© 2014 Embarcadero Technologies, Inc. Embarcadero, the Embarcadero Technologies logos, and all other Embarcadero Technologies product or service names are trademarks or registered trademarks of Embarcadero Technologies, Inc. All other trademarks are property of their respective owners.

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.

June, 2014
# CONTENTS

Mobile Tutorials: Mobile Application Development (iOS and Android) .......... 11
Setup..................................................................................................................... 12
Using Basic User Interface Elements................................................................. 12
Using Device Functionality................................................................................. 14
Accessing a Database ......................................................................................... 15
See Also .................................................................................................................. 15

Mobile Tutorial: Set Up Your Development Environment on the Mac (iOS) ...... 17
Requirements on the Mac and iOS........................................................................ 17
Steps to Configure Your Mac to Run Your iOS Application .............................. 18
  Step 1: Install the Platform Assistant .............................................................. 18
  Step 2: Run the Platform Assistant ................................................................. 19
  Step 3: Install Xcode on the Mac ................................................................. 21
Next Steps ............................................................................................................. 21

Additional Steps to Configure Your Mac to Run Your iOS Application on Your iOS
Device .................................................................................................................. 22
Step 1: Install the Xcode Command Line Tools on a Mac ................................. 22
Step 2: Sign Up for a Developer Account ......................................................... 23
Step 3: Request, Download and Install Your Development Certificate .............. 23
   Request, Download and Install Your Certificate ........................................... 24
Step 4: Register Your Device for Deployment .................................................. 25
Step 5: Create and Install a Provisioning Profile .............................................. 25
See Also .................................................................................................................. 26

Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS) ... 28
Setting Up Your RAD Studio Environment.......................................................... 28
  Create a Connection Profile for the Mac ......................................................... 28
  Add an SDK to the Development System for the iOS Device Connected to
  the Mac .............................................................................................................. 31
See Also .................................................................................................................. 33

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) ....................... 41
Creating an Android Emulator (Optional)......................................................... 42
See Also .................................................................................................................. 42

Mobile Tutorial: Creating a FireMonkey Mobile Application (iOS and Android) 44
Before You Start ................................................................................................... 44
Step 1: Create a New FireMonkey Application for Android or iOS .................. 44
Step 2: Place Components on the FireMonkey Mobile Form ............................ 47
Step 3: Write an Event Handler for a Button Click by the User ....................... 52
<table>
<thead>
<tr>
<th>Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Creating the Project</td>
<td>111</td>
</tr>
<tr>
<td>Step 2: Adding Fields</td>
<td>113</td>
</tr>
<tr>
<td>Step 3: Creating LiveBindings</td>
<td>114</td>
</tr>
<tr>
<td>Step 4: Adding More Fields</td>
<td>117</td>
</tr>
<tr>
<td>Step 5: Adding the onButtonClick Event Handler</td>
<td>118</td>
</tr>
<tr>
<td>The Results</td>
<td>119</td>
</tr>
<tr>
<td>See Also</td>
<td>120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)</th>
<th>122</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using ListBox Components to Display a Table View in Mobile Applications</td>
<td>122</td>
</tr>
<tr>
<td>Plain List</td>
<td>123</td>
</tr>
<tr>
<td>Grouped List</td>
<td>124</td>
</tr>
<tr>
<td>Search Box</td>
<td>124</td>
</tr>
<tr>
<td>Create Items on the ListBox Component</td>
<td>126</td>
</tr>
<tr>
<td>Add a Header</td>
<td>129</td>
</tr>
<tr>
<td>Add a Group Header/Footer to the List</td>
<td>130</td>
</tr>
<tr>
<td>Show List Items as Separate Grouped Items</td>
<td>131</td>
</tr>
<tr>
<td>Add a Check Box or Other Accessory to a ListBox Item</td>
<td>132</td>
</tr>
<tr>
<td>Add an Icon to a ListBox Item</td>
<td>133</td>
</tr>
<tr>
<td>Add Detail Information to an Item</td>
<td>133</td>
</tr>
<tr>
<td>Running Your Application</td>
<td>134</td>
</tr>
<tr>
<td>Create Your ListBox Application</td>
<td>134</td>
</tr>
<tr>
<td>Add Items to a ListBox from Your Code</td>
<td>134</td>
</tr>
<tr>
<td>Create an Overflow Menu</td>
<td>137</td>
</tr>
<tr>
<td>Creating the Event Handler for the Overflow Button</td>
<td>138</td>
</tr>
<tr>
<td>Add a Search Box</td>
<td>139</td>
</tr>
<tr>
<td>Running Your Application</td>
<td>140</td>
</tr>
<tr>
<td>See Also</td>
<td>141</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mobile Tutorial: Using LiveBindings to Populate a ListBox in Mobile Applications (iOS and Android)</th>
<th>142</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Creating the Project</td>
<td>142</td>
</tr>
<tr>
<td>Step 2: Creating the LiveBindings</td>
<td>144</td>
</tr>
<tr>
<td>The Results</td>
<td>146</td>
</tr>
<tr>
<td>See Also</td>
<td>147</td>
</tr>
</tbody>
</table>

| Mobile Tutorial: Using Layout to Adjust Different Form Sizes or Orientations (iOS and Android) | 148 |
Every FireMonkey Component Can Have an Owner, a Parent, and Children .................................................. 148
Using Common Layout-Related Properties of a FireMonkey Component .......................................................... 149
Using the Align Property .............................................................................................................................. 149
Using the Margins Property .......................................................................................................................... 151
Using the Padding Property ......................................................................................................................... 151
Using the Anchors Property .......................................................................................................................... 151
Using the TLayout Component ..................................................................................................................... 153
Working with a Busy Interface: Using a TVertScrollBox Component ............................................................. 153
See Also ......................................................................................................................................................... 155
Mobile Tutorial: Taking and Sharing a Picture, and Sharing Text (iOS and Android) .............................................. 157
Building the User Interface for the Application ............................................................................................... 161
Taking a Picture with a Mobile Device Camera .............................................................................................. 161
Using a Picture from the Mobile Device Photo Library .................................................................................. 162
Sharing or Printing a Picture .......................................................................................................................... 163
Sharing Text .................................................................................................................................................. 165
See Also ......................................................................................................................................................... 167
Mobile Tutorial: Using Location Sensors (iOS and Android) .............................................................................. 168
Design the User Interface .................................................................................................................................. 169
The Location Sensor ........................................................................................................................................ 170
Read Location Information (Latitude, Longitude) from the LocationSensor Component ................................. 170
Show the Current Location Using Google Maps via a TWebBrowser Component .............................................. 172
Use Reverse Geocoding .................................................................................................................................. 173
Show a Readable Address in the ListBox Component ..................................................................................... 175
See Also ......................................................................................................................................................... 176
Mobile Tutorial: Using Notifications (iOS and Android) .................................................................................... 177
Three Basic Notification or Alert Styles ........................................................................................................ 177
iOS Badge Number and Android Notification Number ..................................................................................... 177
Notification Banner on Mobile Devices ......................................................................................................... 178
Notification Center on Mobile Devices .......................................................................................................... 178
Access the Notification Service ..................................................................................................................... 179
Set the Icon Badge Number and Notification Number from Code ............................................................... 180
Schedule Notification ..................................................................................................................................... 181
Update or Cancel a Scheduled Notification Message ...................................................................................... 182
Present the Notification Message Immediately ............................................................................................. 183
Notification Banner or Notification Alert ....................................................................................................... 184
Add Action to the Notification Alert (iOS Only) .............................................................................................. 185
Add Action to Notifications .......................................................................................................................... 186
See Also ......................................................................................................................................................... 187
Mobile Tutorial: Using Remote Notifications (iOS and Android) ..................................................................... 189
Remote Push Notification ............................................................................................................................... 189
Run Your Application on the mobile platform.......................................................... 317
See Also ...................................................................................................................... 317
Mobile Tutorials: Mobile Application Development (iOS and Android)

This integrated set of tutorials walks you through development of an iOS or Android mobile application for both Delphi and C++:

- 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 FireMonkey Mobile 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 Web Browser Component (iOS and Android)**

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

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

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

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

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

- **Mobile Tutorials: Table of Components Used**
- **FireMonkey Quick Start**
- **Creating an iOS App**
- **Creating an Android App**
Accessing a Database

- 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
  (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:
<table>
<thead>
<tr>
<th>iOS version</th>
<th>Xcode version</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS 6.0</td>
<td>Xcode 4.5</td>
</tr>
<tr>
<td>iOS 6.1</td>
<td>Xcode 4.6</td>
</tr>
<tr>
<td>iOS 7.0</td>
<td>Xcode 5.0</td>
</tr>
<tr>
<td>iOS 7.1</td>
<td>Xcode 5.1</td>
</tr>
</tbody>
</table>

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

- **Platform Assistant (PAServer)**
  For debugging purposes, RAD Studio uses the **Platform Assistant**, an application that you must install and run on the 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

As discussed, 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 **PAServer14.0.pkg** and it is available in two places:

- Inside the RAD Studio installation directory:
  
  C:\Program Files (x86)\Embarcadero\Studio\14.0\PAServer\PAServer14.0.pkg

- On the Web, for download to the Mac:
  

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 14.0.app**) as follows:

1. Navigate to the top-level **Applications** folder.
2. Double-click **PAServer 14.0.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>

```
Platform Assistant Server Version 5.0.2.09
Copyright (c) 2009-2014 Embarcadero Technologies, Inc.

Connection Profile password <press Enter for no password>:

Acquiring permission to support debugging...succeeded

Starting Platform Assistant Server on port 64211

Type ? for available commands
``` 

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

**Step 3: Install Xcode on the Mac**

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

- To run an iOS Application now on the iOS Simulator, see 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

To run your iOS Application on your **iOS Device**, please use the following steps on this page to complete the configuration of your Mac. Note that for Delphi apps only, you can perform these steps after you test an application on the **iOS Simulator**.

### 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**.

---

![Development on Windows](image1.png)

---

**Step 1: Install the Xcode Command Line Tools on a Mac**

To install the necessary Xcode tools using Xcode on the Mac:

1. Start Xcode on the Mac.
2. Choose **Preferences** from the Xcode menu.
3. In the General panel, click **Downloads**.
4. On the Downloads window, choose the **Components** tab.
5. 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 Screenshot](image1.png)

   Save the certificate request as a file, and then send it to your Certificate Authority by uploading it in the Apple iOS provisioning portal.

   - 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: [Creating signing certificates](https://developer.apple.com/documentation/xcode/conceptual/app_development) for details.

2. Go to [iOS Provisioning Portal](https://developer.apple.com/myappleid/). You can download the Development certificate after the status changes from Submitted to Issued:

   ![iOS Provisioning Portal Screenshot](image2.png)

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 Organizer (Window > Organizer).
3. In the Devices tab, click on your device.
4. Next to the Identifier label is a string of characters:

   ![Image showing the Identifier string on an iPad]

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

Step 5: Create and Install a Provisioning Profile

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

- If you are an individual developer, you must create a provisioning profile. For specific information, see: Creating and Downloading a Distribution Provisioning Profile.

- If you are part of a company/organization, your team admins must create a provisioning profile that you can use.

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

1. Open Xcode on the Mac and go to the 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.

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 a FireMonkey Mobile Application (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
- 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)
Additional Steps to Configure Your Mac to Run Your iOS Application on Your iOS Device

- 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**. Only iOS devices are supported by BCCIOSARM.

To deploy an iOS Application to your device for debugging and testing purposes, RAD Studio uses the **Platform Assistant**, which you must **install** and **run** on 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**

The following configuration steps accelerate the iOS development with RAD Studio.

**Create a Connection Profile for the Mac**

1. Open Tools > Options > Environment Options > **Connection Profile Manager**.
2. Select **Add**:
3. Now you see the Create a Connection Profile wizard. Define a name for the connection profile, such as "My Mac". Make sure you select OS X as the platform, and then click Next:
4. On the **Remote machine information** page, set the name or IP address of the host Mac, a port number to use (the default port 64211 typically works), and an optional password (if you want to use a password).

5. Click **Test Connection** and make sure that the connection profile succeeds with no error (you should receive the message "Connection to <hostname> on port <portnumber> succeeded").

6. If the **Test Connection** succeeds click the **Finish** button, otherwise check the settings and try again.
Add an SDK to the Development System for the iOS Device Connected to the Mac

1. Open Tools > Options > Environment Options > SDK Manager:
2. Select **Add**.

3. On the **Add a New SDK** dialog box, select **iOS Device** as a platform.

4. Select a Platform to connect (such as the "iOS Device"): 
5. 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:

![Add a New SDK dialog](image)

Click **OK** to close the dialog.

**See Also**

- Mobile Tutorial: Creating a FireMonkey Mobile Application (iOS and Android)
- 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
  - Provisioning an iOS Device
  - Preparing Your iOS App for Distribution in the App Store
Mobile Tutorial: Set Up Your Development Environment on Windows PC (iOS)

- 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)
Android Development Tools Are Installed with RAD Studio

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\14.0\PlatformSDKs\adt-bundle-windows-x86-20131030\sdk</td>
</tr>
<tr>
<td>Android Native Development Kit (NDK)</td>
<td>C:\Users\Public\Documents\Embarcadero\Studio\14.0\PlatformSDKs\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:
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:
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.

   o 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.

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

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

```
Create a new Android SDK
Setting Android Development Kit (SDK)
Android SDK is required in order to compile, deploy and run Android applications. Please specify the root directories of the SDK and the NDK to continue. The Java Development Kit is also required and should be installed in your system. It will be detected automatically.

Android SDK Base path:

Android NDK Base path:

Java installation path:
C:\Program Files\Java\jdk-1.7.0_25

NDK API Version:
android-14
```

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).

   o 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:

- 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](https://developer.android.com/index.html).

## 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](https://developer.android.com/tools/extras/oem-usb.html) using the Android SDK Manager, as follows:

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

   ![SDK Manager](https://via.placeholder.com/150)

   **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:

   a. 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 a FireMonkey Mobile Application (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 a FireMonkey Mobile Application (iOS and Android)

This topic describes how to create a "Hello World" FireMonkey mobile 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 > FireMonkey Mobile Application - Delphi
   - File > New > FireMonkey Mobile Application - C++Builder

The FireMonkey Mobile Application wizard appears:
2. Select **Blank Application**. The [FireMonkey Mobile Form Designer](#) shows a new form for the default mobile platform, **Android**:
3. 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](#).

4. 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++):
Step 2: Place Components on the FireMonkey Mobile Form

The first step in creating a FireMonkey Mobile application is designing the user interface; this is also the first step when you are targeting desktop platforms. 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 **FireMonkey Mobile Form Designer**.
An instance of the **TEdit** component appears on the form. Here is an Android app:

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 FireMonkey Mobile Form Designer. Here is an iOS app:

![Image of iOS app](image)

7. After you place these components on the FireMonkey Mobile 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 FireMonkey Mobile 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 FireMonkey Mobile 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:
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 FireMonkey Mobile Form Designer and the **Code Editor**:

The **Code Editor** displays the source code that the IDE has generated. You should find three components defined (Edit1, Label1, and Button1):
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 3: 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 FireMonkey mobile application in the same way you define event handlers for desktop applications. For the TButton component, the most typical event is a button click.

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

Delphi

```delphi
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Hello ' + Edit1.Text);
end;
```

C++

```cpp
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MessageBoxA(MAKEINTRESOURCE(IDS_WELCOME), Edit1->Text.c_str(), MB_OK);
}
```

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

```delphi
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Hello ' + Edit1.Text);
end;
```

C++

```cpp
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MessageBoxA(MAKEINTRESOURCE(IDS_WELCOME), Edit1->Text.c_str(), MB_OK);
}
```
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 4: 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**):

![Launch Image Dialog](image)

**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 FireMonkey Mobile Applications

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 mobile application:

- Buttons on the Form:

<table>
<thead>
<tr>
<th>iOS 6</th>
<th>iOS 7</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="iOS 6 Buttons" /></td>
<td><img src="image" alt="iOS 7 Buttons" /></td>
<td><img src="image" alt="Android Buttons" /></td>
</tr>
</tbody>
</table>

- Buttons on the Navigation Bar (also known as Toolbar):
Define the Look and Feel for a Button Component

After you place a new button on the FireMonkey Mobile 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 **Tint** 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.
For the Android target platform, you can apply a tint to buttons of any style.

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 6, the segmented buttons do not support the tint feature, but on iOS 7, these buttons have the TintColor property.

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:

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.

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:

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

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

6. Click and drag these three buttons to select these components:

7. Set the **GroupName** property to a unique name such as LocationSegments:
8. To specify that one of these components is to appear as Pressed by default, set the `IsPressed` property for one component to True:

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="iOS Button" /></td>
<td><img src="image" alt="Android Button" /></td>
</tr>
</tbody>
</table>

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`

<table>
<thead>
<tr>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="iOS Scope Bar" /></td>
<td><img src="image" alt="Android Scope Bar" /></td>
</tr>
</tbody>
</table>

See Also

- [Mobile Tutorial: Creating a FireMonkey Mobile Application (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)]
Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)

- `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 FireMonkey Mobile Applications

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 FireMonkey Mobile 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 Mobile 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(FormatDateTime('dddddd', DateEdit1.Date));
end;
```

**C++Builder:**

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

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

**iOS (iPad)**

![iOS screenshot showing a date picker with a selected date of January 24, 2013]

**Android (LG-E612)**

![Android screenshot showing a date picker with a selected date of January 24, 2013]

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

- FMX.DateTimeCtrls.TDateEdit
Mobile Tutorial: Using Combo Box Components to Pick Items from a List (iOS and Android)

Implementing a Picker in FireMonkey Mobile 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></td>
<td>Arizona</td>
</tr>
<tr>
<td></td>
<td>Arkansas</td>
</tr>
<tr>
<td></td>
<td>California</td>
</tr>
<tr>
<td></td>
<td>Colorado</td>
</tr>
<tr>
<td></td>
<td>Connecticut</td>
</tr>
</tbody>
</table>

To define a picker and the associated list items:

1. Select either of the following:
   - File > New > FireMonkey Mobile Application - Delphi > Blank Application
   - File > New > FireMonkey Mobile Application - C++Builder > Blank Application

2. Select the TComboBox component in the Tool Palette, and drop it on the FireMonkey Mobile 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:
Mobile Tutorial: Using Combo Box Components to Pick Items from a List (iOS and Android)

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
class TForm27
private
  // Code for TForm27
begin
  ComboBox1.Items.Add('Alabama');
  ComboBox1.Items.Add('Alaska');
  ComboBox1.Items.Add('Arizona');
  ComboBox1.Items.Add('Arkansas');
  ComboBox1.Items.Add('California');
  ComboBox1.Items.Add('Virginia');
  ComboBox1.Items.Add('Washington');
  ComboBox1.Items.Add('West Virginia');
  ComboBox1.Items.Add('Wisconsin');
  ComboBox1.Items.Add('Wyoming');
end
```

**C++Builder:**

...
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:**

```c++
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 `FireMonkey Mobile 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:

```
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:

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

Android (LG - E612)

Item California at Index 4 was selected.

iOS6 (iPad)

Item California at Index 4 was selected.

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 the Web Browser Component (iOS and Android)

Using the Web Browser Component in FireMonkey Mobile Applications

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

2. Select the **TToolBar** component in the **Tool Palette**, and drop it on the **FireMonkey Mobile Form Designer**. To find **TToolBar**, enter a few characters (such as “tool”) in the **Search** box of the Tool Palette:

![Tool Palette](image)

3. After you drop the component, you can see the **TToolBar** component at the top of the Mobile Form Designer:

![ToolBar](image)

4. Select the **TButton** component in the Tool Palette and drop it on the **TToolBar**.

5. Select the **TButton** component on the Mobile 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, it is similar to the following image:

   ![Back Button](image)

   For more detail about selecting a style in FireMonkey mobile 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**:

![Edit Control](image)
7. Select the Edit box on the Mobile 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 FireMonkey mobile applications, see [Selecting the Proper Virtual Keyboard for the Web Browser Application](#).

8. Select the **TWebBrowser** component in the Tool Palette and drop it on the form.

9. Select the Web Browser component on the Mobile 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 the desktop platform, mobile devices 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:
Mobile Tutorial: Using the Web Browser Component (iOS and Android)

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:

```c++
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:

```c++
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 Mobile 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:

<table>
<thead>
<tr>
<th>Alphabet:</th>
<th>Default:</th>
<th>EmailAddress:</th>
<th>NamePhonePad:</th>
</tr>
</thead>
<tbody>
<tr>
<td>QWERTYUIOP</td>
<td>QWERTYUIOP</td>
<td>QWERTYUIOP</td>
<td>QWERTYUIOP</td>
</tr>
<tr>
<td>ASDFGHJKL</td>
<td>ASDFGHJKL</td>
<td>ASDFGHJKL</td>
<td>ASDFGHJKL</td>
</tr>
<tr>
<td>ZXCVBNM</td>
<td>ZXCVBNM</td>
<td>ZXCVBNM</td>
<td>ZXCVBNM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NumberPad:</th>
<th>NumbersAndPunctuation:</th>
<th>PhonePad:</th>
<th>URL:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
<td>1234567890</td>
<td>1 2 3 4</td>
<td>QWERTYUIOP</td>
</tr>
<tr>
<td>7 8 9 0</td>
<td>. , ? !</td>
<td>7 8 9</td>
<td>ASDFGHJKL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>ZXCVBNM</td>
</tr>
</tbody>
</table>

Android provides several virtual keyboards as follows:

<table>
<thead>
<tr>
<th>Alphabet:</th>
<th>Default:</th>
<th>EmailAddress:</th>
<th>NamePhonePad:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>QWERTYUIOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ASDFGHJKL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ZXCVBNM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NumberPad:</th>
<th>NumbersAndPunctuation:</th>
<th>PhonePad:</th>
<th>URL:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>QWERTYUIOP</td>
</tr>
<tr>
<td></td>
<td>7 8 9 0</td>
<td>7 8 9 0</td>
<td>ASDFGHJKL</td>
</tr>
<tr>
<td></td>
<td>. /</td>
<td>0</td>
<td>ZXCVBNM</td>
</tr>
</tbody>
</table>

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 Mobile Form Designer, and then in the **Object Inspector**, set the **KeyboardType** property to **URL**.
WebBrowser Mobile Code Snippet

The **WebBrowser** project in [Mobile Code Snippets](#) demonstrates the functionality described in this tutorial.

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

- **Start | Programs | Embarcadero RAD Studio XE6 | Samples** and navigate to `\Object Pascal\Mobile Snippets\WebBrowser`

- **Subversion Repository:**

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

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

Tabs in FireMonkey Mobile Applications

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

For each tab, you can specify:

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

Using the Native Style for Tabs on iOS and Android

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

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

- **On iOS:**
Tabs are typically shown at the bottom of the screen (so you should set `TabPosition` to `Bottom`).

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

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 > FireMonkey Mobile Application - Delphi
   - For C++: File > New > FireMonkey Mobile Application - C++Builder

2. Select `TTabControl` from the Tool Palette:

3. After you drop the `TTabControl`, an empty `TabControl` is shown on the FireMonkey Mobile Form Designer:
4. Typically, applications that use TabControl use the full screen to show pages. To do this, you need to change the default alignment of TabControl. In the
Object Inspector, change the Align property of TabControl to Client:

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

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

7. On the FireMonkey Mobile Form Designer, select the first TabItem and change its StyleLookup property:
Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)

iOS 7

[Image of iOS 7 development environment]

Android

[Image of Android development environment]
8. 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**:

9. 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>
</tbody>
</table>

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

Tabs are displayed at the Bottom.
No Tabs are displayed. Instead, three Dots ([...]) are displayed to indicate additional pages.
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.

### Aligning the TabControl to the Top of the Android Form

For aligning the TabControl to the top on Android, add this code:

#### Delphi:

```delphi
procedure TForm3.FormCreate(Sender: TObject);
begin
  {$IFDEF ANDROID}
 TabControl1.TabPosition := TTabPosition.Top;
  {$ENDIF}
end;
```

#### C++Builder:
void __fastcall TForm7::FormCreate(TObject *Sender)
{
  #if defined(__ANDROID__)
    TabControll->TabPosition = TTabPosition(Top);
  #endif
}

Using Custom Multi-Resolution Icons for Your Tabs

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

Notes:

- In **Android** apps, predefined icons are not supported, so you must use custom icons.
- In **iOS** apps, you can use either predefined icons or custom icons.
- To use custom icons on either iOS or Android, select the appropriate iOS or Android design device in the **Mobile Form Designer**, set the **StyleLookup** property of **TTabItem** to **tabitemcustom**, specify your custom icon as described in this section, and then build your app.
- For iOS, you can use our predefined icons by setting the **StyleLookup** property of **TTabItem** to the icon of your choice, such as **(tabitemsearch)**.
- The custom glyphs used in this section are available in a zip file that is delivered in your **$(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 mobile app (either Android or iOS), 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:

      ![Image of MultiResBitmap Editor with scales added]

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:

![Object Inspector](image)

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

Original Image  Selected Image  Non-Selected Image
Using a Single-Resolution Bitmap for a Custom Icon

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

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

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

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

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

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

Defining Controls within a TabControl

As discussed, each Tab Page can contain any number of controls including another TabControl. In such a case, you can easily browse and manage different tab pages in the Structure View:
Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)

iOS
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 FireMonkey mobile 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:

   **iOS:**

   ![iOS Diagram]

   **Android:**

   ![Android Diagram]
3. 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:**

[Diagram of iOS interface]

**Android:**

[Diagram of Android interface]
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:**

**Android:**
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:**

**Android:**
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:

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

C++Builder:

```cpp
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++Builder:

```
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++Builder:

```
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 FireMonkey Mobile Application for this example. In the wizard, choose Blank Application.

2. In the Tool Palette, locate a ListView 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**.

After placing the required components on the mobile form, the form should look like the following screen on an iOS device:
Step 2: Adding Fields

1. Right-click the TPrototypeBindSource component and select Add Field....
Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

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

Step 3: Creating LiveBindings

1. Open the LiveBindings Designer, if it is not already open, and drag the ColorsName1 property of the TPrototypeBindSource onto the Item.Text property of the list view to bind these properties.
Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)

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

2. In the **Structure View**, locate the **ListView1** component and then click **ItemAppearance** in the hierarchy.

3. In the **Object Inspector**, locate the **ItemAppearance** property and change its value to **ImageListItemRightButton**.
4. Optionally, you can apply a tint to the TListView text buttons. Do the following:
   
   o In the Structure View, locate the ListView1 component and then expand ItemAppearance in the hierarchy.
   
   o Under ItemAppearance, expand Item, and then select TextButton.
   
   o In the Object Inspector, locate the TintColor property and set its value to an appropriate value, such as Seagreen.
At this point, the list view component is configured to display an image on the left-hand side of the item text, and a button on the right-hand side of the item text. The image and button are populated with sample data that you add in the next step.

**Step 4: Adding More Fields**

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 mobile 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: TListItem; const AObject: TListItemSimpleControl);
begin
  ShowMessage(AItem.Text + ' ' + AItem.ButtonText + ' is clicked.');
end;
```

**C++Builder:**

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

**The Results**

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)
• FMX.ListView.TListView
• TPrototypeBindSource
• LiveBindings in RAD Studio
• LiveBindings Designer
• 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 Applications

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>[Select a State]</td>
<td>[Select a State]</td>
</tr>
<tr>
<td><img src="image" alt="iPod" /></td>
<td><img src="image" alt="Android" /></td>
</tr>
<tr>
<td><img src="image" alt="11:13 AM" /></td>
<td><img src="image" alt="17:52" /></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="A" /></td>
<td><img src="image" alt="A" /></td>
</tr>
<tr>
<td>Alabama</td>
<td>Alabama</td>
</tr>
<tr>
<td><img src="image" alt="Alabama" /></td>
<td><img src="image" alt="Alabama" /></td>
</tr>
<tr>
<td>Alaska</td>
<td>Alaska</td>
</tr>
<tr>
<td><img src="image" alt="Alaska" /></td>
<td><img src="image" alt="Alaska" /></td>
</tr>
<tr>
<td>Arizona</td>
<td>Arizona</td>
</tr>
<tr>
<td><img src="image" alt="Arizona" /></td>
<td><img src="image" alt="Arizona" /></td>
</tr>
<tr>
<td>Arkansas</td>
<td>Arkansas</td>
</tr>
<tr>
<td><img src="image" alt="Arkansas" /></td>
<td><img src="image" alt="Arkansas" /></td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>California</td>
<td>California</td>
</tr>
<tr>
<td><img src="image" alt="California" /></td>
<td><img src="image" alt="California" /></td>
</tr>
<tr>
<td>Colorado</td>
<td>Colorado</td>
</tr>
<tr>
<td><img src="image" alt="Colorado" /></td>
<td><img src="image" alt="Colorado" /></td>
</tr>
<tr>
<td>Connecticut</td>
<td>Connecticut</td>
</tr>
<tr>
<td><img src="image" alt="Connecticut" /></td>
<td><img src="image" alt="Connecticut" /></td>
</tr>
</tbody>
</table>
Grouped List

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.
<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alaska</td>
</tr>
<tr>
<td>Arizona</td>
</tr>
<tr>
<td>Arkansas</td>
</tr>
<tr>
<td>California</td>
</tr>
<tr>
<td>Colorado</td>
</tr>
<tr>
<td>Connecticut</td>
</tr>
<tr>
<td>Delaware</td>
</tr>
</tbody>
</table>
This tutorial describes the basic steps to build items for a Table View in your FireMonkey mobile applications.

Create Items on the ListBox Component

1. Select:
   - For Delphi: File > New > FireMonkey Mobile Application - Delphi
   - For C++Builder: File > New > FireMonkey Mobile Application - C++Builder

2. Select the **ListBox** component in the Tool Palette, and drop it on the FireMonkey Mobile Form Designer. To find TListBox, enter a few characters (such as "TListB") in the Search box of the Tool Palette:
3. Select the TListBox component on the Mobile Form Designer, go to the Object Inspector and select Client for the Align property:

4. On the FireMonkey Mobile Form Designer, right-click the TListBox component, and select Items Editor:
5. On the **Items Designer**, click the **Add Item** button several times to add several items to the ListBox:

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

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

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

2. On the Tool Palette, select the **TLabel** component and drop it on top of the **TListBoxHeader** component you just added:
3. In the Object Inspector, change the properties of the TLabel component as follows:

<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:

1. On the FireMonkey Mobile Form Designer, right-click the TListBox component, and select Items Editor.

2. On the Item Designer, select TListBoxGroupHeader from the drop-down list, and then select Add Item:
3. Select `TListBoxGroupFooter` from the drop-down list, and then select **Add Item**.

4. Select `ListBoxGroupHeader1` in the list of items, and click the **Up** button several times until this item becomes the top item on the list:

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

**Show List Items as Separate Grouped Items**

Items on a ListBox can be shown as either a **Plain** list or a **Grouped** list (only for iOS target platform). This choice is controlled by the `GroupingKind` property and the `StyleLookup` property, as shown in the following graphic:
Show Items as Plain List

<table>
<thead>
<tr>
<th>ListBoxGroupName1</th>
<th>ListBoxItem1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListBoxItem2</td>
<td></td>
</tr>
<tr>
<td>ListBoxItem3</td>
<td></td>
</tr>
<tr>
<td>ListBoxItem4</td>
<td></td>
</tr>
</tbody>
</table>

Plain = GroupingKind Property Value

listboxstyle = StyleLookup Property Value

Show Items as Grouped List

<table>
<thead>
<tr>
<th>ListBoxGroupName1</th>
<th>ListBoxItem1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListBoxItem2</td>
<td></td>
</tr>
<tr>
<td>ListBoxItem3</td>
<td></td>
</tr>
<tr>
<td>ListBoxItem4</td>
<td></td>
</tr>
</tbody>
</table>

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

Add an Icon to a ListBox Item

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

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

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:
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 > FireMonkey Mobile Application - Delphi
   - For C++Builder: File > New > FireMonkey Mobile Application - C++Builder

2. Select the TListBox component in the Tool Palette, and drop it on the FireMonkey Mobile Form Designer.

3. Select the TListBox component on the Mobile 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:

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

- C++Builder:

  ```cpp
  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><strong>A</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>aa</td>
<td>aa</td>
</tr>
<tr>
<td>aaa</td>
<td>aaa</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td><strong>B</strong></td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>bb</td>
<td>bb</td>
</tr>
<tr>
<td>bbb</td>
<td>bbb</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td><strong>C</strong></td>
</tr>
<tr>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>cc</td>
<td>cc</td>
</tr>
<tr>
<td>ccc</td>
<td>ccc</td>
</tr>
</tbody>
</table>

**Delphi:**

```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);
end;
```
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++Builder:

void __fastcall TForm1::FormCreate(TObject *Sender)
{  
    char c;
    int i;
    String Buffer ;
    TListBoxItem *ListBoxItem ;
    TListBoxGroupHeader *ListBoxGroupHeader ;

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

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

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

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

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

1. Add a TToolBar component on the form and set the alignment to Top.
2. Place three TSpeedButton components on the TToolBar component:
   - For the first TSpeedButton:
     - Set the Align property to Left.
     - Change the Name property to OrganizeButton.
     - Set the StyleLookup to organizetoolbutton.
   - For the second TSpeedButton:
     - Set the Align property to Right.
     - 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.
Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

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

4. For the first four **TLBoxItem** 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 **TLBoxItem** 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
```
Mobile Tutorial: Using ListBox Components to Display a Table View (iOS and Android)

- **C++Builder:**

```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:

![Search Box in TListBox Context Menu](image)

- 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;
end;
```
end;

C++Builder:

```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.
Android (Samsung Tab 2.0)

Displaying the overflow menu

Displaying the search box

See Also

- FMX.ListBox.TListBox
- Mobile Tutorial: Using LiveBindings to Populate a ListView (iOS and Android)
- Mobile Tutorial: Using LiveBindings to Populate a ListBox in Mobile Applications (iOS and Android)
- Mobile Tutorial: Using a Button Component with Different Styles (iOS and Android)
- Mobile Tutorial: Using the Web Browser Component (iOS and Android)
- Mobile Tutorial: Using Tab Components to Display Pages (iOS and Android)
Mobile Tutorial: Using LiveBindings to Populate a 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 > FireMonkey Mobile Application - Delphi
   - For C++: File > New > FireMonkey Mobile Application - C++Builder

2. Select the TListBox component in the Tool Palette, and drop it on the FireMonkey Mobile Form Designer.

3. Select the list box component on the Mobile 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 FireMonkey Mobile Form Designer.

5. Select the TRectangle component in the Tool Palette, and drop it on the FireMonkey Mobile Form Designer.

6. Select the rectangle component on the Mobile 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).

![Add Field](image)

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

### Step 2: Creating the LiveBindings

Open the **LiveBindings Designer**. The diagram with no connections is similar to the following image:
1. Bind **ColorsName1** from **PrototypeBindSource1** to **Item.Text** from **ListBox1**. (Click **ColorsName1** and drag the mouse to **Item.Text**.) Now the list box displays all the color names in the prototyping binding source.

2. Bind **ColorsName1** from **PrototypeBindSource1** to **ItemHeader.Text** from **ListBox1**. Now the list box also displays headers with all the color names in the prototyping binding source.

3. In the **LiveBindings Designer**, click the binding that connects **ColorsName1** to **ItemHeader.Text**.

4. In the **Object Inspector**, locate the **FillHeaderCustomFormat** property and select the **SubString(%s, 0, 1)** binding expression from the drop-down list.

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

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

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

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

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

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

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

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

**The Results**

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

When you select a list box item, the rectangle’s color changes accordingly:
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 one common form for different form factors (such as phone or tablet), without using different forms for each form factor.

In the FireMonkey Mobile Form Designer, you can preview the user interface without running the application on a device — just change the device or orientation in the dropdown at the upper right corner:

For more information, see Mobile Device Settings.

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

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

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

Margins ensure separation between controls automatically positioned by a parent.

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

Using the Padding Property

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

- Left
- Right
- Bottom
- Top

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

Using the Anchors Property

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

**Anchors Property for the Edit Control**

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

If you want the *Edit* control to maintain the same relative position in relation to the *ToolBar* (its parent), 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](image1)

**Android**

![Android Edit Control](image2)

**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](image3)

**Android:**

![Android Button Control](image4)
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.

Working with a Busy Interface: Using a TVertScrollBox Component

In general, you do not want your form to have too many items, which can force users to scroll the user interface. In many cases, you can use a TabControl component with several pages to avoid any scrolling.

If you need to place many items on your form, you can use a TVertScrollBox component to define a scrolling area as described here:

1. Select Custom Device on FireMonkey Mobile Form Designer.
2. Change the size of the Custom Device by dragging the edge of the designer to the shape and size you want.

3. Drop a **TVertScrollBox** component, and set its **Align** property to **alClient**. This causes the TVertScrollBox to fill the client area.

4. Locate components on the TVertScrollBox component:
You can scroll this form at run time as you swipe the user interface.

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

- 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 in your mobile applications.

**On iOS Devices:**

Taking a picture with the device camera  
Using a picture from the device Photo Library
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:

- **TToolBar** 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 `actiontoolbuttonbordered`, 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++Builder:**

```cpp
void __fastcall TForm2::TakePhotoFromCameraAction1DidFinishTaking(TBitmap *Image)
{
  Image1->Bitmap->Assign(Image);
}
```

This code assigns a picture taken from the mobile device camera to the `TBitmap` 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++Builder:**

```cpp
void __fastcall TForm2::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.

### Sharing or Printing a Picture

From your mobile 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++Builder:**

```cpp
void __fastcall TForm2::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 (Delphi).

**Sharing Text**

From your mobile 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 mobile 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 FireMonkey mobile application.
2. Place the following components on the **Form Designer**:
   - **ToolBar** 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:
```
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, `LocationSensor` 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.

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

- 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 property. If you set Distance to "10", TLocationSensor fires an OnLocationChanged event when you move "10 meters".

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

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

C++Builder:

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

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

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

Delphi:

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

C++Builder:

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

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

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

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

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

**Delphi:**

```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++Builder:**

```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(L"https://maps.google.com/maps?q=%2.6f,%2.6f",
    NewLocation.Latitude, NewLocation.Longitude);

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

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

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

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

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

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

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

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

**Delphi:**

```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++Builder:**

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

```cpp
class TForm1 : public TForm
```

Embarcadero Technologies
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
  // 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);
end;
```

**C++Builder:**

```cpp
void __fastcall TForm1::LocationSensor1LocationChanged(TObject *Sender, const TLocationCoord2D &OldLocation, const TLocationCoord2D &NewLocation)
{
  // code for previous steps goes here
```
// Setup an instance of TGeocoder
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);
}

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;
```

**C++Builder:**

```cpp
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;
    }

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

```csharp
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

![iOS Badge Number](image)

iPad

Android Notification Number

![Android Notification Number](image)

Android
Notification Banner on Mobile Devices

**iOS**

![Notification Banner on iOS](image-url)

**Android**

![Notification Banner on Android](image-url)

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**

![Notification Center on iOS](image-url)
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:

- For Delphi applications, add the following units to the uses clause if they are not present:

```plaintext
uses
  FMX.Platform, FMX.Notification;
```

- For C++ applications, add the following `include` operators to the project header file:

```plaintext
#include <FMX.Platform.hpp>
#include <FMX.Notification.hpp>
```

- Select the `TNotificationCenter` component in the Tool Palette, and drop it on the FireMonkey Mobile Form Designer.

The `FMX.Notification.TNotificationCenter.CreateNotification` method allows you to create an instance of the `TNotification` class object.
Set the Icon Badge Number and Notification Number from Code

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

To set the iOS icon badge number (for Delphi or C++) or the Android notification number (only for Delphi apps), you 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();
    }
  }
}
```
Note: In your source code, you should call the sample methods that are presented in this tutorial from any appropriate event handler, such as the OnClick event handler of a TButton component.

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):

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

Schedule Notification

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

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

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 message](image_url)

**Update or Cancel a Scheduled Notification Message**

Each Scheduled Notification Message is identified through the `Name` property of the `TNotification` object.

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
delete;
```
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 notification 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](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:
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
```
C++:

```cpp
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 FireMonkey Mobile 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 TForm1 NotificationCenter1ReceiveLocalNotification(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:

- **iOS:** Apple Push Notification (APN)
  - Two Providers:
    - Parse
    - Kinvey

- **Android:** Google Cloud Messaging (GCM)
  - One Provider: Kinvey

In order to receive push notifications, you need to set up the messaging service (APS or GCM), the device, the cloud service (Kinvey or Parse), 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
Mobile Tutorial: Using Remote Notifications (iOS and Android)

- 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)
Setting Up the Messaging Service

Remote Notifications require:

- A new project on Google Developer Console for Android apps
- A new App-ID on Apple Developer Program for iOS apps

Use the following steps to set up the messaging service for Android and iOS apps.

Registering with Google

1. Go to Google Developer Console
2. Select Create Project and define a name for your project:

3. On the project page, you need both Project ID and Project Number later in this procedure:

4. Select APIs & auth from the left-hand side, and enable Google Cloud Messaging API for Android:
5. Select **Credentials** from Public API access section, press the **CREATE NEW KEY** button, and select **SERVER KEY**. For testing purposes, you can set the SERVER KEY to `0.0.0.0/0`.

6. Record the API key, which is needed for the next step:

### Registering with Apple

#### Creating iOS App ID on Apple Developer Program

1. Go to [iOS Developer Program > Certificates, Identifiers & Profile > Identifiers > APP ID page](#).

2. Press the + button at top right to define a **new App ID**. Fill **App Name**, **App ID Prefix**, **App ID Suffix**. Also, select **Push Notifications** in the **App Services** section:
Registering an App ID

The App ID string contains two parts separated by a period (.)—an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an AppID has different and important uses for your app.

App ID Description

Name:

You cannot use special characters such as @, $, "", ", ".

App ID Prefix

Value:

App ID Suffix

Explicit App ID

If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit AppID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

Bundle ID: com.embarcadero...com.embarcadero.

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

Wildcard App ID

This allows you to use a single App ID to match multiple apps. To create a wildcard App ID, enter an asterisk (*) as the last digit in the Bundle ID field.

Bundle ID:

Example: com.domainname.*

App Services

Select the services you would like to enable in your app. You can edit your choices after this AppID has been registered.

Enable Services:

- Data Protection
  - Complete Protection
  - Protected Unless Open
  - Protected Until First User Authentication
- Game Center
- iCloud
- In-App Purchase
- Inter-App Audio
- Passbook

- Push Notifications
3. Click **Continue**, and then press **Submit** to confirm your App ID:

![Registration complete](image)

Now your app is registered.

**Creating a Certificate for your App**

1. On **Certificates, Identifiers & Profile** Apple page select your app and click **Edit**:
2. Select Create Certificate in the Development SSL Certificate section:

3. In the About Creating a Certificate Signing Request (CSR) page, select Continue.

4. On the Mac, select Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority...
5. On the Certificate Assistant page:
   - Enter your e-mail address and your name.
   - Select the Save to disk option.
   - Press Continue:

6. Specify a file name and save. Now you have a CSR saved as a file.

7. On the Apple Developer Program web page, specify a CSR:
8. Press **Generate**, and you will get a Certificate. Download the certificate, and install it on your system.
9. Now you have a certificate installed on your system:
Creating a Provisioning Profile

1. On the **iOS Provisioning Profiles** page, select + at the top right:

2. On the **Add iOS Provisioning Profile** page:
   - Select **iOS App Development** as type of the profile. Press **Continue**:
o Select the App ID that you created in the previous step:

3. Select the list of people who can use this profile, and click Continue.

4. Select the list of devices for this profile, and click Continue.

5. Specify a name for the profile, and click Generate:
6. Download your profile:
7. Double-click the profile, and you should now see it as a new profile in Xcode on your Mac:

![Xcode screenshot showing a new profile]

Exporting the .p12 Certificate

1. In Keychain Access on the Mac, select the certificate, right-click it and choose Export.

2. Define a name and click **Save**.

3. You can only choose a password if you use Kinvey as Cloud Service.
Setting Up the Messaging Service

Next

- Using the Cloud Service
- Creating the Application

See Also

- Mobile Tutorial: Using Notifications (iOS and Android)
Using the Cloud Service

There are multiple providers on the market that can be used for sending notifications. RAD Studio comes with pre-installed components for two providers:

- **Kinvey** (for iOS and Android)
- **Parse** (for iOS only).

These providers usually offer more services than just sending out remote-notifications.

**Note:** You must have an account in order to use either one of the providers.

Create a Project at Kinvey

**Setting up your Project**

1. Go to the [Kinvey console](https://console.kinvey.com), and create a new project from the top right side. (Click [Sign up for Kinvey](https://www.kinvey.com).)

2. To create your app backend:
   - Set a name for the project.
   - For platform, select **REST API** from the **Other** tab.
3. Now you can see the **AppKey**, **App Secret** and **Master Secret** parameters that you will need later.

**NOTE:** You can also go to the **app settings page** to copy **App Secret** and **Master Secret**.
Push Configuration

1. Go to Addons > Messaging > Push.

Google Cloud Messaging Configuration

1. Copy:
   - the Project ID from Registering at Google, step 3.
   - the API Key from Registering at Google, step 7.

2. Click Save GCM Configuration.

iOS Configuration

1. Upload your certificate from this step to Kinvey.
2. Click Save iOS Configuration.

Adding a User

1. Go to Addons > Core > Users.
2. Select **Add User** and define a user and a password.

![Add User Form](image)

3. Click the **Create User** button.

**Create a Project at Parse**

1. Go to the [Parse page](#), click either **Sign Up** or **Try it for free**, and create a new app.

![Parse App Creation](image)

2. Specify a name for your app.
3. A list of Keys is displayed. Click **Data Browser** to see configuration details.

**NOTE:** You can access this list later by browsing to **Settings > Application keys**.

4. Go to **Settings > Push Notification**.
5. In **Apple Push Certificate** section upload your certificate from this [step](#). The Apple App ID and the Parse App are now linked.

**Previous**
- [Setting Up the Messaging Service](#)

**Next**
- [Creating the Application](#)

**See Also**
- [Mobile Tutorial: Using Notifications (iOS and Android)](#)
Creating the Application

In this step, you create the basic application with elements that are generic to Android and iOS, and you add the code that makes the characteristics for the specific platforms.

Note: Because our implementation is based on the REST BAAS framework, RAD Studio allows you to use:

- Either provider (Kinvey or Parse) with iOS apps
- Only Kinvey with Android apps

Design and Set Up the User Interface

1. To create an **HD FireMonkey Mobile Application**, select either of the following:
   - File > New > FireMonkey Mobile Application - Delphi > Blank Application
   - File > New > FireMonkey Mobile Application - C++Builder > Blank Application

2. Drop a **TCheckBox** component on the form.

   In the **Object Inspector**, set the following properties of the **CheckBox**:
   
   1. Set the **Align** to **Top**.
   2. Set the **IsChecked** property to **False**.
   3. Set the **Text** to **Active**.

3. Add a **TMemo** component to the form and set the **Align** property to **Client**.

4. Drop a **TPushEvents** object.

5. Set the **AutoActivate** to **False**.

6. In LiveBindings Designer add a link from **Active** in **PushEvents** to **IsChecked** in **CheckBox**. The **Active** property of **PushEvents** is set to **True** when the **CheckBox** component is checked.
7. Depending on the Cloud Service you want to use, add a `TParseProvider` for Parse or `TKinveyProvider` for Kinvey.

**Note:** This tutorial uses both of these cloud services as follows:
- Kinvey for Android devices
- Parse for iOS devices

3. **Using `TKinveyProvider` (iOS or Android)**

In the **Object Inspector**, set the following properties of the `KinveyProvider`:

- Expand the **Android Push** node and copy to **GCMAppID** the **Project Number** from [step 2](#).
- Set the **AppKey**, **AppSecret** and **MasterSecret** to API Keys values from **Setting up your Project** - [step 4](#).
- Set **UserName** and **Password** to the values defined at **Adding an User**.
4. Using **TParseProvider** (iOS only)

In the **Object Inspector**, set the following properties of the **ParseProvider**:

- Set the **ApplicationID**, **MasterKey** and **RestApikey** to API Keys from step 3 in **Creating a Project at Parse**.

---

Creating the Event Handlers

1. On the form, select **PushEvent1** and go to the **Object Inspector**:

   - Check whether the **Provider** property is set to **KinveyProvider1** or to **ParseProvider1**, depending on which provider you used.

   - Go to **Events** tab and create an event handler for each event by double-clicking the Value field.
2. Switch to **Code** tab by pressing F12.

3. Define the event handlers as follows:

**Delphi:**

```plaintext
implementation

{$R *.fmx}

procedure TForm1.PushEvents1DeviceRegistered(Sender: TObject);
begin
  Memo1.Lines.Add('Device Registered');
  Memo1.Lines.Add('');
end;

procedure TForm1.PushEvents1DeviceTokenReceived(Sender: TObject);
begin
  Memo1.Lines.Add('Device Token Received');
  Memo1.Lines.Add('');
end;

procedure TForm1.PushEvents1DeviceTokenRequestFailed(Sender: TObject;
  const AErrorMessage: string);
begin
  Memo1.Lines.Add('Device Token Request Failed');
  Memo1.Lines.Add(AErrorMessage);
  Memo1.Lines.Add('');
end;

procedure TForm1.PushEvents1PushReceived(Sender: TObject;
  const AData: TPushData);
begin
  Memo1.Lines.Add('Device push received');
  Memo1.Lines.Add(AData.Message);
  Memo1.Lines.Add('');
end;
end.
```

**C++:**

```plaintext
//-----------------------------------------------
```
void __fastcall TForm1::PushEvents1DeviceRegistered(TObject *Sender)
{
    Memo1->Lines->Add("Device Registered");
    Memo1->Lines->Add("");
}  //----------------------------------------------------------------------

void __fastcall TForm1::PushEvents1DeviceTokenReceived(TObject *Sender)
{
    Memo1->Lines->Add("Device Token Received");
    Memo1->Lines->Add("");
}  //----------------------------------------------------------------------

void __fastcall TForm1::PushEvents1DeviceTokenRequestFailed(TObject *Sender, const UnicodeString AErrorMessage)
{
    Memo1->Lines->Add("Device Token Request Failed");
    Memo1->Lines->Add(AErrorMessage);
    Memo1->Lines->Add(""棚);
}  //----------------------------------------------------------------------

void __fastcall TForm1::PushEvents1PushReceived(TObject *Sender, const TPushData *AData)
{
    Memo1->Lines->Add("Push Received");
    Memo1->Lines->Add(AData->Message);
    Memo1->Lines->Add("");
}  //----------------------------------------------------------------------

Android Settings

**Note:** To verify whether your Android device supports GCM, see [GCM Overview](#).

**Editing AndroidManifest.template.xml**

To enable GCM support in an Android application you will need to include some additional entries in the `AndroidManifest.xml` for the project. When you build your project, RAD Studio uses `AndroidManifest.template.xml` as a template to generate `AndroidManifest.xml` in the output directory.

**Adding the Permissions**

Some of the necessary permissions can be enabled in the project options, but it is easier just to copy and insert the whole block including all permissions needed for GCM. In the `AndroidManifest.template.xml` file search for a template variable:

```xml
<%uses-permission%>
```

Insert the following lines directly below:

```xml
<!-- GCM connects to Google Services. -->
<uses-permission android:name="android.permission.INTERNET" />
```
Creating the Application

<!-- GCM requires a Google account. -->
<uses-permission android:name="android.permission.GET_ACCOUNTS" />

<!-- Keeps the processor from sleeping when a message is received. -->
<uses-permission android:name="android.permission.WAKE_LOCK" />

<!-- Creates a custom permission so only this app can receive its messages.

NOTE: the permission *must* be called PACKAGE.permission.C2D_MESSAGE,
where PACKAGE is the application's package name.
-->
<permission android:name="%package%.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
<uses-permission android:name="%package%.permission.C2D_MESSAGE" />

<!-- This app has permission to register and receive data message. -->
<uses-permission
android:name="com.google.android.c2dm.permission.RECEIVE" />

Registering the Receiver Class

Search for this block:

</intent-filter>
</activity>

Add the following code:

<receiver
    android:name="com.embarcadero.gcm.notifications.GCMNotification"
    android:exported="true"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="%package%" />
    </intent-filter>
</receiver>

Registering the Service Class

This is a Java class that will create an entry in the Notification Center of the Android Device - even if the application itself is not running while the remote notification comes in. If you don’t need or want items in the Notification Center, you can skip this step.

Add the following code below the previous block from Registering the Receiver Class:

<service android:name="com.embarcadero.gcm.notifications.GCMIntentService" />
iOS Settings

OpenSSL

The REST API of Kinvey and Parse are accessed through secure http (HTTPS). Your application will need OpenSSL support in order to make HTTPS requests. On platforms that do not include OpenSSL libraries, such as iOS, you will need to download OpenSSL libraries.

For iOS devices you need to download the `libcrypto.a` and `libssl.a` static libraries.

The location of downloaded libraries is important. For static libraries, the files need to be in the linker library path or in the project directory.

**Project Settings**

1. Right-click your project in the Project Manager. Choose **Project > Options > Entitlement List**, and select Receive Push Notification in iOS.

2. Choose **Project > Options > Version Info**, and set the `CFBundleIdentifier` key. This should be the same as the identifier of your App ID. It is the **Bundle ID** from step 2 in this tutorial ([Creating iOS App ID on Apple Developer Program section](#)).
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.
2. Choose either of the following commands:
   - Run > Run
   - Run > Run Without Debugging
3. Click the Active checkbox.
4. Go to Parse or Kinvey and send a new push:
Creating the Application

Parse

Switch to your mobile device:

Kinvey

5. Switch to your mobile device:
6. Send your app to background and send another push from Parse or Kinvey. Now go to Notifications:

iOS

Android

Push Received
Hello from Kinvey to Android device!
Creating the Application

Android

Previous

- Setting Up the Messaging Service
- Using the Cloud Service

See Also

- Mobile Tutorial: Using Notifications (iOS and Android)
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 > FireMonkey Mobile Application - Delphi
   - For C++: File > New > FireMonkey Mobile 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++Builder:**

     ```cpp
     #include <FMX.Platform.hpp>
     #include <FMX.PhoneDialer.hpp>
     ```
Mobile Tutorial: Using the Phone Dialer on Mobile Devices (iOS and Android)

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

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

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

**Delphi:**

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

**C++Builder:**

```
private: // User declarations
_DI_IFMXPhoneDialerService phoneDialerService;
bool serviceSupported;
```

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

```
constructor TForm1.Create(AOwner: TComponent);
begin
inherited Create(AOwner);
TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService,
IInterface(PhoneDialerService));
end;
```

- Only for C++Builder apps: in the Structure View, click the form, and in the Object Inspector, open the Events tab, and then double-click onCreate. Implement the following onCreate event handler for the application form:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    serviceSupported = (TPlatformServices::Current->
>SupportsPlatformService(__uuidof(IFMXPhoneDialerService)) &&
(phoneDialerService = TPlatformServices::Current->
>GetPlatformService(__uuidof(IFMXPhoneDialerService))));
}
```

Now your application can access the phone dialer services.
Designing the User Interface

This tutorial uses the TLabel, TButton, and TEdit components as the user interface elements.

**To set up the UI elements, perform the following steps:**

1. Drop two TLabel components on the Form Designer, and then set their Name properties to lblCarrierName and lblISOCountryCode, respectively.
2. Set the Text property for the labels to Carrier Name and ISO Country Code, respectively.
3. Drop a TButton component on the Form Designer, and in the Object Inspector, set the following properties of this button:
   - Name to btnGetCarrierInfo
   - Text to Get Carrier Info

Getting the Carrier Properties

To get information on the carrier, make the following changes:

1. On the Form Designer, select the Get Carrier Info button.
2. In the Object Inspector, double-click the onClick event, and implement the onClick event handler as follows:

**Delphi:**

```delphi
procedure TForm1.btnGetCarrierInfoClick(Sender: TObject);
begin
  if Assigned(PhoneDialerService) then
  begin
    lblCarrierName.Text := 'Carrier Name: ' + PhoneDialerService.GetCarrier.GetCarrierName;
  end;
end;
```

**C++Builder:**

```cpp
void __fastcall TForm1::btnGetCarrierInfoClick(TObject *Sender)
{
  if (serviceSupported) {
    lblCarrierName->Text = "Carrier Name: ") + phoneDialerService->GetCarrier().GetCarrierName();
  }
}
```
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:
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:
Mobile Tutorial: Using the Phone Dialer on Mobile Devices (iOS and Android)

- 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;
```

**C++Builder:**

```cpp
void __fastcall TForm1::btnMakeCallClick(TObject *Sender)
{
  if (serviceSupported) {
    if (edtTelephoneNumber->Text != "") {
      phoneDialerService->Call(edtTelephoneNumber->Text);
    } else {
      ShowMessage("Please type-in a telephone number.");
      edtTelephoneNumber->SetFocus();
    }
  } else ShowMessage("This device does not support the Phone Dialer
  services.");
}
```

**To make a call:**

1. Run the application.
2. In the TEdit field under **Telephone Number**, type the phone number.
3. Click the **Make Call** button.
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 "+").
Mobile Tutorial: Using the Phone Dialer on Mobile Devices (iOS and Android)

<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++Builder:**
   ```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++Builder:**
   ```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:

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++Builder:

void __fastcall TForm1::MyOnCallStateChanged(const UnicodeString aCallID, const TCallState aCallState) {
    switch (aCallState) {
    case TCallState::None:
        lblCallState->Text = "No calls";
        break;
    case TCallState::Connected:
        lblCallState->Text = "Connected";
        break;
    case TCallState::Incoming:
        lblCallState->Text = "Incoming call";
        break;
    case TCallState::Dialing:
        lblCallState->Text = "Dialing";
        break;
    case TCallState::Disconnected:
        lblCallState->Text = "Disconnected";
        break;
    }
}
After you implement this event handler, the application shows the phone call state. For example, the following iOS screen indicates that the phone is in a dialing state:

Note: In this sample project, the TLabel component is next to the TEdit box and the Make Call button, under Telephone Number.

See Also

- Mobile Tutorial: Taking and Sharing a Picture, and Sharing Text (iOS and Android)
- Mobile Tutorial: Using Location Sensors (iOS and Android)
- FMX.PhoneDialer.IFMXPhoneDialerService
- Android Mobile Application Development
• iOS Mobile Application Development
• Mobile Code Snippets: Phone Dialer
Mobile Tutorial: Using FireDAC in Mobile Applications (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 multi-device 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.
The FireDAC native driver supports SQLite database version 3.0 and later. For a detailed discussion on SQLite usage in FireDAC for a Delphi application, read the "Using SQLite with FireDAC" article.

To use SQLite in your application, the **sqlite3.dll** file must be 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).

**Design and Set Up the User Interface**

Create a new project. Choose a FireMonkey Mobile Application.

1. Drop a **TFDConnection** component on the form.
2. Right click the **TFDConnection** component and choose **Connection Editor**.
3. In the FireDAC Connection Editor, set the following parameters of the **TFDConnection**:
   1. Set the **Driver ID** property to **SQLite**.
   2. Set the **Database** parameter to:
      C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\Employees.s3db (location of the database)

      and click **Open** in the File Open dialog box.

      (To display Employees.s3db in the Open dialog, set the All Files (*.*) option.)

3. Click the **Test** button to test the connection.
4. 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**.

**Using the LiveBindings Wizard**

Use the **LiveBindings Wizard** to add the LiveBindings components (**TBindSourceDB**, **TBindingsList**, **TFDQuery**, and the **TListView** 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 Name, Department, Seniority from Employee order by Department`.

At this point, **TBindSourceDB** and **TFDQuery** components were added to your form.

Add the **ListView component**

1. Reopen the **LiveBindings Wizard**.
2. Select **Link a control with a field** binding task.

9. Click the **Test Command** button.
10. Click the **Next** button.
11. Click the **Finish** button.
3. Click the **Next** button.
4. Select **TListView**.
5. Click the **Next** button.
6. Select **BindSourceDB1**.
7. Click the **Next** button.
8. Select **Name** Field Name.
9. Click the **Next** button.

10. Click the **Finish** button to close the wizard.

**Using the LiveBindings Wizard**

LiveBindings Designer
Use the LiveBindings Designer to add new connections between the TBindSourceDB and TListView components.

1. Select the ItemHeader.Text member from TListView.
2. Bind to the Department member of the BindSourceDB1 component by dragging (a connection line appears).
3. Select the Item.Text member from TListView.
4. Bind to the Name member of the BindSourceDB1 component by dragging (a connection line appears).
5. In the Object Inspector, set the following properties of TListView:
   1. Set the ItemAppearance to ImageListItemRightButton.

   In the LiveBindings Designer, the TListView should have a new member, Item.ButtonText.
   2. Set the Align property to Client.

6. In the LiveBindings Designer, select the Item.ButtonText member from TListView.
7. Bind to the Seniority member of the BindSourceDB1 component by dragging (a connection line appears).

Preparing Your Application for Run Time

FireDAC has a loosely-coupled multilayered architecture, where layers provide services. A service API is defined as a COM interface that other layers can request using the interface factory.

To properly operate FireDAC, you must link the implementation of the IFDGUIxWaitCursor and IFDPhysDriver interfaces to your application.

For this, drop the TFDGUIxWaitCursor and TFDPhysSQLiteDriverLink components on the form.

Setting Up Your Database Deployment for mobile

Up to this point, you have used SQLite on your desktop. This means that the actual database is located on your local hard disk drive (for example, C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\Employees.s3.db). 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.
Mobile Tutorial: Using FireDAC in Mobile Applications (iOS and Android)

- Change 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 (Employees.s3db).

1. Open the **Deployment Manager** by selecting Project > Deployment.
2. Select **Add Files** and select the database file (for example, \C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\Employees.s3db).
3. Select **Employees.s3db**, and change **Remote Path** to **StartUp\Documents** (for iOS platform) or **assets\internal** (for Android platform).
   
   **RemotePath on iOS device platform**
   
   **RemotePath on Android platform**

4. Select the **Platforms** column (double-click the ellipsis [...] in the row for Employees.s3db):
   
   1. Ensure that **iOS Simulator** and **iOS Device** or **Android** are present for Employees.s3db.
   2. Remove Win32 from the list if it is present (you do not have to copy database files to the Win32 platform).
5. Select **All-Configurations - iOS Device** platform or **All-Configurations - Android** platform and make sure **Employees.s3db** is set to be deployed to **StartUp\Documents** or **assets\internal**.

As you just configured, when you run the app on the mobile device, the database file (Employees.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 mobile app.
Modifying Your Code to Connect to a Local Database File on Mobile

The basic features of this application are now implemented. The database file used in this application was created on Windows and the file is not available on your mobile device unless you copy it to the mobile device or create it on the fly.

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
   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++ (only for iOS):
   
   ```cpp
   void __fastcall TForm1::FDConnection1BeforeConnect(TObject *Sender)
   {
     FDConnection1->Params->Values["ColumnMetadataSupported"] = "False";
     FDConnection1->Params->Values["Database"] = System::IoUtils::TPath::Combine(System::IoUtils::TPath::GetDocumentsPath(), "Employees.s3db");
   }
   ```

   The TPath record is declared in System.IOUtils library, so you need to add #include <System.IOUtils.hpp> in your header unit.

Running Your Application on the Simulator or on the Mobile Device

Now your application is ready to run (select Run > Run).
See Also

- [Connect to SQLite (FireDAC)]
- [Preparing a FireDAC Application for Run Time]
- [LiveBindings in RAD Studio]
- [Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)]
- [Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)]
- [Tutorial: Using FireDAC in a HD FireMonkey Application]
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 XE5 Professional or higher
  - Delphi XE5 Professional with Mobile or higher
- 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 `PublicInterBaseXE3Path`

**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.
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 FireMonkey Mobile Application, select either of the following:
   - File > New > FireMonkey Mobile Application - Delphi > Blank Application
   - File > New > FireMonkey Mobile 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 listboxitemlabel.
   - Set the HorzAlign property in TextSettings to Center.
   - Set the Text property to DB DEMO.

Connecting to the Data

Following are the basic steps to connect to data in a database using FireDAC:

1. On the Tool Palette, double-click the TFDConnection component.
2. Right-click the **TFDConnection** component and choose **Connection Editor**.

3. In the FireDAC Connection Editor, set the following parameters of the **TFDConnection**:
   
   1. Set the **Driver ID** property to **IB**.
   2. Set the **Database** parameter to:
      
      ```plaintext
      C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\dbdemos.db
      (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. Click the **Test** button to test the connection.

6. 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 `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 TFDGUxWaitCursor 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\14.0\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.
- Change 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. Open the Deployment Manager by selecting Project > Deployment.
2. Select All-Configurations - iOS Device platform or All-Configurations - Android platform from the drop-down list of target platforms at the top of the Deployment Manager.
3. Select Add Featured Files: 

   ![Add Featured Files](image)

4. 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_nnnnnn.txt`, where `nnnnnn` 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\redis\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.

5. Select Add Files and select the database file (for example, `C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\dbdemos.gdb`).
6. Select `dbdemos.gdb` and change Remote Path to `StartUp\Documents` (for iOS platform) or `assets\internal` (for Android platform).

**RemotePath on iOS device platform**

**RemotePath on Android platform**

7. Select the Platforms column (double-click the ellipsis [...] in the row for `dbdemos.gdb`):

   1. Ensure that iOS Simulator and iOS Device or Android are present for `dbdemos.gdb`.

   2. Remove Win32 from the list if it is present (you do not have to copy database files to the Win32 platform).

Select All-Configurations - iOS Device platform or All-Configurations - Android platform, and make sure `dbdemos.gdb` is set to be deployed to `StartUp\Documents` or `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 mobile app.

**Delphi:**

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

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

**C++:**
void __fastcall TForm1::SQLConnection1BeforeConnect(TObject *Sender)
{
#if ((defined(__arm__) && defined(__APPLE__)) || defined(__ANDROID__))
    FDConnection1->Params->Values["Database"] =
        System::IoUtils::Path::Combine(System::IoUtils::Path::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.
Troubleshooting

InterBase License Issues

If you get an error ("unavaiable database") when you connect to the database in the development environment, this means you forgot to deploy the Interbase ToGo license.

- To execute your application on mobile, deploy the `reg_ibtogo.txt` or `reg_iblite.txt` license files.
- The license files are located in `PublicInterBaseXE3Path` directory.

Exception Handling Issues

If your application raises an exception without having proper exception handling code, your mobile app 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++:**

```c++
void __fastcall TForm1::Button1Click(TObject *Sender) {
  try {
    FDConnection1->Connected = true;
    FDQuery1->Active = true;
  } catch(Exception &e) {
    ShowMessage(e.Message);
  }
}
```

**Typical Errors and Resolutions**

Following are typical errors that you might encounter when you connect to the database, and suggestions for resolving the issues:
Error on mobile

I/O error for file */private/var/mobile/Applications/9379975A-EE1A-41C2-A1C8-860FBE07DF96/Documents/dbdemos.gdb*
   Error while trying to open file
   No such file or directory

OK

Suggestion

Check whether the database file (dbdemos.gdb) is delivered to ‘StartUp\Documents\’ (for iOS) or ‘assets\internal\’ (for Android).

Check whether the license file is delivered for InterBase ToGo.

Check whether you pointed to the local file (add an event handler for the OnBeforeConnect event of the FDConnection1 component).

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 XE5 Professional or higher
  - Delphi XE5 Professional with Mobile or higher
- 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.

**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.
Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)

**Note:** You can use FireDAC, dbExpress, and Interbase Express (IBX) components to build **Interbase ToGo** applications. For a detailed discussion on Interbase Express components usage in a Delphi application, read the [Getting Started with InterBase Express](#) article. For this tutorial, we will connect to **Interbase ToGo** using dbExpress framework.

### Using dbExpress to Connect to the Database

**dbExpress** is a very fast database access framework, written in Delphi. RAD Studio provides drivers for most major databases, such as InterBase, Oracle, DB2, SQL Server, MySQL, Firebird, SQLite and ODBC. You can access these different databases using procedures similar to the procedure described here.

- For the mobile platforms, dbExpress supports **InterBase ToGo** as well as **SQLite**. These database products can run on iOS and Android devices.
- For other databases, such as Oracle, you need to have at least a client library. On Windows platforms, the client library is provided as a DLL to connect to. Therefore, you need to develop applications using middle-tier technologies such as DataSnap to connect to these database products from a mobile device.
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 FireMonkey Mobile Application, select either of the following:
   - File > New > FireMonkey Mobile Application - Delphi > Blank Application
   - File > New > FireMonkey Mobile 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\14.0\Samples\Data\db demos.gdb (location of the database); then close the dialog box:
4. 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.

3. 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**.

4. Open the **LiveBindings Designer** and connect the data and the user interface as follows:

   1. Click **COMMON_NAME** in SQLDataSet1, and drag the mouse cursor to **Item.Text** in ListView1.
At this point, `TBindSourceDB` and `TBindingsList` components were added to the form.

2. Click `SPECIES_NAME` in `BindSourceDB1`, and drag the mouse cursor to `Item.Detail` in `ListView1`.

Deploying Your Application to Mobile

Up to this point, you have used InterBase on your desktop. This means that the actual database is located at your local hard disk drive (for example,
C:\Users\Public\Documents\Embarcadero\Studio\14.0\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.
- Change 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. Open the Deployment Manager by selecting Project > Deployment.
2. Select All-Configurations - iOS Device platform or All-Configurations - Android platform from the drop-down list of target platforms at the top of the Deployment Manager.
3. Select Add Featured Files (Add Featured Files):
4. 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_nnnnnn.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\redis\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.

  o **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**.

5. Select **Add Files** and select the database file (for example, C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\dbdemo.gdb).
6. Select `dbdemos.gdb` and change Remote Path to `StartUp\Documents\` (for iOS platform) or `assets\internal\` (for Android platform).

**RemotePath on iOS device platform**

**RemotePath on Android platform**

7. Select the Platforms column (double-click the ellipsis [...] in the row for `dbdemos.gdb`):

1. Ensure that iOS Simulator and iOS Device or Android are present for `dbdemos.gdb`.

2. Remove Win32 from the list if it is present (you do not have to copy database files to the Win32 platform).

Select All-Configurations - iOS Device platform or All-Configurations - Android platform, and make sure `dbdemos.gdb` is set to be deployed to `StartUp\Documents\` or `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 mobile app.

**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
     {$IFDEF(iOS) or DEFINED(ANDROID)}
     SQLConnection1.Params.Values['Database'] :=
       TPath.Combine(TPath.GetDocumentsPath, 'dbdemos.gdb');
     {$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 License Issues

If you get an error ("unavailabale database") when you connect to the database in the development environment, this means you forgot to deploy the Interbase ToGo license.

- To execute your application on mobile, deploy the `reg_ibtogo.txt` or `reg_iblete.txt` license files.
- The license files are located in
  C:\Users\Public\Documents\Embarcadero\InterBase\redist\InterBaseXE3 directory.

Exception Handling Issues

If your application raises an exception without having proper exception handling code, your mobile app 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++**

```c++
void __fastcall TForm1::Button1Click(TObject *Sender) {
  try {
    SQLConnection1->Connected = true;
    SQLDataSet1->Active = true;
  }
  catch(Exception *e) {
    ShowMessage(e->Message);
  }
```

**Typical Errors and Resolutions**

Following are typical errors that you might encounter when you connect to the database, and suggestions for resolving the issues:
Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)

**Error on mobile**

I/O error for file "/private/var/mobile/Applications/9379876A-EE1A-41C2-A1CS-860FBE07DF6/Documents/dbdemos.gdb"
Error while trying to open file
No such file or directory

**Suggestion**

Check whether the database file (dbdemos.gdb) is delivered to 'StartUp\Documents\' (for iOS) or 'assets\internal\' (for Android).

**Error on mobile**

Your user name and password are not defined. Ask your database administrator to set up an InterBase login.

**Suggestion**

Check whether the license file is delivered for InterBase ToGo.

**Error on mobile**

Unable to complete network request to host "C"
Failed to locate host machine.
The specified name was not found in the hosts file or Domain Name Services.

**Suggestion**

Check whether you pointed to the local file (add an event handler for the OnBeforeConnect event of the SQLConnection1 component).

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.

### iOS

- Apples
- Oranges
- Milk
- Beer
- Ice Cream
- Potato Chips
- Soft Drinks
- Coffee
- Pears
- Tea
- Bread

### Android

- Shopping List
  - Apples
  - Oranges
  - Milk
  - Beer
  - Ice Cream
  - Potato Chips
  - Soft Drinks
  - Coffee
  - Pears
  - Tea
  - Bread

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 FireMonkey Mobile Form Designer to design the user interface of your Mobile App.

1. To create an **HD FireMonkey Mobile Application**, select:
   - For Delphi: File > New > FireMonkey Mobile Application - Delphi > Blank Application
   - For C++: File > New > FireMonkey Mobile 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\14.0\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.

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**

![Design and Set Up the User Interface](image)

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**.
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**.
Mobile Tutorial: Using FireDAC and SQLite (iOS and Android)

- Set the Text to Delete.
- Set the Visible property to False.

4. Drop a TLabel on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Align to Client.
   - Set the StyleLookup to toollabel.
   - Set the Text to Shopping List.
   - Expand the TTextSettings node and set the HorzAlign property to Center.

5. Drop a TListView component on the form and set the following properties in the Object Inspector:
   - Set the Align property to Client, so that the ListView component uses the entire form.

Using the LiveBindings Wizard

Use the LiveBindings Wizard to add the LiveBindings components (TBindSourceDB, TBindingsList), and the TFDQuery component.

Add the LiveBinding components

2. Select LiveBindings Wizard.
3. Select Create a data source binding task.
4. Click the Next button.
5. Select FireDAC class name.
6. Click the Next button.
7. Change the **Command Type** to **Query**.

8. Set the **Command Text** property to **select ShopItem from Item**.

9. Click the **Test Command** button.

10. Click the **Next** button.

11. Click the **Finish** button.

At this point, **TBindSourceDB** and **TFDQuery** components were added to your form.

**Connecting to the Data**

1. Reopen the **LiveBindings Wizard**.

2. Select **Link a control with a field** binding task.
3. Click the **Next** button.
4. Select the **Existing Control** tab.
5. Select the **ListView1** component.
6. Click the **Next** button.
7. Select **BindSourceDB1**.
8. Click the **Next** button.
9. Select **ShopItem Field Name**.
10. Click the **Next** button.

11. Click the **Finish** button to close the wizard.

**Note:** These last steps are not mandatory for this tutorial since there is only one field in `BindSourceDB1`. These steps are useful to link with the selected value if we are managing several fields of a database.

**Displaying ShopItem in the ListView**

The following step displays the text of `ShopItem` in the `TListView` component.

1. In the **LiveBindings Designer**, select `ShopItem` in the `BindSourceDB1` component and drag `ShopItem` to `Item.Text` in `ListView1`. 
Following these steps connects the 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++:**

```c++
void __fastcall TForm1::ListView1ItemClick(const TObject *Sender,
                                          const TListItem *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.

4. 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
  TaskName: String;
begin
  try
    if InputQuery('Enter New Item', 'Name', TaskName) and (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;
```

**C++:**

```cpp
```
void __fastcall TForm1::ButtonAddClick(TObject *Sender) {
    String TaskName;
    try {
        if ((InputQuery("Enter New Item", "Name", TaskName)) &&
            (!Trim(TaskName) == "")) {
            FDQueryInsert->ParamByName("ShopItem")->AsString = TaskName;
            FDQueryInsert->ExecSQL();
            FDQuery1->Close();
            FDQuery1->Open();
            ButtonDelete->Visible = (ListView1->Selected != NULL);
        }
        catch (Exception &e) {
            ShowMessage(e.Message);
        }
    }
}

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.
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:
     
```sql
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
      ShowMessage(e.Message);
  end;
end;
```

**C++:**

```cpp
void __fastcall TForm1::ButtonDeleteClick(TObject *Sender) {
  String TaskName = ListView1->Selected->Text;
  try {
    FDQueryDelete->ParamByNamed("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 **TFDGUILxWaitCursor** 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\14.0\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.
- Change 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. Open the Deployment Manager by selecting Project > Deployment.
2. Select Add Files, and select the database file (for example, C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\shoplist.s3db).
3. Select shoplist.s3db, and change Remote Path to StartUp\Documents\ (for iOS platform) or assets\internal\ (for Android platform).

RemotePath on iOS device platform

RemotePath on Android platform

4. Select the Platforms column (double-click the ellipsis [...] in the row for shoplist.s3db):
   1. Ensure that iOS Simulator and iOS Device or Android are present for shoplist.s3db.
   2. Remove Win32 from the list if it is present (you do not have to copy database files to the Win32 platform).
5. Select All-Configurations - iOS Device platform or All-Configurations - Android platform and make sure shoplist.s3db is set to be deployed to StartUp\Documents\ or 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 mobile app.

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) {
    #if defined(__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");
}
```

**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
**Note**: If you have an issue with running the application, follow the steps given in **Troubleshooting**.

### See Also

- [Mobile Tutorial: Using InterBase ToGo with FireDAC (iOS and Android)](#)
- [Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)](#)
- [SQLite support in RAD Studio](#)
- [Android Mobile Application Development](#)
- [iOS Mobile Application Development](#)
Mobile Tutorial: Using dbExpress and SQLite (iOS and Android)

⚠️ **Caution:** dbExpress, which is described in this tutorial, is being deprecated. This means that dbExpress will be removed from RAD Studio in an upcoming release.

Instead of dbExpress, we recommend that you use our newer database solution, FireDAC, which is described in a similar tutorial, here:

Mobile Tutorial: Using FireDAC and SQLite (iOS and Android).

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

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

This tutorial describes the basic steps to use SQLite as a local data storage on your mobile device through the dbExpress framework.
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 FireMonkey Mobile Form Designer to design the user interface of your Mobile App.

Create the Database in the Data Explorer

1. Go to Data Explorer, right-click the SQLite node and select Add New Connection:

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

3. Specify the location of the database file:

4. Click the Advanced button and open the Advanced Properties dialog box.

5. Change the FailIfMissing property to False and click OK to close the Advanced Properties dialog box:
Mobile Tutorial: Using dbExpress and SQLite (iOS and Android)

Note: Setting FailIfMissing to False instructs the Data Explorer to create a new database file if the file is not available.

6. Back on the Modify Connection dialog box, click the Test Connection button. With this operation, the new database file is created if no file existed:

Note: Ensure that sqlite3.dll is present on your development system. If this file is not present, download sqlite3.dll from http://www.sqlite.org/download.html to your system path (such as C:\Windows\SysWOW64 for 64-bit Windows).

Create Table on DataExplorer

1. On the Data Explorer, double-click the ShoppingList node under the SQLite section, right-click Tables, and then select New Table from the context menu.
2. Set **Data Type** for a **ShopItem** column to **TEXT**.

3. Click the **Save** button and specify a table name (for example, **Item**.)
Design and Set Up the User Interface

Visible UI components are loaded on the designer

This tutorial uses one TListBox component as the UI element.

To set up a ListBox component and other UI elements, use the following steps:


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 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 Name property to ButtonDelete.
Mobile Tutorial: Using dbExpress and SQLite (iOS and Android)

5. Drop a TLabel on the ToolBar component and set the following properties in the Object Inspector:
   - Set the Align to Client.
   - Set the Text to Shopping List.
   - Set the TextSettings.HorzAlign to Center.

6. Drop a TListBox component on the form and set the following properties in the Object Inspector:
   - Set the Align property to Client, so that the ListBox component uses the entire form.

Connecting to the Data

Following are the basic steps to connect to data in a database which is already defined in the Data Explorer:

1. Select the Item table on the Data Explorer and drag it to the Form Designer.

Note: This creates two components (ShoppinglistConnection: TSQLConnection and ItemTable: TSQLDataSet) on the form.
2. Select the **ShoppinglistConnection** component on the form, and then change the **Connected** property to **True**.

3. Select the **ItemTable** component on the form, and then change the **Active** property to **True**.

4. Select **View > LiveBindings Designer** and the **LiveBindings Designer*** opens.

5. Select **ShopItem** in the **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:
     
     `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
     TaskName: String;
   begin
     try
       if InputQuery('Enter New Item', 'Name', TaskName) and (TaskName.Trim <> '') 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
     end;
   end;
   ```
ButtonDelete.Visible := False;
end;
except
  on e: Exception do
    begin
      ShowMessage(e.Message);
      end;
end;
end;
end;

C++Builder:

void __fastcall TForm1::ButtonAddClick(TObject *Sender) {
  String TaskName;
  try {
    if ((InputQuery("Enter New Item", "Name", TaskName))
        && (!Trim(TaskName) == "")) {
      SQLQueryInsert->ParamByName("ShopItem")->AsString = TaskName;
      SQLQueryInsert->ExecSQL();
      ItemTable->Refresh();
      LinkFillControlToField1->BindList->FillList();
      if (ListBox1->Selected)
        ButtonDelete->Visible = True;
      else
        ButtonDelete->Visible = False;
    }
  } catch (Exception &e) {
    ShowMessage(e.Message);
  }
}

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.
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:
     
     ```sql
     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->ExecuteSQL();
    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**

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\14.0\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.
- Change 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. Open the **Deployment Manager** by selecting **Project > Deployment**.
2. Select **Add Files**, and select the database file (for example, C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\shoplist.s3db).
3. Select **shoplist.s3db**, and change **Remote Path** to **StartUp\Documents** (for iOS platform) or **assets\internal** (for Android platform).

**RemotePath on iOS device platform**

**RemotePath on Android platform**

4. Select the **Platforms** column (double-click the ellipsis [...] in the row for shoplist.s3db):
   1. Ensure that **iOS Simulator** and **iOS Device** or **Android** are present for shoplist.s3db.
   2. Remove Win32 from the list if it is present (you do not have to copy database files to the Win32 platform).
5. Select **All-Configurations - iOS Device** platform or **All-Configurations - Android** platform and make sure **shoplist.s3db** is set to be deployed to **StartUp\Documents** or **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 mobile app.

**Modifying Your Code to Connect to a Local Database File on mobile**

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 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
Note: If you have an issue with running the application, follow the steps given in Troubleshooting.

See Also

- Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)
- Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (iOS and Android)
- SQLite support in RAD Studio
- Android Mobile Application Development
- iOS Mobile Application Development
Mobile Tutorial: Connecting to an Enterprise Database from a Mobile Client (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)**
- **Mobile Tutorial: Using InterBase ToGo with dbExpress (iOS and Android)**

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 mobile 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. Create a new project. Choose **File > New > Other** and from the New Items dialog select **Delphi Projects > DataSnap Server > DataSnap Server** or **C++Builder Projects > DataSnap Server > DataSnap Server** in order to create a new Delphi or C++ project.

![New DataSnap Server Wizard](image)

2. The **New DataSnap Server** wizard appears and you need to follow its steps without modifying too many parameters.
In the New DataSnap Server wizard:

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.

3. Save the form unit as **DataSnapServerUnit.pas**.
4. Switch to **DataSnapServerUnit**, and change the **Name** property of the Form to **DSServerForm**.
5. Save the server methods unit (by default as created by the Wizard: **ServerMethodsUnit1**) as **ServerModuleUnit.pas**.
6. Save the server container unit (by default as created by the Wizard: **ServerContainerUnit1**) as **ServerContainerUnit.pas**.
7. Save the new project as **DataSnapServerProject.droj**.
8. Select ProjectGroup1 in the Project Manager, and save the project as DataSnapTutorialProjectGroup1.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.
     - Expand the Driver node, and set the DataBase property to C:\Users\Public\Documents\Embarcadero\Studio\14.0\Samples\Data\EMPLOYEE.GDB.
     - Change the Connected property to True. If you get an error, double-check the Driver properties:
Drop a `TSQLDataSet` component on the Server Module, and set the following properties:

`TSQLDataSet` represents the data retrieved using dbExpress.

- Set the `Name` property to `SQLDataSet_EMPLOYEE`.
- Set the `SQLConnection` property to `SQLConnection_EMPLOYEE`.
- Set the `CommandType` property to `ctTable`.
- Set the `CommandText` property to `EMPLOYEE`.
- Change the `Active` property to `True`. If you get an error, double-check the properties you just configured.

Drop a `TDataSetProvider` component on the Server Module, and set the following properties:

`TDataSetProvider` packages data from a dataset and passes one or more transportable data packets to the DataSnap client.

- Set the `Name` property to `DataSetProvider_EMPLOYEE`.
- Set the `DataSet` property to `SQLDataSet_EMPLOYEE`.
Note: This tutorial uses InterBase as an example. However, you can connect to any database server using the same steps. Select the proper driver, and other properties to point to your database.

Expose the DataSet from the DataSnap Server

You have just created a new Server Module that contains a DataSet and a DataSetProvider that packages data to the next layer. The next step is to expose the Server Module to the DataSnap client.

1. In the Form Designer, open ServerContainerUnit.
2. Select DSServerClass1, and update the existing event handler for the OnGetClass event. Add the following code to the DSServerClass1 event handler:

Delphi:

```delphi
procedure TServerContainer1.DSServerClass1GetClass(DSServerClass: TDSServerClass;
var PersistentClass: TPersistentClass);
begin
  PersistentClass := TDSServerModule_EMPLOYEE;
end;
```

C++ (only for iOS):

```c++
void __fastcall TServerContainer1::DSServerClass1GetClass(TDSServerClass *
DSServerClass,
  TPersistentClass &PersistentClass) {
  PersistentClass = __classid(TDSServerModule_EMPLOYEE);
}
```

With this event handler, the DataSnap Server exposes providers as well as public methods in this class to a DataSnap client. Based on the steps in the previous section, now you are going to expose the DataSetProvider_EMPLOYEE DataSetProvider component to your DataSnap client.
Run the DataSnap Server

Implementation of the DataSnap Server is complete. Right-click DataSnapServerProject.exe and select Run Without Debugging.

Now you can see the DataSnap server running on your Windows machine. Because this DataSnap server has no UI element, it looks like a blank form, and this is as expected at this point.

Creating a Mobile Application that Connects to the DataSnap Server

The next step is creating the mobile client application.

1. In the Project Manager, right-click DataSnapTutorialProjectGroup, and select Add New Project.
2. Select **FireMonkey Mobile Application** on the **Delphi Projects** page:

3. Save the new Unit as **DataSnapClientUnit.pas**.

4. Save the new Project as **DataSnapClientProject.dproj**.

5. Open **DataSnapClientUnit**, and change the **Name** property of the Form to **DSClientForm**.

6. Drop the following components on the **FireMonkey Mobile 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:
Deploy the MIDAS Library to iOS Simulator

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

- MIDAS Library

1. Open the Deployment Manager by selecting Project > Deployment.
2. Select Add Featured Files (⌘):
3. Select the following module, and then click **OK** to close the Deployment Manager:
   - MIDAS Library

   ![Featured Files Image]

### 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](#)
- [Ddatasnap.DSServer.TDSServer](#)