false;
  CameraComponent1->Kind = TCameraKind::ckBackCamera;
  CameraComponent1->Active = true;
}
```

**To implement the onSampleBufferReady event handler**

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

**Delphi:**

```delphi
procedure TForm7.CameraComponent1SampleBufferReady(Sender: TObject;
const ATime: Int64);
begin
  TThread.Synchronize(TThread.CurrentThread, GetImage);
end;
```

**C++Builder:**

```cpp
void __fastcall TForm7::CameraComponent1SampleBufferReady(TObject *
Sender,
const __int64 ATime)
{
  GetImage();
}
```

For the **TForm7** class, you should implement the private method **GetImage**. The onSampleBufferReady event handler calls this method to get the image from the device camera.
Mobile Tutorial: Using a MultiView Component to Display Alternate Views of Information (iOS and Android)

Do the following:

1. In the private section of the TForm7 class, declare the GetImage method:

   **Delphi:**
   ```
   private
   { Private declarations }
   procedure GetImage;
   ```

   **C++Builder:**
   ```
   private: // User declarations
   void __fastcall GetImage();
   ```

2. Implement the GetImage method as follows:

   **Delphi:**
   ```
   procedure TForm7.GetImage;
   begin
   CameraComponent1.SampleBufferToBitmap(imgCameraView.Bitmap, True);
   end;
   ```

   **C++Builder:**
   ```
   void __fastcall TForm7::GetImage()
   {
   CameraComponent1->SampleBufferToBitmap(imgCameraView->Bitmap,
   true);
   }
   ```
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 Tutorial: Using a MultiView Component to Display Alternate Views of Information (iOS and Android)

Mobile Product Samples that Use TMultiView

See C:\Users\Public\Documents\Embarcadero\Studio\15.0\Samples\Mobile Samples

  - MultiViewDemo
  - LocationDemo
  - MusicPlayer
  - Device Sensors and Services
  - SurfSpot Finder Sample (Delphi)

See Also

  - FMX.MultiView.TMultiView
  - FMX.MultiView.TMultiViewPresentation
Mobile Tutorial: Using the Web Browser Component (iOS and Android)

For mobile platforms, FireMonkey wraps the Web Browser component as the TWebBrowser component. 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:
   ![Back Button 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**:
   ![TEdit Component on 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. **Note**: If the **TWebBrowser** component is disabled, select the target platform for Android or iOS.

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".

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 Code Editor, create the following OpenURL new private method:

**Delphi:**

```delphi
type
 TForm1 = Class(TForm)
  ToolBar1: TToolBar;
  Button1: TButton;
  Edit1: TEdit;
  WebBrowser1: TWebBrowser;
private
  { Private declarations }
  procedure OpenURL;
public
  { Public declarations }
end;
```

**C++Builder:**

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

2. Implement the openURL method as follows:

**Delphi:**

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

**C++Builder:**

```cpp
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:**

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

**C++Builder:**

```cpp
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:**

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

**C++Builder:**

```cpp
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:

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**.

![Object Inspector](image)

**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 XE7** | **Samples** and navigate to **Object Pascal\Mobile Snippets\WebBrowser**
Mobile Tutorial: Using the Web Browser Component (iOS and Android)

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.Mobile.KeyboardTypes Sample (Delphi)
- FMX.Mobile.KeyboardTypes Sample (C++)
- 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 \texttt{TTabPosition} either to \texttt{Top} or to \texttt{PlatformDefault}).
  - Tabs traditionally display only text. However, FireMonkey allows you to specify custom icons to be displayed on tabs (see \texttt{Using Custom Multi-Resolution Icons for Your Tabs}).

- **On iOS:**
  - Tabs are typically shown at the bottom of the screen (so you should set \texttt{TTabPosition} either to \texttt{Bottom} or to \texttt{PlatformDefault}).
  - Tab items always display both text and an icon, which can be set via the \texttt{StyleLookup} property for each tab.

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

- In iOS apps, tabs are aligned at the lower edge of the \texttt{TTabControl}.
- In Android apps, tabs are aligned at the top edge of the \texttt{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. In the **Views** field on the Form Designer, create a view for each platform you want to support. This tutorial uses an iOS and an Android view.

3. Select **TTabControl** from the **Tool Palette**:
4. 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**):

**iOS**

![iOS TabControl](image)

**Android**

![Android TabControl](image)
5. 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**:

![Object Inspector](image1.png)

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

![Items Editor](image2.png)

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

![Items Designer](image3.png)
8. On the Form Designer, select the first TabItem and change its StyleLookup property:

iOS 7

Android
9. 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`:

10. To change the location of tabs, select the `TabPosition` property for the `TabControls` component. For each tab, you can select any of the following values of the `TabPosition` property 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.

<table>
<thead>
<tr>
<th>Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
</tr>
<tr>
<td>Android</td>
</tr>
</tbody>
</table>

Tabs are displayed at the Top.
<table>
<thead>
<tr>
<th>Bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>iOS</strong></td>
</tr>
<tr>
<td><strong>Android</strong></td>
</tr>
</tbody>
</table>

Tabs are displayed at the Bottom.
No Tabs are displayed. Instead, three Dots (\ldots\ldots\ldots) 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

<table>
<thead>
<tr>
<th></th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
</table>

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:

![Tabs Example](image)

**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 design device 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](image).

- The custom glyphs used in this section are available in a zip file that is delivered in your $(BDS)/Images/GlyFX directory. The three PNGs used here are located in the Aero 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. For a multi-device application, drop a TabControl component onto the Form Designer, set its Align property to Client, and add several tabs on the TabControl:

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

The initial empty Scale entry is assumed to have a scale of 1.000. Click the **Size** size selector, choose **Default size**, and set the size to **32 x 32**:
4. Repeat the following step to add any additional scales that you want to support:

   1. 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:
5. Click the Fill All from File button, navigate to the .png file you want to use, and click Open.

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


7. Repeat steps 2 - 6 for each of the remaining tabitems, and assign each tabitem a custom icon image.

8. In the Text property, change the text on each tab:
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:

In the Structure View, select Empty under CustomIcon:

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

![Bitmap Editor](image)

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

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

![Object Inspector](image)

**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**:
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 a multi-device application, place a TabControl, and add several tab items on it (TabItem1, TabItem2, and TabItem3).

2. From the Tool Palette, add a TButton to the form, and then add an ActionList component:
Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)

iOS:

Android:
3. In the Form Designer, activate the Master view. Then select the button component in the Object Inspector, and 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):

**iOS:**

![iOS diagram](image)

**Android:**

![Android diagram](image)
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`:

iOS:

![iOS screenshot](image)

Android:

![Android screenshot](image)
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. Change the size of the button to fit the new caption text, or change the CustomText property of ChangeTabAction1 component as shown here:

iOS:

[Image of iOS interface]

Android:

[Image of Android interface]

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

You can use any of the following three ways to change the active tab page from your source code:

Assign an instance of `TTabItem` to the `ActiveTab` property

**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`.

**Delphi:**

```
TabControl1.TabIndex := 1;
```

**C++:**

```
TabControl1->TabIndex = 1;
```

If `ChangeTabAction` is defined, you can execute an action from your code

**Delphi:**

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

// Call the action
ChangeTabAction1.Execute;
```

**C++:**

```
ChangeTabAction1->Tab = TabItem2;
ChangeTabAction1->Execute();
```
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)
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.
Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

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**:

![Form Design Screen](image)

**Note:** For more information about the selection of the **Style** and **Views**, see **Style Selector** and **Using FireMonkey Views**.
Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

Step 2: Adding Fields

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

![Add Field dialog box]

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 `TLListView.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 property and change its value to ImageListItemRightButton.

3. Optionally, you can apply a tint to the TLListView 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**:

![Structure View and Object Inspector](image)

**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 useful 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:**

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

**C++Builder:**

```cpp
void __fastcall TForm1::ListView1ButtonClick(TObject * const Sender,
    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.
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
- RAD Studio Tutorials
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

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android (LG E-612)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>Alabama</td>
</tr>
<tr>
<td>Alaska</td>
<td>Alaska</td>
</tr>
<tr>
<td>Arizona</td>
<td>Arizona</td>
</tr>
<tr>
<td>Arkansas</td>
<td>Arkansas</td>
</tr>
<tr>
<td>California</td>
<td>California</td>
</tr>
<tr>
<td>Colorado</td>
<td>Colorado</td>
</tr>
<tr>
<td>Connecticut</td>
<td>Connecticut</td>
</tr>
</tbody>
</table>

Select a State
Grouped List

![Image of a grouped list on an iPod](image)

**Note:** Only iOS devices support the grouped lists.
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:

![ListBox items](image)

**Add a Header**

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

![Header for TListBox](image)

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:

![Image of TLabel and TListBoxHeader](image)

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Align</td>
<td>Client</td>
</tr>
<tr>
<td>StyleLookup</td>
<td>toollabel</td>
</tr>
<tr>
<td>TextSettings.HorzAlign</td>
<td>Center</td>
</tr>
<tr>
<td>Text</td>
<td>(Text value as you want)</td>
</tr>
</tbody>
</table>

**Add a Group Header/Footer to the List**

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

![Image of ListBox with Group Header and Footer](image)

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**:

![Item Designer Screenshot]

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:

![Item Designer Screenshot]

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 (only for iOS target platform). This choice is controlled by the GroupingKind property and the StyleLookup property, as shown in the following graphic:

![Plain List and Grouped List Comparison]

Plain = GroupingKind Property Value

Grouped = GroupingKind Property Value

listboxstyle = StyleLookup Property Value

transparentlistboxstyle = StyleLookup Property Value

**Important:** For iOS devices, you can specify either style for your TListBox component in the Object Inspector. For Android devices, you can specify only the plain list.

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:

![Values for ItemData.Accessory](image)

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:

![Object Inspector for Icon](image)

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:

<table>
<thead>
<tr>
<th>StyleLookup property</th>
<th>Look &amp; Feel</th>
</tr>
</thead>
<tbody>
<tr>
<td>listBoxItemnodetail</td>
<td>ListBoxItem1</td>
</tr>
<tr>
<td>listBoxItembottomdetail</td>
<td>ListBoxItem2</td>
</tr>
<tr>
<td>listBoxItemrightdetail</td>
<td>ListBoxItem3</td>
</tr>
<tr>
<td>listBoxItemleftdetail</td>
<td>ListBoxItem4</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android (LG E-612)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPod 3:04 PM</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>aa</td>
<td></td>
</tr>
<tr>
<td>aaa</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>bb</td>
<td></td>
</tr>
<tr>
<td>bbb</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td></td>
</tr>
<tr>
<td>ccc</td>
<td></td>
</tr>
</tbody>
</table>
Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

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++:

```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::TAccesory>(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.
     - 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.

   - For the last TSpeedButton:
     - Set the Align property to Right.
     - Change the Name property to OverflowButton.
     - Select detailstoolbutton for the StyleLookup property.

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 and define Bitmap property and change Text property to the text value that you want.

5. For the last `TListBoxItem` set Accessory to `aMore` and Text to `More`.

6. Add a `TShadowEffect` component to the overflow menu.
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:**

```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
ListBox1.SendToBack;
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++:**

```cpp
void __fastcall TForm1::OverflowButtonClick(TObject *Sender) {
OverflowMenu->Visible = !(OverflowMenu->Visible); // change the visibility status
if (OverflowMenu->Visible) { // the Overflow Menu is displayed
ListBox1->SendToBack();
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:**

```delphi
procedure TForm1.SearchButtonClick(Sender: TObject);
begin
  SearchBox1.Visible := not SearchBox1.Visible; //change the visibility status
end;
```

**C++:**

```cpp
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.

<table>
<thead>
<tr>
<th>Android (Samsung Tab 2.0)</th>
<th>Android (Samsung Tab 2.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Displaying the overflow menu" /></td>
<td><img src="image2.png" alt="Displaying the search box" /></td>
</tr>
</tbody>
</table>
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 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 $\Rightarrow$ Run.

When you select a list box item, the rectangle’s color changes accordingly:
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**.

![Diagram showing layout for landscape and portrait orientations.]

**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.

![Diagram showing the use of margins.]

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

![iOS Edit Control Anchors Example](image)

Android

![Android Edit Control Anchors Example](image)

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:

![iOS Button Control Anchors Example](image)

Android:

![Android Button Control Anchors Example](image)
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
This functionality is provided as Actions, and you need to write only one line of code for each task.

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

**Actions** serve two purposes:

- An action can represent properties common to the user interface elements—such as whether a control is enabled or whether a check box is selected.
- An action can respond when a control fires—such as when the user clicks a button or chooses a menu item.

In this tutorial, you learn how to assign actions to user interface elements (such as a button) for each functionality that you want to support.

**Building the User Interface for the Application**

The user interface of this sample application is quite simple, as shown in the following picture:
Place the following components on the Form Designer:

- **ToolBar** component
  - On the toolbar, put three TButton components. Each button uses different icons.
  - Set the **StyleLookup** property for the three buttons to `cameratoolbuttonbordered`, `searchtoolbuttonbordered`, and `actiontooltbuttonbordered`, respectively.

- **TImage** component
  - Set the **Align** property to `Client`.

- **TActionList** component

**Taking a Picture with a Mobile Device Camera**

You can define an action to take a photo using the camera on your mobile device. Perform the following steps:

1. On the Form Designer, select the button (for taking a photo).
2. In the Object Inspector, select the drop-down list for the Action property.
3. Select **New Standard Action** | **Media Library** | **TTakePhotoFromCameraAction**:

4. On the Events tab, expand the Action node, and then double-click the **OnDidFinishTaking** event.
5. Add the following code to the **OnDidFinishTaking** event handler:

**Delphi:**

```delphi
procedure TForm1.TakePhotoFromCameraAction1DidFinishTaking(Image: TBitmap);
begin
  Image1.Bitmap.Assign(Image);
end;
```

**C++:**

```cpp
void __fastcall TForm1::TakePhotoFromCameraAction1DidFinishTaking(TBitmap *Image)
{
  Image1->Bitmap->Assign(Image);
}
```

This code assigns a picture taken from the mobile device camera to the **Bitmap** property of the **TImage** component.

### Using a Picture from the Mobile Device Photo Library

You can define an action to use a photo from the Photo Library with the following steps:

1. On the **Form Designer**, choose the button that you want to use (for picking up a photo).

2. In the **Object Inspector**, click the drop-down list for the Action property and select **New Standard Action | Media Library | TTakePhotoFromLibraryAction**.

3. In the **Events** tab, expand the **Action** node, and then double-click the **OnDidFinishTaking** event.
4. Add the following code to the **OnDidFinishTaking** event handler:

**Delphi:**

```delphi
procedure TForm1.TakePhotoFromLibraryAction1DidFinishTaking(Image: TBitmap);
begin
  Image1.Bitmap.Assign(Image);
end;
```

**C++:**

```cpp
void __fastcall TForm1::TakePhotoFromLibraryAction1DidFinishTaking(TBitmap *Image)
{
  Image1->Bitmap->Assign(Image);
}
```

The code above assigns a picture taken from the Photo Library to the **Bitmap** property of the **TImage** component.

**Enabling Image Editing**

You can enable the **Editable** property of the actions **TTakePhotoFromCameraAction** and **TTakePhotoFromLibraryAction** to allow cropping the image after taking or loading it.

To enable the **Editable** property follow the steps:

1. On the **Structure panel**, go to the action:
   1. **ActionList1 | TakePhotoFromCameraAction1**
   2. **ActionList1 | TakePhotoFromLibraryAction1**
2. On the **Object Inspector** set the **Editable** property to **True**.

After enabling the **Editable** property you can crop the photos on iOS and Android.
Mobile Tutorial: Taking and Sharing a Picture, and Sharing Text (iOS and Android)

On iOS Devices:

- **Loading the Picture**
- **Zooming the Picture**
- **Cropping the Picture**

On Android Devices:

- **Loading the Picture**
- **Android Menu for the Action**
- **Cropping the Picture**
Sharing or Printing a Picture

From your application, you can share a photo on social networking sites (such as Facebook and Twitter), you can send the picture to a printer, use the picture as an attachment to e-mail, assign it to Contacts, and so on.

This multi-share service is called Share Sheet Functionality, and you can implement this functionality using the following steps:

1. On the Form Designer, select the button (for sharing a photo).

2. In the Object Inspector, click the drop-down list for the Action property, and select New Standard Action | Media Library | TShowShareSheetAction.

3. On the Events tab, expand the Action node, and then double-click the OnBeforeExecute event.
4. Add the following code to the **OnBeforeExecute** event handler:

**Delphi:**

```delphi
procedure TForm1.ShowShareSheetAction1BeforeExecute(Sender: TObject);
begin
  ShowShareSheetAction1.Bitmap.Assign(Image1.Bitmap);
end;
```

**C++:**

```cpp
void __fastcall TForm1::ShowShareSheetAction1BeforeExecute(TObject *Sender)
{
    ShowShareSheetAction1->Bitmap->Assign(Image1->Bitmap);
}
```

The code above assigns a picture on the TImage component to "Share Sheet Functionality".

After you select **Facebook** from the list of services, you can post the picture on Facebook with your comment:

![Facebook Share Sheet](image)

---

**Note:** In this subsection, screenshots of iOS devices are used as an example.

For a sample application that uses share sheet functionality, see the [FMX.Mobile.PhotoEditorDemo Sample](FMX.Mobile.PhotoEditorDemo) [Delphi].
Sharing Text

From your application, you can share text using the mobile device's share sheet. The applications that appear in the share sheet depend on the device:

- On an iOS device, a list of suitable sharing options is shown depending on the content that your app wants to share, but there is not a full app-sharing mechanism.
- On an Android device, the choices are dependent on sharing options and installed apps.

From your application, you can share text on social networking sites (such as Facebook and Twitter), you can send it by email, SMS (Short Message Service), and other available methods.

You can implement this functionality with the multi-share service called Share Sheet Functionality. The TShowShareSheetAction is the standard action for sharing images and text. TShowShareSheetAction shows the available sharing options depending on the type of content you are sharing. That is, the options shown for Text are different than for a Bitmap.

This example shows a simple implementation of this functionality. We share the text that the user has typed into a Memo.

1. Create a multi-device application. In our example, the form is named Demonstration.

2. Place the following components on the Form Designer:
   - TToolBar component
     - On the toolbar, add a TButton component.
     - Set the StyleLookup property for the button as follows:
       - actiontoolbuttonbordered or actiontoolbutton for iOS
       - actiontoolbutton for Android

   Note: The actiontoolbuttonbordered property includes the button with a border, while actiontoolbutton shows the button without a border.
   - TMemo component
   - TActionList component
After you have added the components to the sample application:

1. On the **Form Designer**, select the button (for sharing the text).

2. In the **Object Inspector**, click the drop-down list for the **Action** property, and select `New Standard Action | Media Library | TShowShareSheetAction`.

3. On the **Events** tab, expand the **Action** node, and then double-click the **OnBeforeExecute** event.
4. Add the following code to the `OnBeforeExecute` event handler:

**Delphi:**

```delphi
procedure TDemonstration.ShowShareSheetAction1BeforeExecute(Sender: TObject);
begin
end;
```

**C++:**

```cpp
void __fastcall TDemonstration::ShowShareSheetAction1BeforeExecute(TObject *Sender)
{
  ShowShareSheetAction1->TextMessage = Memo1->Lines->Text;
}
```

The code above uses the `TextMessage` property to assign the text to be shared, as previously typed in a `TMemo`.

**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.Mobile.PhotoEditorDemo Sample (Delphi)
- [http://appleinsider.com/articles/12/02/16/share_sheets](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.

- 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 to show the title "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:**

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

**C++:**

```cpp
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:**

```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++:**

```cpp
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 \texttt{OnLocationChanged} as follows:

**Delphi:**

```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++:**

```cpp
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("https://maps.google.com/maps?q=%2.6f,%2.6f",
        NewLocation.Latitude, NewLocation.Longitude);
    FormatSettings.DecimalSeparator = LDecSeparator;
    WebBrowser1->Navigate(URLString);
}
```

**Use Reverse Geocoding**

\texttt{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 \texttt{TGeocoder} to "Reverse Geocode" our location (in Latitude and Longitude) to readable address information.

Here is the basic sequence of actions with \texttt{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:**

```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)

```cpp
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:**

```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
        if Assigned(FGeocoder) then
      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++:

```cpp
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:

```delphi
procedure TForm1.OnGeocodeReverseEvent(const Address: TCivicAddress); begin
    ListBoxItemAdminArea.ItemData.Detail := Address.AdminArea;
    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;
```
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;
    }
}

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
Mobile Tutorial: Using Notifications (iOS and Android)

This tutorial describes the basic steps to use notifications on your mobile device.

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 requires dismissal by the user.

iOS Badge Number and Android Notification Number

iOS Badge Number

iPad

Android Notification Number

Android
Notification Banner on Mobile Devices

iOS

![iOS Notification Banner Example](image)

Android

![Android Notification Banner Example](image)

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

![iOS Notification Center Example](image)
Mobile Tutorial: Using Notifications (iOS and Android)

Access the Notification Service

On the mobile platforms, FireMonkey provides the `TNotificationCenter` component that allows you to easily access the Notification Service.

To access the notification service, do the following:

- Create a new mobile application:
  - File > New > Multi-Device Application-Delphi
  - File > New > Multi-Device Application-C++Builder
- For Delphi applications, add the following 2 units to the uses clause if they are not present:
  ```
  uses
  FMX.Platform, FMX.Notification;
  ```
- For C++ applications, add the following `include` operators to the project header file:
  ```
  #include <FMX.Platform.hpp>
  #include <FMX.Notification.hpp>
  ```
- Select the `TNotificationCenter` component in the Tool Palette, and drop it on the Form Designer. Note: If the `TNotificationCenter` component is disabled, select the target platform for Android or iOS.
The `FMX.Notification.TNotificationCenter.CreateNotification` method allows you to create an instance of the `TNotification` class object.

**Note:** In your source code, call the following methods in different **OnClick** events of **TButton** components.

**Set the Icon Badge Number and Notification Number from Code**

`TNotification` 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 can implement the following methods:

**Delphi:**

```delphi
procedure TForm1.SetNumber;
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;
```
C++:

```cpp
void __fastcall TForm1::setNumber()
{
    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 you set the MyNotification.Number field to 18, you can see the following:

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

![iOS badge](image)

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

![Android badge](image)
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:

**Delphi:**

```delphi
procedure TForm1.ScheduleNotification;
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;
```

**C++:**

```cpp
void __fastcall TForm1::scheduleNotification()
{
  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 you set the Notification Message (AlertBody), you can see it at the top of your device Home Screen. For iOS devices, this message is similar to the following screen:

![iOS Notification Example](image)

**Update or Cancel a Scheduled 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 only those notifications that have not been presented yet. If a notification has already appeared in the **Device Notification List**, you can neither update nor cancel the notification on iOS.

To update a scheduled notification, simply call the **ScheduleNotification** method again with an instance of **TNotificationCenter** that has the same name (Name property).

To cancel a scheduled notification, you can simply call the **CancelNotification** method with the identifier you used:

**Delphi:**

```delphi
procedure TForm1.CancelNotification;
begin
 NotificationCenter1.CancelNotification('MyNotification');
end;
```

**C++:**

```cpp
void __fastcall TForm1::cancelNotification()
{
  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.

**Delphi:**

```delphi
procedure TForm1.PresentNotification;
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;
```

**C++:**

```cpp
void __fastcall TForm1::presentNotification()
{
  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();
    }
  }
}
```
Notification Banner or Notification Alert

By default, your application shows the notification banner:

- **Notification Banner on iPad**
  - ![Notification Banner on iPad](image1)

- **Notification Banner on Android devices**
  - ![Notification Banner on Android devices](image2)

- **Notification Alert (only for iOS devices)**
  - ![Notification Alert (only for iOS devices)](image3)
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:

Alerts require an action before proceeding. Banners appear at the top of the screen and go away automatically.

- Badge App Icon
- Sounds
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.

**Delphi:**

```delphi
procedure TForm1.ActionNotification;
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;
```

**C++:**

```c++
void __fastcall TForm1::actionNotification()
{
  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."

**Delphi:**

```delphi
procedure TForm1NotificationCenter1ReceiveLocalNotification(Sender: TObject; ANotification: TNotification);
begin
  ShowMessage('The ' + ANotification.Name + ' notification clicked.' );
end;
```

**C++:**

```cpp
void __fastcall TForm1::NotificationCenter1ReceiveLocalNotification(TObject *Sender, TNotification *ANotification)
{
  ShowMessage("The " + ANotification->Name + " notification clicked.");
}
```
See Also

- 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)
- FMX.Notification.IFMXNotificationCenter
- Mobile Code Snippets: Notifications
  - SendCancelNotifications
  - SetResetBadgeNumber (iOS only)
- Creating events Index
- Using the Mac OS X Notification Center
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 and providers:

- Protocols:
  - iOS: Apple Push Notification APN
  - Android: Google Cloud Messaging GCM

- Providers:
  - Parse
  - Kinvey

In order to receive push notifications, you need to set up the messaging service (APS or GCM), the device, the cloud service provider, 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).
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.

- Setting Up the Messaging Service
- Using the Cloud Service
- Creating the Application

See Also

- Mobile Tutorial: Using Notifications (iOS and Android)

Code Samples

- BaaS ToDo Sample
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:**

```delphi
uses
  FMX.Platform, FMX.PhoneDialer;
```

**C++:**

```cpp
#include <FMX.Platform.hpp>
#include <FMX.PhoneDialer.hpp>
```

- Only for Delphi apps: Add the following line to the public section of the form definition:

```delphi
constructor Create(AOwner: TComponent); override;
```

- Add the following properties to the private section of the form definition:

**Delphi:**

```delphi
private: // User declarations
  PhoneDialerService: IFMXPhoneDialerService;
```

**C++:**

```cpp
private: // User declarations
  _di_IFMXPhoneDialerService phoneDialerService;
  bool serviceSupported;
```

- Only for Delphi apps: in the implementation section, override the form constructor as follows:

```delphi
constructor TForm1.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService, INTERFACE(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:

```cpp
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:**

```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;
    end;
end;
```

**C++:**

```c++
void __fastcall TForm1::btnGetCarrierInfoClick(TObject *Sender)
{
    if (serviceSupported) {
        lblCarrierName->Text = "Carrier Name: " + phoneDialerService->GetCarrier()->GetCarrierName();
    } 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**

- Carrier Name: MegaFon
- ISO Country Code: ru

**Android**

- Carrier Name: MegaFon
- ISO Country Code: ru

### 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:**

   ```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;
   end
   end
   ```

   **C++:**

   ```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.");
   }
   ```
Mobile Tutorial: Using the Phone Dialer on Mobile Devices (iOS and Android)

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**

![iOS Phone Dialer](image)

**Android**

![Android Phone Dialer](image)
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 "+").

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No call state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected</td>
<td>The phone caller is connected to the called party.</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Incoming</td>
<td>An incoming phone call.</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Dialing</td>
<td>The phone is in a dialing state.</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Call is disconnected.</td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

**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**:

   ```delphi
   procedure MyOnCallStateChanged(const ACallID: String; const ACallState: TCallState);
   ```

   **C++**:

   ```cpp
   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:**

```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++:**

```cpp
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:**

```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++:

```cpp
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:

![iOS screen](image)

**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)
- FMX.PhoneDialer.IFMXPhoneDialerService
- Android Mobile Application Development
- iOS Mobile Application Development
- Mobile Code Snippets: Phone Dialer
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 XE7 Professional or higher
    - Delphi XE7 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\InterBaseXE3.

    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 **Uses 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

<table>
<thead>
<tr>
<th>Atlantic Spadefish</th>
<th>Chaetodipterus</th>
<th>Myliobatis</th>
<th>Bat Ray</th>
<th>Myripristis californica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Angelfish</td>
<td>Pomacanthus</td>
<td>Blue Angelfish</td>
<td>Pomacanthus</td>
<td>Pomacanthus naso</td>
</tr>
<tr>
<td>Bluehead Wrasse</td>
<td>Thalassoma</td>
<td>Bluehead Wrasse</td>
<td>Thalassoma</td>
<td>hilarius</td>
</tr>
<tr>
<td>Cabezon</td>
<td>Scorpaenidae</td>
<td>Cabezon</td>
<td>Scorpaenidae</td>
<td>scorpaenidae</td>
</tr>
<tr>
<td>California Moray</td>
<td>Gymnothorax</td>
<td>California Moray</td>
<td>Gymnothorax</td>
<td>moringa</td>
</tr>
<tr>
<td>Clown Triggerfish</td>
<td>Balistidae</td>
<td>Clown Triggerfish</td>
<td>Balistidae</td>
<td>conspicillum</td>
</tr>
<tr>
<td>Dog Snapper</td>
<td>Leiognathus</td>
<td>Dog Snapper</td>
<td>Leiognathus</td>
<td>ferox</td>
</tr>
<tr>
<td>Firefish</td>
<td>Pterois</td>
<td>Firefish</td>
<td>Pterois</td>
<td>volitans</td>
</tr>
<tr>
<td>French Grunt</td>
<td>Haemulon</td>
<td>French Grunt</td>
<td>Haemulon</td>
<td>flavolineatum</td>
</tr>
<tr>
<td>Giant Maori Wrasse</td>
<td>Cheilinus</td>
<td>Giant Maori Wrasse</td>
<td>Cheilinus</td>
<td>undulatus</td>
</tr>
<tr>
<td>Great Barracuda</td>
<td>Sphyraena</td>
<td>Great Barracuda</td>
<td>Sphyraena</td>
<td>barracuda</td>
</tr>
<tr>
<td>Lingcod</td>
<td>Opisthurus</td>
<td>Lingcod</td>
<td>Opisthurus</td>
<td>octopinna</td>
</tr>
<tr>
<td>Lumpsucker Roeckcod</td>
<td>Zanclus</td>
<td>Lumpsucker Roeckcod</td>
<td>Zanclus</td>
<td>lucius</td>
</tr>
<tr>
<td>Nassau Grouper</td>
<td>Epinephelus</td>
<td>Nassau Grouper</td>
<td>Epinephelus</td>
<td>ventralis</td>
</tr>
<tr>
<td>Nurse Shark</td>
<td>Carcharodon</td>
<td>Nurse Shark</td>
<td>Carcharodon</td>
<td>sobreroi</td>
</tr>
<tr>
<td>Ornate Butterflyfish</td>
<td>Chaetodon</td>
<td>Ornate Butterflyfish</td>
<td>Chaetodon</td>
<td>ornata</td>
</tr>
<tr>
<td>Red Emperor</td>
<td>Lutjanus</td>
<td>Red Emperor</td>
<td>Lutjanus</td>
<td>salmone</td>
</tr>
<tr>
<td>Redband Parrotfish</td>
<td>Sparisoma</td>
<td>Redband Parrotfish</td>
<td>Sparisoma</td>
<td>auratanum</td>
</tr>
<tr>
<td>Redtail Surperon</td>
<td>Amphiprion</td>
<td>Redtail Surperon</td>
<td>Amphiprion</td>
<td>chrysopterus</td>
</tr>
</tbody>
</table>

**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:

6. On the Tool Palette, double-click the TFDConnection component.

7. Right-click the TFDConnection component and choose Connection Editor.

8. 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\15.0\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.

4. 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](#).

5. Add a **TFDQuery** component to the form.

6. Right-click the **TFDQuery** component and choose **Query Editor**.
   1. Write in the **SQL Command Text editor**
      
      ```sql
      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.

7. In the [Object Inspector](#), set the **Active** property of the **TFDQuery** component to **True**.
8. 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`.

9. Add a `TFDPhysIBDriverLink` component to the form.

10. 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\15.0\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 and the Database File to Mobile**

To execute your application on mobile, you need to deploy the following files:

- Interbase ToGo
- 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\15.0\Samples\Data, select the database dbdemos.gdb and click Open.

   - Navigate to the database location C:\Users\Public\Documents\Embarcadero\Studio\15.0\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_nnnnnnnn.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\InterBaseXE3`), 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 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\[Image]
   - Remote Path on Android platform: assets\internal\[Image]

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.

**Delphi:**

```delphi
procedure TForm1.FDConnection1BeforeConnect(Sender: TObject);
begin
  {$IFDEF DEFINED(iOS) or DEFINED(ANDROID)}
  FDConnection1.Params.Values['Protocol'] := 'Local';
  FDConnection1.Params.Values['Database'] :=
    TPath.Combine(TPath.GetDocumentsPath, 'dbdemos.gdb');
  {$ENDIF}
end;
```

The TPath record is declared in System.IOUtils unit, so you need to add System.IOUtils in the uses clause.
C++:

```cpp
void __fastcall TForm1::SQLConnection1BeforeConnect(TObject *Sender)
{
    #if ((defined(__arm__) && defined(__APPLE__)) || defined(__ANDROID__))
    FDConnection1->Params->Values["Protocol"] = "Local";
    FDConnection1->Params->Values["Database"] =
    System::IUtils::TPath::Combine(System::IUtils::TPath::GetDocumentsPath(),
                                     "dbdemos.gdb");
    #endif
}
```

You need to add `#include <System.IOUtils.hpp>`. 
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**

- Atlantic Spadefish
- Blue Angelfish
- Clown Triggerfish
- Firefish
- Ornate Butterflyfish
- Redband Parrotfish

**Android**

- Search Box
- Atlantic Spadefish
- Blue Angelfish
- Clown Triggerfish
- Firefish
- Ornate Butterflyfish
- Redband Parrotfish
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:

   ```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++:

   ```cpp
   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 in RAD Studio
- IBLite and IBToGo Licensing
- Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)
- Mobile Tutorial: Using FireDAC and SQLite (iOS and Android)
- Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)
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 XE7 Professional or higher
  - Delphi XE7 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\redis\InterBaseXE3.

Note: On Android devices, InterBase ToGo apps require specific Uses 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

- **Atlantic Spadefish**
- **Bat Ray**
- **Blue Angelfish**
- **Bluehead Wrasse**
- **Cabezon**
- **California Moray**
- **Clown Triggerfish**
- **Dog Snapper**
- **Firefish**
- **French Grunt**
- **Giant Maori Wrasse**
- **Great Barracuda**
- **Lingcod**
- **Lunartail Rockcod**
- **Nassau Grouper**
- **Nurse Shark**
- **Omane Butterflyfish**
- **Red Emperor**
- **Redband Parrotfish**
- **Redtail Surpero**

### Android

- **Atlantic Spadefish**
- **Bat Ray**
- **Blue Angelfish**
- **Bluehead Wrasse**
- **Cabezon**
- **California Moray**
- **Clown Triggerfish**
- **Dog Snapper**
- **Firefish**
- **French Grunt**
- **Giant Maori Wrasse**
- **Great Barracuda**
- **Lingcod**
- **Lunartail Rockcod**
- **Nassau Grouper**
- **Nurse Shark**
- **Omane Butterflyfish**
- **Red Emperor**
- **Redband Parrotfish**
- **Redtail Surpero**

**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\15.0\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:

![Value List Editor](image)

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, \Users\Public\Documents\Embarcadero\Studio\15.0\Samples\Data\dbdemos.gdb). On the mobile Device, the application is sandboxed, 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\15.0\Samples\Data, select the database dbdemos.gdb and click Open.
   
   - Navigate to the database location C:\Users\Public\Documents\Embarcadero\Studio\15.0\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\InterBaseXE3), 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:**

   ```delphi
   procedure TForm1.SQLConnection1BeforeConnect(Sender: TObject);
   begin
     {$IFDEF DEFINED(iOS) or DEFINED(ANDROID)}
     SQLConnection1.Params.Values['Database'] :=
       TPath.Combine(TPath.GetDocumentsPath, 'dbdemos.gdb');
     {$ENDIF}
   end;
   ```
C++:

```cpp
void __fastcall TForm1::SQLConnection1BeforeConnect(TObject *Sender)
{
  #if (defined(__arm__) && (defined(__APPLE__) || defined(__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.
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:

```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++:

```cpp
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
- 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: 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.
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 HD 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\15.0\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. (To display **Employees.s3db** or other *.s3db files in the Open dialog, set the **All Files (*.*)** option.)

   - Set the **LockingMode** parameter to **normal**.

   ![FireDAC Connection Editor](image)

   - 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 **TextSettings** 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

Use 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.

![LiveBindings Wizard](image)

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 app’s user interface 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:

```delphi
procedure TForm1.ListView1ItemClick(const Sender: TObject; const AItem: TListViewItem);
begin
    ButtonDelete.Visible := ListView1.Selected <> nil;
end;
```

C++:

```cpp
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.

**Delphi:**

- Declare the following procedure in the private section:

  ```delphi
  private
    procedure OnInputQuery_Close(const AResult: TModalResult; const AValues: array of string);
  end;
  ```

- Add the following procedure:

  ```delphi
  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:

  ```delphi
  procedure TForm1.ButtonAddClick(Sender: TObject);
  var
    Values: array[0 .. 0] of String;
  begin
    Values[0] := String.Empty;
    InputQuery('Enter New Item', ['Name'], Values, Self.OnInputQuery_Close);
  end;
  ```
**C++:**

To replicate the same functionality in C++, additional steps are required:

- Add the following definition after the TForm1 definition:

```cpp
typedef void __fastcall(_closure * TInputCloseQueryProcEvent)
    (const System::Uitypes::TModalResult AResult,
    System::UnicodeString const *AValues, const int AValues_High);
```

- Add the following class definition:

```cpp
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:

```cpp
private:
    void __fastcall OnInputQuery_Close
        (const System::Uitypes::TModalResult AResult,
        System::UnicodeString const *AValues, const int AValues_High);
```
o Add the following code:

```cpp
void __fastcall TForm1::OnInputQuery_Close(const System::Uitypes::TModalResult AResult,
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);
}
```

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

```cpp
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);
    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.

**Delphi:**

```delphi
procedure TForm1.ButtonDeleteClick(Sender: TObject);
var
  TaskName: String;
begin
  TaskName := 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;
```

**C++:**

```c++
void __fastcall TForm1::ButtonDeleteClick(TObject *Sender) {
  String TaskName = 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 IFDGUILxWaitCursor and IFDPhysDriver interfaces to your application.

For this, drop the TFDGUlxWaitCursor 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\15.0\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\15.0\Samples\Data, select the database shoplist.s3db and click Open.
   - Navigate to the database location C:\Users\Public\Documents\Embarcadero\Studio\15.0\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 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:

#### Delphi:

```delphi
procedure TForm1.FDConnection1BeforeConnect(Sender: TObject);
begin
  {$IFDEF DEFINED(iOS) or DEFINED(ANDROID)}
  FDConnection1.Params.Values['Database'] :=
    TPath.Combine(TPath.GetDocumentsPath, 'shoplist.s3db');
  {$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.

#### C++:

```c++
void __fastcall TForm1::FDConnection1BeforeConnect(TObject *Sender) {
  #ifdef __ANDROID__ || defined(__APPLE__)
    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:

**Delphi:**

```delphi
procedure TForm1.FDConnection1AfterConnect(Sender: TObject);
begin
  FDConnection1.ExecSQL('CREATE TABLE IF NOT EXISTS Item (ShopItem TEXT NOT NULL)');
end;
```

**C++:**

```cpp
void __fastcall TForm1::FDConnection1AfterConnect(TObject *Sender) {
  FDConnection1->ExecSQL("CREATE TABLE IF NOT EXISTS Item (ShopItem TEXT NOT NULL)"usz);
}
```
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 your 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
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.
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:

![Add SQLite Connection](image)

2. Define the name of the connection, such as ShoppingList.

![Modify Connection: ShoppingList (SQLite)](image)

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:

![Advanced Properties dialog box](image)

**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:

![Test connection succeeded](image)

**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](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 addtoolbuttonbordered.
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.

   ![Data Explorer](image)

   **Note:** This creates two components (ShoppinglistConnection: TSQLConnection and ItemTable: TSQLDataSet) on the form.

   ![Components](image)

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.

5. Select **ShopItem** in the **Item** component and drag **ShopItem** to **Item.Text** of **ListBox1**.

Following these steps connects the app's user interface 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:**

```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:

```cpp
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:
     ```sql
     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:**

```delphi
procedure TForm1.ButtonAddClick(Sender: TObject);
var
  Values: array[0 .. 0] of string;
begin
  Values[0] := String.Empty;
  InputQuery('Enter New Item', ['Name'], Values, 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:

```delphi
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 (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:

```cpp
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);
  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 Form Designer, double-click the **ButtonDelete** component. Add the following code to this event handler.

**Delphi:**

```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:

```cpp
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\15.0\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\15.0\Samples\Data, select the database shoplist.s3db and click **Open**.
   - Navigate to the database location C:\Users\Public\Documents\Embarcadero\Studio\15.0\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 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\assets\internal\assets\shoplist.s3db
   - **Remote Path** on Android platform: assets\internal\assets\shoplist.s3db

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:**

```delphi
procedure TForm1.ShoppinglistConnectionBeforeConnect(Sender: TObject);
begin
  {$IFDEF Defined(iOS) or Defined(ANDROID)}
  ShoppinglistConnection.Params.Values['ColumnMetadataSupported'] := 'False';
  ShoppinglistConnection.Params.Values['Database'] :=
    TPath.Combine(TPath.GetDocumentsPath, 'shoplist.s3db');
  {$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:**

```cpp
void __fastcall TForm1::ShoppinglistConnectionBeforeConnect(TObject *Sender) {
  #if defined(__ANDROID__) || defined(__APPLE__)
    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:**

```delphi
procedure TForm1.ShoppinglistConnectionAfterConnect(Sender: TObject);
begin
    ShoppinglistConnection.ExecuteDirect('CREATE TABLE IF NOT EXISTS Item
(ShopItem  TEXT NOT NULL)');
end;
```

**C++Builder:**

```cpp
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 your target platform.

![Project Manager screenshot](image-url)
2. Choose either of the following commands:
   - Run > Run
   - Run > Run Without Debugging

**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
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 XE3 | 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:

![DataSnap Server Wizard Steps]

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.
Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)

**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\15.0\Samples\Data\EMPLOYEE.GDB.

- Change the **Connected** property to **True**. If you get an error, double-check the **Driver** properties:

  ![Diagram of database 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.

In the Form Designer, open **ServerContainerUnit**.

Select **DSServerClass1**, and update the existing event handler for the **OnGetClass** event. Add the following code to the **DSServerClass1** event handler:

**Delphi:**

```delphi
procedure TServerContainer1.DSServerClass1GetClass(DSServerClass: TDSServerClass;
  var PersistentClass: TPersistentClass);
begin
  PersistentClass := TDSServerModule_EMPLOYEE;
end;
```
C++ (only for iOS):

```cpp
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` 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 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 DataSnap server's Server Module.
   - 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:

![Add Featured Files](image)

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