RAD Studio
iOS Tutorials: Delphi iOS Application Development

Version XE4
© 2013 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.

April, 2013
iOS Tutorial: Using Notification Center on the iOS Device
Add Action to the Notification Alert ............................................. 79
Notification Banner or Notification Alert ........................................ 80
Schedule Notification .................................................................... 81
Set the Icon Badge Number from Code ............................................ 82
Access the Notification Service ....................................................... 83
iOS Tutorial: Taking and Sharing a Picture in an iOS Application .. 84
Add Detail Information to an Item .................................................. 75
Add Items to a ListBox from Your Code .......................................... 76
Add a Search Box .......................................................................... 78
See Also ....................................................................................... 79
Every FireMonkey Component Can Have an Owner, a Parent and Children . 80
Using Common Layout-Related Properties of a FireMonkey Component ... 81
Using the Align Property ............................................................... 81
Using the Margins Property ........................................................... 82
Using the Padding Property ........................................................... 82
Using the Anchors Property ........................................................... 83
Using the TLayout Component ....................................................... 84
Working with a Busy Interface: Using a TVertScrollBox Component ... 85
See Also ....................................................................................... 86
Building the User Interface for the Application ............................. 88
Taking a Picture with the iOS Device Camera .................................. 88
Using a Picture from the iOS Device Photo Library ....................... 89
Sharing or Printing a Picture .......................................................... 90
See Also ....................................................................................... 91
iOS Tutorial: Using Location Sensors on the iOS Device ............... 92
Design the User Interface ............................................................... 93
The Location Sensor ..................................................................... 94
Read Location Information (Latitude, Longitude) from the LocationSensor Component ....................................................... 94
Show the Current Location Using Google Maps via a TWebBrowser Component ........................................................... 95
Use Reverse Geocoding ................................................................. 96
Show a Readable Address in the ListBox Component .................... 98
See Also ....................................................................................... 98
iOS Tutorial: Using Notification Center on the iOS Device .......... 99
Three Basic Notification or Alert Styles ....................................... 99
Badge on Application Icon ............................................................ 99
Notification Banner on iPad .......................................................... 99
Notification Alert ......................................................................... 99
Notification Center on iPad ........................................................... 100
Access the Notification Service .................................................... 100
Set the Icon Badge Number from Code ....................................... 101
Schedule Notification .................................................................... 102
Update or Cancel a Scheduled Notification Message .................... 103
Present the Notification Message Immediately ............................. 104
Notification Banner or Notification Alert ...................................... 105
Add Action to the Notification Alert .............................................. 106

Embarcadero Technologies
IOS TUTORIALS: DELPHI iOS APPLICATION DEVELOPMENT

This integrated set of tutorials walks you through development of an iOS application using RAD Studio:

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

SETUP

- [Set Up Your Development Environment on the Mac](#)
- [Set Up Your Development Environment on Windows PC](#)

USING BASIC USER INTERFACE ELEMENTS

- [Creating a FireMonkey iOS Application](#)
- [Using a Button Component with Different Styles in an iOS Application](#)
Using a Calendar Component to Pick a Date in an iOS Application

Using Combo Box Components to Pick Items from a List in an iOS Application

Using the Web Browser Component in an iOS Application

Using Tab Components to Display Pages in an iOS Application

Using ListBox Components to Display a Table View in an iOS Application

Using Layout to Adjust Different Form Sizes or Orientations in an iOS Application

Using Device Functionality

Taking and Sharing a Picture in an iOS Application
- Using Location Sensors on the iOS Device
- Using the Notification Center on the iOS Device

ACCESSING A DATABASE

- Using InterBase ToGo in an iOS Application
- Using SQLite in an iOS Application
- Connecting to an Enterprise Database from an iOS Client Application

SEE ALSO

- FireMonkey Quick Start
- Creating a FireMonkey iOS App
- FireMonkey Application Design
- iOS Code Snippets
- iOS Mobile Application Development
**iOS Tutorial: Set Up Your Development Environment on the Mac**

A FireMonkey application destined for the iOS target platform is 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.

- The first half of this tutorial describes the steps that you need to perform in order to run your iOS application on the **iOS Simulator** on the Mac.
- The second half of this tutorial describes additional steps required in order to run your iOS Application on your **iOS Device**.

**Requirements on the Mac**

- OS X 10.7 Lion or 10.8 Mountain Lion
  (Neither OS is supported on legacy PowerPC- and 680x0-based Macintosh systems. All Macs since 2007 are Intel-based; all since 2008 are 64-bit, which Lion requires.)
- iOS 5.1 and above
- Latest version of Xcode and the iOS SDK installed, and the Xcode command line tools installed
  - Requires 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 on the iOS Simulator**

To deploy an iOS application to the **iOS Simulator** on the Mac, 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.

![Xcode Diagram](image)

**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 **RADPAServerXE4.pkg**, and it is available in two places:

- In the **PAServer** folder inside the RAD Studio installation directory.
  
  For example, `C:\Program Files\Embarcadero\RAD Studio\11.0\PAServer\RADPAServerXE4.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 (RAD PA Server XE4.app) as follows:

1. Navigate to the top-level **Applications** folder.
2. Double-click **RAD PA Server XE4.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 3.0.4.02
   Copyright © 2009-2013 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**:

![Developer Tools Access is trying to take control of another process. Type your password to allow this.](image)

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](https://appstore.com), download Xcode for free.
- As a registered Apple Developer, you can download the latest version of Xcode as a bundle (`.dmg`). To register and then download Xcode:
**Next Steps**

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

- To run an iOS Application now on the **iOS Simulator**, see [iOS Tutorial: Set Up Your Development Environment on Windows PC](#) to complete the configuration of your RAD Studio IDE.

- 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 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**.
**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:

   ![Certificate Assistant](image)

   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 for details.

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

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 iOS Tutorial: Set Up Your Development Environment on Windows PC 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

- iOS Tutorial: Set Up Your Development Environment on Windows PC
- paserver, the Platform Assistant Server Application
- Apple Developer Program
- iOS Developer Program
- Creating and Configuring App IDs
- Creating signing certificates
- iOS Provisioning Portal
- Devices tab of the Apple Provisioning Portal
- Create an Apple ID
- Creating and Downloading a Distribution Provisioning Profile
- Installing the Platform Assistant on a Mac
- Running the Platform Assistant on a Mac
**iOS Tutorial: Set Up Your Development Environment on Windows PC**

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

- **iOS Tutorial: Set Up Your Development Environment on the Mac**

A FireMonkey application destined for the iOS target platform is 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. 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:

![Connection Profile Manager](image.png)
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 target platform, and then click **Next**:

![Create a Connection Profile](image1)

4. On the 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"):

![Test Connection](image2)
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 profile, the IDE fills a Profile (such as "My Mac") and SDK version combo box with the list of SDK versions available on the target platform:

![Add a New SDK dialog](image)

Click **OK** to close the dialog.

**SEE ALSO**

- iOS Tutorial: Creating a FireMonkey iOS Application
- Working with a Mac and a PC
- Running Your Application on an iOS Device
- Running Your Application on the iOS Simulator
- FireMonkey Platform Prerequisites
- Creating a FireMonkey 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
  - iAd Network
IOS TUTORIAL: CREATING A FIREMONKEY IOS APPLICATION

This topic describes how to create a "Hello World" FireMonkey application for the iOS target platform.

BEFORE YOU START

To develop iOS 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:

- iOS Tutorial: Set Up Your Development Environment on the Mac
- iOS Tutorial: Set Up Your Development Environment on Windows PC

STEP 1: CREATE A NEW FIREMONKEY APPLICATION FOR IOS

1. Select File > New > FireMonkey Mobile Application - Delphi:
2. Select **Blank Application**.

The **FireMonkey Mobile Form Designer** shows a new form for an iOS Application:

![FireMonkey Mobile Form Designer](image)

**STEP 2: PLACE COMPONENTS ON THE FIREMONKEY IOS FORM**

The first step in creating a FireMonkey iOS application is designing the user interface, the same 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 drop it onto the **Form Designer**. An instance of the TEdit component appears on the form:

![Tool Palette](image)
Repeat these steps, but now add a TLabel and a TButton component to the form. Select the button and change the Text property in the Object Inspector to "Say Hello".

Now you should see three components on the Form Designer:

After you place these components on the Form Designer, the IDE automatically sets names for the components.
To see or to change the name of a component, click the component on the Form Designer, and then find its **Name** property on the **Object Inspector** and the **Structure View**:

![Image showing Form Designer and Object Inspector](image)

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).

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**:

![Image showing Structure View and Object Inspector](image)
You can easily switch to source code by selecting the **Code** tab at the bottom of the Form Designer or pressing the **F12** key. You can switch between the **Form Designer** and the **Code Editor** any time you want:

When you switch to the **Code Editor**, you can see the source code that the IDE has generated. You should find three components defined (Edit1, Label1, and Button1):
STEP 3: WRITE AN EVENT HANDLER IN DELPHI 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 iOS 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
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Text := 'Hello ' + Edit1.Text + '!';
end;
end.
```

Now you can implement responses within the `begin` and `end` statements of the `Button1Click` method.

The following code snippet implements a response that displays a small dialog box, which reads "Hello + <name entered into the edit box>":

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 iOS APPLICATION ON THE MAC (iOS SIMULATOR)**

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:

By default, FireMonkey 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:
STEP 5: TEST YOUR iOS APPLICATION ON A CONNECTED iOS DEVICE

If you complete the steps described in iOS Tutorial: Set Up Your Development Environment on the Mac and iOS Tutorial: Set Up Your Development Environment on Windows PC 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 (available in $(BDS)\bin\Artwork\iOS):

---

**SEE ALSO**

- iOS Tutorial: Using a Button Component with Different Styles in an iOS Application
- iOS Mobile Application Development
- Mac OS X Application Development
- iOS Code Snippets
**iOS Tutorial: Using a Button Component with Different Styles in an iOS Application**

**Buttons in FireMonkey iOS 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 iOS application:

- **Buttons on the Form:**
  ![Button Examples](image)

- **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:
CREATE A SEGMENTED CONTROL USING BUTTON COMPONENTS

FireMonkey uses the Button 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:

![Segmented Control on Toolbar](image)

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

![Object Inspector and Properties](image)

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

![Segments are Pressed](image)

**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`

![Scope Bar Example](image)
SEE ALSO

- iOS Tutorial: Creating a FireMonkey iOS Application
- iOS Tutorial: Using a Calendar Component to Pick a Date in an iOS Application
- FMX.StdCtrls.TButton
- FMX.Controls.TStyledControl.StyleLookup
- FMX.StdCtrls.TToolBar
**iOS Tutorial: Using a Calendar Component to Pick a Date in an iOS Application**

**Calendar in FireMonkey iOS Applications**

FireMonkey uses the `TCalendarEdit` component to wrap a calendar component or datepicker for the iOS target platform:

![Calendar Component Example](image)

You can easily use the `TCalendarEdit` component with the following simple steps:

1. Select the `TCalendarEdit` 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 ("Cale") in the search box (🔍):

![Tool Palette Search](image)
After you drop the component, you can see your TCalendarEdit component on the Mobile Form Designer:

2. Basically, that’s it. Run your application on either the iOS Simulator or your connected iOS Device. After you tap TCalendarEdit, 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 an OnChange event handler:**

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

```pascal
procedure TForm25.CalendarEdit1Change(Sender: TObject);
begin
  ShowMessage(FormatDateTime('dddddd', CalendarEdit1.Date));
end;
```

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):
SEE ALSO

- iOS Tutorial: Using a Button Component with Different Styles in an iOS Application
- iOS Tutorial: Using Combo Box Components to Pick Items from a List in an iOS Application
- Date and Time Support
- Type conversion routines
**iOS Tutorial: Using Combo Box Components to Pick Items from a List in an iOS Application**

**Implementing a Picker in FireMonkey iOS Applications**

On the iOS platform, FireMonkey wraps the Picker component with the `TComboBox` component:

To define a picker and the list items to pick:

1. Select **File > New > FireMonkey Mobile Application - Delphi > 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), the first item in the list is "Alabama".
7. Edit other items as well, such as Alaska, Arizona, Arkansas, California, Colorado, and so forth.
8. Run the application on either the iOS Simulator or the iOS Device target platform.
   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 can use the Add method as follows:

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

**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:

```pascal
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 := 4;
end;
```
If you do not know the index value, you can find the value by using the `IndexOf` method as follows:

```pascal
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;
```

**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.

**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 `OnClick`.
3. The Code Editor opens. Write code as follows:

```pascal
procedure TForm27.CalendarEdit1Change(Sender: TObject);
begin
    ShowMessage(Format('Item %s at Index %d was selected.',
        [ComboBox1.Selected.Text, ComboBox1.ItemIndex]));
end;
```

This event handler displays a message dialog that indicates the item that was selected.
The **Format** function returns a formatted string assembled from a format string and an array of arguments:

### SEE ALSO

- [iOS Tutorial: Using a Calendar Component to Pick a Date in an iOS Application](#)
- [iOS Tutorial: Using Tab Components to Display Pages in an iOS Application](#)
- [iOS Mobile Application Development](#)
- [Mac OS X Application Development](#)
**iOS Tutorial: Using the Web Browser Component in an iOS Application**

**Using the Web Browser Component in FireMonkey iOS Applications**

On the iOS platform, FireMonkey wraps the Web Browser component as the `TWebBrowser` component. This topic describes how to create a simple FireMonkey Web Browser application for iOS.
STEP 1: 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:

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

4. Select the TButton component in the Tool Palette and drop it on the ToolBar.
5. Select the Button Component on the Mobile Form Designer, and then select priortoolbuttonbordered in the StyleLookup property in the Object Inspector.

The priortoolbuttonbordered StyleLookup value for TButton adds a Back button label in the iOS style:

For more detail about selecting a style in FireMonkey iOS applications, see iOS Tutorial: Using a Button Component with Different Styles in an iOS Application.
6. Select the `TEdit` component in the Tool Palette and drop it on the ToolBar. Make sure the size of the Edit control is wide enough to fill the area of the ToolBar:

![Edit control on ToolBar]

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

After you complete these steps, the form should look like the following picture:

![Web browser on form]
STEP 2: 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 image. The user can complete the action by clicking either "Done" or "Return". 

![Virtual Keyboard Image]

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 "Return" 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 new line:

   ```
   procedure OpenURL; next to { Private declarations }
   ```


type
 TForm34 = class(TForm)
  ToolBar1: TToolBar;
  Button1: TButton;
  Edit1: TEdit;
  WebBrowser1: TWebBrowser;
private
  { Private declarations }
  procedure OpenURL;
public
  { Public declarations }
end;

2. Press **CTRL+SHIFT+C** to create a placeholder at the end of the file:

   ```
   procedure TForm34.OpenURL;
   begin
   end;
   ```

3. Implement the **OpenURL** method as in the following code snippet:

   ```
   procedure TForm34.OpenURL;
   begin
     WebBrowser1.Navigate(Edit1.Text);
   end;
   ```

**IMPLEMENT AN EVENT HANDLER FOR THEOnChange EVENT**

1. In the Mobile Form Designer, select the Edit component, and then in the Object Inspector (Events tab), double-click the white space next to the **OnChange** event to create the event handler.
The Object Inspector creates a new event handler called **Edit1Change**: 
2. Complete the event handler by adding the following code:

```pascal
procedure TForm34.Edit1Change(Sender: TObject);
begin
    OpenURL;
end;
```

**IMPLEMENT AN EVENT HANDLER TO SUPPORT THE ENTER KEY**

There is no specific event defined for the Enter key. However, you can still monitor such events through the `OnKeyDown` event.

`OnKeyDown` gives information about the pressed key through several parameters in its event handler. You can implement this event as follows:

```pascal
procedure TForm34.Edit1KeyDown(Sender: TObject; var Key: Word;
    var KeyChar: Char; Shift: TShiftState);
begin
    if (Key = vkReturn) then
        begin
            OpenURL;

            // Change Focus to Back Button so that virtual keyboard is closed.
            Button1.SetFocus;
        end;
end;
```

**IMPLEMENT AN EVENT HANDLER FOR THE BACK BUTTON**

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

```pascal
procedure TForm34.Button1Click(Sender: TObject);
begin
    WebBrowser1.GoBack;
end;
```

The basic behavior is now implemented for this Web Browser application. Try running your application on either the iOS Simulator or your iOS Device.
**STEP 3: SELECT THE PROPER 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>vktAlphabet</th>
<th>vktDefault</th>
<th>vktEmailAddress</th>
<th>vktNamePhonePad</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Virtual Keyboard 1" /></td>
<td><img src="image2.png" alt="Virtual Keyboard 2" /></td>
<td><img src="image3.png" alt="Virtual Keyboard 3" /></td>
<td><img src="image4.png" alt="Virtual Keyboard 4" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vktNumberPad</th>
<th>vktNumbersAndPunctuation</th>
<th>vktPhonePad</th>
<th>vktURL</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.png" alt="Virtual Keyboard 5" /></td>
<td><img src="image6.png" alt="Virtual Keyboard 6" /></td>
<td><img src="image7.png" alt="Virtual Keyboard 7" /></td>
<td><img src="image8.png" alt="Virtual Keyboard 8" /></td>
</tr>
</tbody>
</table>

Select **vktURL** as the proper virtual keyboard in the **KeyboardType** property for the Edit control:
See Also

- iOS Tutorial: Using Combo Box Components to Pick Items from a List in an iOS Application
- iOS Tutorial: Using Tab Components to Display Pages in an iOS Application
- TWebBrowser
- TToolBar
- TButton
- TEdit
- KeyboardType
- StyleLookup
iOS TUTORIAL: USING TAB COMPONENTS TO DISPLAY PAGES IN AN iOS APPLICATION

TABLES IN FIREMONKEY iOS 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 predefined icons as well as a custom icon, and a text label.

Also, you can place tabs at either the top or the bottom of the control.

DESIGN TAB PAGES USING THE FORM DESIGNER

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

1. To create an HD FireMonkey mobile application, select File > New > FireMonkey Mobile Application - Delphi > Blank Application. Use the default target platform for iPhone (iOSSimulator).
2. Select TTabControl from the Tool Palette:

![Tool Palette](image)

3. After you drop the TTabControl, an empty TabControl is shown on the FireMonkey Mobile Form Designer:

![FireMonkey Mobile Form Designer](image)
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 **alClient**:

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 to **tabitembookmarks**:

![Image of FireMonkey Mobile Form Designer]

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**:

![Image of 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**:

<table>
<thead>
<tr>
<th><strong>tpTop</strong></th>
<th><strong>tpBottom</strong></th>
<th><strong>tpDots</strong></th>
<th><strong>tpNone</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabs are displayed at the Top.</td>
<td>Tabs are displayed at the Bottom.</td>
<td>No Tabs are displayed. However, Dots or ellipsis ([...]) are displayed to indicate additional pages.</td>
<td>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.</td>
</tr>
</tbody>
</table>
USE CUSTOM ICONS FOR YOUR TABS

You can use custom icons as well as custom text for tab pages by following these steps:

1. Place a TabControl, set its alignment (Align property), and create several tabs on it:

2. Select a tab, and select the ellipsis button [...] in the Bitmap field of the CustomIcon property of TTabItem in the Object Inspector:

3. Select Edit... from the drop-down menu:
4. 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 (you set the BitmapHiRes icon in the next step):

![Bitmap Editor](image)

5. Close the Bitmap Editor, and in the Object Inspector select the icon you want to use for the BitmapHiRes (high-resolution) field of CustomIcon.

7. In the **Text** property, change the text on the tab.

![Image of Object Inspector with Text property set to "People"]

The custom glyphs used in this tutorial are available in your `$BDS/Images/GlyFX` directory.

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:

<table>
<thead>
<tr>
<th>Original Image</th>
<th>Selected Image</th>
<th>Non-Selected Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Original Image" /></td>
<td><img src="image2.png" alt="Selected Image" /></td>
<td><img src="image3.png" alt="Non-Selected Image" /></td>
</tr>
</tbody>
</table>
DEFINE 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**:
CHANGING THE PAGE AT RUN TIME

BY THE USER TAPPING THE TAB

If Tabs are visible (when TabPosition property is set to other than tpNone), end user can simply Tap to Tab to switch between pages.

BY THE 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 aTabControl, 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:
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):

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

![Image of Object Inspector and ChangeTabAction widget]

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

![Image of Object Inspector showing ChangeTabAction properties]

**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:

```pascal
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).

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

• If **ChangeTabAction** is defined, you can execute an action from your code as well:

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

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

**SEE ALSO**

- [iOS Tutorial: Using a Button Component with Different Styles in an iOS Application](#)
- [iOS Tutorial: Using the Web Browser Component in an iOS Application](#)
- [iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application](#)
**iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application**

**Using ListBox Components to Display a Table View in an iOS Application**

On the iOS platform, FireMonkey uses the `FMX.ListBox TLabelBox` component to present a Table View in the iOS style, like these ListBoxes:

![Table View in iOS Style](image)

This tutorial describes the basic steps to build items for a Table View in your FireMonkey iOS application.
CREATE ITEMS ON THE ListBox COMPONENT

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 "tlist") in the Search box of the Tool Palette:

![Tool Palette with TListBox selected](image1)

3. Select the ListBox component on the Mobile Form Designer, go to the Object Inspector and select alClient for the Align property:

![Object Inspector with alClient selected](image2)
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>alClient</td>
</tr>
<tr>
<td>StyleLookup</td>
<td>toollabel</td>
</tr>
<tr>
<td>TextAlign</td>
<td>taCenter</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**:

   ![Items Designer](image1)

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

     ![Items Designer](image2)
5. Close the dialog box. Now you have a Group Header and a Group Footer on the TListBox component.

**SHOW LIST ITEMS AS SEPARATE GROUPED ITEMS**

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

<table>
<thead>
<tr>
<th>Show Items as Plain List</th>
<th>Show Items as Grouped List</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListBoxGroupHeader1</td>
<td>ListBoxGroupHeader1</td>
</tr>
<tr>
<td>ListBoxItem1</td>
<td>ListBoxItem1</td>
</tr>
<tr>
<td>ListBoxItem2</td>
<td>ListBoxItem2</td>
</tr>
<tr>
<td>ListBoxItem3</td>
<td>ListBoxItem3</td>
</tr>
<tr>
<td>ListBoxItem4</td>
<td>ListBoxItem4</td>
</tr>
<tr>
<td>ListBoxGroupFooter1</td>
<td>ListBoxGroupFooter1</td>
</tr>
</tbody>
</table>

gsPlain = GroupingKind Property Value  
gsGrouped = GroupingKind Property Value  
listboxstyle = StyleLookup Property Value  
transparentlistboxstyle = StyleLookup Property Value

You can select either style for your TListBox component in the Object Inspector.
ADD A CHECK BOX OR OTHER ACCESSORY TO A LISTBOX ITEM

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

![Values for ItemData.Accessory]

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

![Object Inspector]

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:

<table>
<thead>
<tr>
<th>StyleLookup property</th>
<th>Look &amp; Feel</th>
</tr>
</thead>
<tbody>
<tr>
<td>listBoxItemnodetail</td>
<td>ListBoxItem1</td>
</tr>
<tr>
<td>listBoxItembottomdetail</td>
<td>ListBoxItem2</td>
</tr>
<tr>
<td></td>
<td>Detail</td>
</tr>
<tr>
<td>listBoxItemrightdetail</td>
<td>ListBoxItem3</td>
</tr>
<tr>
<td></td>
<td>Detail</td>
</tr>
<tr>
<td>listBoxItemleftdetail</td>
<td>ListBoxItem4</td>
</tr>
<tr>
<td></td>
<td>Detail</td>
</tr>
</tbody>
</table>
ADD ITEMS TO A ListBox FROM YOUR CODE

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

```csharp
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 code adds items to a ListBox, as shown in the picture:
procedure TForm40.FormCreate(Sender: TObject);
var
  c: Char;
  i: Integer;
  Buffer: String;
  ListBoxItem : TListBoxItem;
  ListBoxGroupHeader : TListBoxGroupHeader;
begin
  ListBox1.BeginUpdate;
  for c := 'a' to 'z' do
  begin
    // Add header ('A' to 'Z') to the List
    ListBoxGroupHeader := TListBoxGroupHeader.Create(ListBox1);
    ListBoxGroupHeader.Text := UpperCase(c);
    ListBox1.AddObject(ListBoxGroupHeader);
    // Add items ('a', 'aa', 'aaa', 'b', 'bb', 'bbb', 'c', ...) to the list
    for i := 1 to 3 do
    begin
      // StringOfChar returns a string with a specified number of repeating characters.
      Buffer := StringOfChar(c, i);
      // Simply add item
      // ListBox1.Items.Add(Buffer);
      // or, you can add items by creating an instance of TListBoxItem by yourself
      ListBoxItem := TListBoxItem.Create(ListBox1);
      ListBoxItem.Text := Buffer;
      // (aNone=0, aMore=1, aDetail=2, aCheckmark=3)
      ListBoxItem.ItemData Accessory := TListBoxItemData.TAccessory(i);
      ListBox1.AddObject(ListBoxItem);
    end;
  end;
  ListBox1.EndUpdate;
end;
**ADD A 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:

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

- FMX.ListBox.TListBox
- iOS Tutorial: Using a Button Component with Different Styles in an iOS Application
- iOS Tutorial: Using the Web Browser Component in an iOS Application
- iOS Tutorial: Using Tab Components to Display Pages in an iOS Application
**IOS TUTORIAL: USING LAYOUT TO ADJUST DIFFERENT FORM SIZES OR ORIENTATIONS IN AN IOS APPLICATION**

This tutorial describes a general strategy for using one common form for different form factors (such as iPhone and iPad), 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:

![Dropdown for device selection](image)

---

**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 alNone, which means that no automatic calculations are performed; the control stays where it is placed.

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

<table>
<thead>
<tr>
<th>alNone</th>
<th>alTop</th>
<th>alBottom</th>
<th>alLeft</th>
<th>alRight</th>
<th>alCenter</th>
<th>alClient</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="alNone" /></td>
<td><img src="image2.png" alt="alTop" /></td>
<td><img src="image3.png" alt="alBottom" /></td>
<td><img src="image4.png" alt="alLeft" /></td>
<td><img src="image5.png" alt="alRight" /></td>
<td><img src="image6.png" alt="alCenter" /></td>
<td><img src="image7.png" alt="alClient" /></td>
</tr>
</tbody>
</table>

If you use an Align value of alTop, alBottom, alLeft or alRight for one component, the Align properties for other components use the remaining area.

The size and shape of the remaining area (alClient) 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 alTop, and one (1) component that uses alClient.

**USING THE MARGINS PROPERTY**

Margins ensure separation between controls automatically positioned by a parent.

In the following picture, the right side of the component (alClient) 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 Padding diagram](image)

**USING THE ANCHORS PROPERTY**

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

**Anchors Property for the Edit Control**

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

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

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:

<table>
<thead>
<tr>
<th>1. Initial State</th>
<th>2. Highlight the Controls to Move</th>
<th>3. Drag onto Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Initial State" /></td>
<td><img src="image2" alt="Controls to Move" /></td>
<td><img src="image3" alt="Drag onto Parent" /></td>
</tr>
</tbody>
</table>

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**

- iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application
- iOS Tutorial: Using Location Sensors on the iOS Device
- Tutorial: Using FireMonkey Layouts
- FireMonkey Layouts Strategies
- Arranging FireMonkey Controls
- Gestures in FireMonkey
**iOS Tutorial: Taking and Sharing a Picture in an iOS Application**

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

- iOS Tutorial: Using a Button Component with Different Styles in an iOS Application

This tutorial covers the following typical tasks for using pictures in an iOS application:

<table>
<thead>
<tr>
<th>Taking a picture with the iOS device camera</th>
<th>Using a picture from the iOS device Photo Library</th>
<th>Sharing or printing a picture</th>
</tr>
</thead>
</table>

This functionality is provided as **Actions**, and you need to write only one line of code for each task.

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

**Actions** serve two purposes:

- An action can represent properties common to the user interface elements—such as whether a control is enabled or whether a check box is selected.
- An action can respond when a control fires—such as when the user clicks a button or chooses a menu item.
In this tutorial, you learn how to assign actions to user interface elements (such as a button) for each functionality that you want to support.

**BUILDING THE USER INTERFACE FOR THE APPLICATION**

The user interface of this sample application is quite simple, as shown in the following picture:

Place the following components on the Form Designer:

- **ToolBar** component
  - 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 `alClient`.
- **TActionList** component

**TAKING A PICTURE WITH THE iOS DEVICE CAMERA**

You can define an action to take a photo using the camera on the iOS device by using 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:

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

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

**USING A PICTURE FROM THE iOS 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:

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

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 an iOS app, 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 a 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 | ShowShareSheetAction**.
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
procedure TForm1.ShowShareSheetAction1BeforeExecute(Sender: TObject);
begin
  ShowShareSheetAction1.Bitmap.Assign(Image1.Bitmap);
end;
```

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:

SEE ALSO

- iOS Tutorial: Using Location Sensors on the iOS Device
- iOS Tutorial: Using the Notification Center on the iOS Device
- FireMonkey Actions
- FMX.StdCtrls.TButton
- FMX.Objects.TImage
- FMX.MediaLibrary
- [http://appleinsider.com/articles/12/02/16/share_sheets_twitter_integration_to_make_mountain_lion_more_social/](http://appleinsider.com/articles/12/02/16/share_sheets_twitter_integration_to_make_mountain_lion_more_social/)
**iOS Tutorial: Using Location Sensors on the iOS Device**

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

- iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application
- iOS Tutorial: Using the Web Browser Component in an iOS Application
- iOS Tutorial: Using Layout to Adjust Different Form Sizes or Orientations in an iOS Application

This tutorial describes the basic steps to locate your iOS 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 **alLeft** 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 Switch (Switch1) component to select on/off of TLocationSensor
  - a TListBoxGroupHeader with the text "Your Location"
  - a TListBoxItem with the name "ListBoxItemLatitude" and "Latitude" as text
  - a TListBoxItem with the name "ListBoxItemLongitude" and "Longitude" as text
  - a TListBoxGroupHeader with the text "Current Address"
  - a TListBoxItem with the name "ListBoxItemAdminArea" and "AdminArea" as text
  - a TListBoxItem with the name "ListBoxItemCountryCode" and "CountryCode" as text
  - a TListBoxItem with the name "ListBoxItemCountryName" and "CountryName" as text
  - a TListBoxItem with the name "ListBoxItemFeatureName" and "FeatureName" as text
  - a TListBoxItem with the name "ListBoxItemLocality" and "Locality" as text
• a TListBoxItem with the name "ListBoxItemPostalCode" and "PostalCode" as text
• a TListBoxItem with the name "ListBoxItemSubAdminArea" and "SubAdminArea" as text
• a TListBoxItem with the name "ListBoxItemSubLocality" and "SubLocality" as text
• a TListBoxItem with the name "ListBoxItemSubThoroughfare" and "SubThoroughfare" as text
• a TListBoxItem with the name "ListBoxItemThoroughfare" and "Thoroughfare" as text

- a TWebBrowser component (WebBrowser1) to show the Web Page (Google Maps). Set the Align property to alClient.

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 LOCATION SENSOR COMPONENT

First, 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.

Here is a code snippet to control TLocationSensor based on the change of value in the TSwitch component:

```pascal
procedure TForm44.Switch1Switch(Sender: TObject);
begin
  LocationSensor1.Active := Switch1.IsChecked;
end;
```
As discussed earlier, **TLocationSensor** fires an **OnLocationChanged** event when you move the iOS device. You can show the current location (Latitude and Longitude) using parameters with this event handler as follows:

```pascal
procedure TForm44.LocationSensor1LocationChanged(Sender: TObject;
  const OldLocation, NewLocation: TLocationCoord2D);
begin
  // Show current location
  ListBoxItemLatitude.ItemData.Detail := NewLocation.Latitude.ToString;
  ListBoxItemLongitude.ItemData.Detail := NewLocation.Longitude.ToString;
end;
```

**SHOW THE CURRENT LOCATION USING GOOGLE MAPS VIA A TWEBBROWSER COMPONENT**

As discussed in the [iOS Tutorial: Using the Web Browser Component in an iOS Application](#), the **TWebBrowser** component wraps a Web browser for iOS.

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

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

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

```pascal
procedure TForm44.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&output=embed',
    [NewLocation.Latitude.ToString, NewLocation.Longitude.ToString]);
  WebBrowser1.Navigate(URLString);
end;
```
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.

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: TGeocoder" in the private section of the form. You can also define an "OnGeocodeReverseEvent procedure" as in the following code snippet.

```pascal
type
  TForm44 = 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;
```
After you define these 2 lines, go to the line of the **OnGeocodeReverseEvent**, and type CTRL+SHIFT+C. This creates the following procedure in your code (that you will use later):

```pascal
procedure TForm44.OnGeocodeReverseEvent(const Address: TCivicAddress);
begin
end;
```

Now you can create an instance of **TGeocoder** and set it up with data with the following 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.

```pascal
procedure TForm44.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
        begin
        if Assigned(FGeocoder) then
        end;
        // Translate location to address
        if Assigned(FGeocoder) and not FGeocoder.Geocoding then
            FGeocoder.GeocodeReverse(NewLocation);
    end;
end;
```
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:

```plaintext
procedure TForm44.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;
```

SEE ALSO

- iOS Tutorial: Using Layout to Adjust Different Form Sizes or Orientations in an iOS Application
- iOS Tutorial: Using the Notification Center on the iOS Device
- System.Sensors.TGeocoder
- FMX.Sensors.TLocationSensor
iOS Tutorial: Using Notification Center on the iOS Device

This tutorial describes the basic steps to use the Notification Center on your iOS device.

Three Basic Notification or Alert Styles

When users set notifications for apps on their iOS 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.

Badge on Application Icon

Notification Banner on iPad

Notification Alert
**NOTIFICATION CENTER ON iPAD**

The following image shows the iPad Notification Center, where the user can pull down the list of all recent notifications:

![iPad Notification Center Image](image)

**ACCESS THE NOTIFICATION SERVICE**

The notification service interface (*IFMXNotificationCenter*) is defined as one of the FireMonkey Platform Services (*TPlatformServices*).

To access the notification service, do these two things:

- Add the following 2 units to the `uses` clause if they are not present:

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

- Run a query of the FireMonkey Platform Services using the following code:

  ```pascal
  var
  NotificationService: IFMXNotificationCenter;
  begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXNotificationCenter) then
    NotificationService := TPlatformServices.Current.GetPlatformService(IFMXNotificationCenter) as IFMXNotificationCenter;
  // Use the Notification Service
  end;
  ```

The *IFMXNotificationCenter* interface provides basic services to use the Icon Badge Number as well as the Notification.
SET THE ICON BADGE NUMBER FROM CODE

`IFMXNotificationCenter` has the `SetIconBadgeNumber` method to define the Icon Badge Number:

```pascal
procedure TForm1.SetIconBadgeNumber;
var
  NotificationService: IFMXNotificationCenter;
begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXNotificationCenter) then
    NotificationService := TPlatformServices.Current.GetPlatformService(IFMXNotificationCenter) as IFMXNotificationCenter;

  // Reset Icon Badge Number
  if Assigned(NotificationService) then
    NotificationService.SetIconBadgeNumber(18);
end;
```

After you set the Icon Badge Number to 18, you can see it on your iOS Home Screen:

![Icon Badge Number](image)

You can also reset the Icon Badge Number using the `ResetIconBadgeNumber` method:

```pascal
procedure TForm1.ResetIconBadgeNumber;
var
  NotificationService: IFMXNotificationCenter;
begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXNotificationCenter) then
    NotificationService := TPlatformServices.Current.GetPlatformService(IFMXNotificationCenter) as IFMXNotificationCenter;

  // Set Icon Badge Number
  if Assigned(NotificationService) then
    NotificationService.ResetIconBadgeNumber;
end;
```
**SCHEDULE NOTIFICATION**

You can also schedule Notification Messages using the `ScheduleNotification` method.

To show a Notification Message, you need to create an instance of the `TNotification` class, and then define the Name (Identifier) and the Message:

```pascal
procedure TForm1.ScheduleNotification;
var
  NotificationService: IFMXNotificationCenter;
  Notification: TNotification;
begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXNotificationCenter) then
    NotificationService := TPlatformServices.Current.GetPlatformService(IFMXNotificationCenter);
  if Assigned(NotificationService) then
    begin
      Notification := TNotification.Create;
      try
        Notification.Name := 'MyNotification';
        Notification.AlertBody := 'Delphi for iOS is here!';

        // Fired in 10 second
        Notification.FireDate := Now + EncodeTime(0, 0, 10, 0);

        // Send notification in Notification Center
        NotificationService.ScheduleNotification(Notification);
      finally
        Notification.Free;
      end;
    end;
end;
```

After you set the Notification Message, you can see it on top of your iOS Home Screen:
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 ScheduleNotification again with an instance of TNotification that has the same name (Name property).

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

```pascal
procedure TForm1.CancelNotification;
var
  NotificationService: IFMXNotificationCenter;
begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXNotificationCenter) then
    NotificationService := TPlatformServices.Current.GetPlatformService(IFMXNotificationCenter) as IFMXNotificationCenter;
  if Assigned(NotificationService) then
    NotificationService.CancelNotification('MyNotification');
end;
```
PRESENT THE NOTIFICATION MESSAGE IMMEDIATELY

You can also show the notification message immediately through `PresentNotification`.

To show a notification message, you need to create an instance of the `TNotification` class, and then define the Name (Identifier) and the Message:

```pascal
procedure TForm1.PresentNotification;
var
  NotificationService: IFMXNotificationCenter;
  Notification: TNotification;
begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXNotificationCenter) then
    NotificationService :=
      TPlatformServices.Current.GetPlatformService(IFMXNotificationCenter) as IFMXNotificationCenter;
  if Assigned(NotificationService) then
    begin
      Notification := TNotification.Create;
      try
        Notification.Name := 'MyNotification';
        Notification.AlertBody := 'Delphi for iOS is here!';

        // Set Icon Badge Number as well
        Notification.ApplicationIconBadgeNumber := 18;

        // Show Notification Message
        NotificationService.PresentNotification(Notification);
      finally
        Notification.Free;
      end;
    end;
end;
```
NOTIFICATION BANNER OR NOTIFICATION ALERT

By default, your application shows the notification banner:

- **Notification Banner on iPad**

- **Notification Alert**

To use a notification alert instead of a notification banner, the end user needs to change the notification Style through the Notification Center configuration page:
ADD ACTION TO THE NOTIFICATION ALERT

You can also customize an alert by adding an Action button.

To customize an Alert Action, you need to set the action to the `AlertAction` property, and then set the `HasAction` property to `True`, as follows:

```pascal
Notification := TNotification.Create;
try
  Notification.Name := 'MyNotification';
  Notification.AlertBody := 'Delphi for iOS is here!';
  Notification.AlertAction := 'Code Now!';
  Notification.HasAction := True;

  // Fired in 10 seconds
  Notification.FireDate := Now + EncodeTime(0,0,10,0);

  // Show Notification Message
  NotificationService.ScheduleNotification(Notification);
finally
  Notification.Free;
end;
```

SEE ALSO

- iOS Tutorial: Taking and Sharing a Picture in an iOS Application
- iOS Tutorial: Using Location Sensors on the iOS Device
- FMX.Notification.IFMXNotificationCenter
**iOS Tutorial: Using InterBase ToGo in an iOS Application**

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

- **iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application**

This tutorial describes the basic steps to browse data managed by InterBase ToGo on your iOS 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 iOS platform, dbExpress supports InterBase ToGo as well as SQLite. These database products can run on iOS 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 iOS devices.

Another tutorial discusses how to connect to Enterprise Database without using a client library on iOS device; see iOS Tutorial: Connecting to an Enterprise Database from an iOS Client Application.

**DESIGN AND SET UP THE USER INTERFACE**

This tutorial uses one **TListBox** component as the UI element.

**To set up a ListBox component, use the following steps:**

2. Drop a **TListBox** component on the form.
3. In the **Object Inspector**, set the following properties of the ListBox:
   - Set the **Align** property to **alClient**, so that the ListBox component uses the entire form.
   - Set the **DefaultItemStyles.ItemStyle** property to **listboxitembottomdetail**.
4. Right-click the **TListBox** component in either the Designer or the Editor, select **Add Item**, and add the following components:
   - **TListBoxHeader** component.
   - **TSearchBox** component.
5. Close the **Items Designer**.
6. Add a **TLabel** component to the **TListBoxHeader**, and set the following properties in the Object Inspector:
   - Set the **Align** property for the **TLabel** component to **alClient**.
   - Set the **StyleLookup** property to **toollabel**.
   - Set the **TextAlign** property to **taCenter**.
   - Set the **Text** property to **DB DEMO**.

---

Embarcadero Technologies
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\RAD Studio\11.0\Samples\Data\dbdemos.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 BindSourceDB1, and drag the mouse cursor to Item.Text in ListBox1.
   2. Click SPECIES_NAME in BindSourceDB1, and drag the mouse cursor to Item.Detail in ListBox1.
DEPLOYING YOUR APPLICATION TO iOS

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\RAD Studio\11.0\Samples\Data\dbdemos.gdb). On the iOS Device, the application is sand-boxed, and typically you can only read and write data that is located in the Documents folder under your application folder.

To connect to a local database on iOS, you need to perform the following actions:

- Deploy the database to the iOS Device.
- Change the configuration (to connect to the database file) to a local file under the Documents folder.

DEPLOY INTERBASE TOGO, DBEXPRESS DRIVER, AND THE DATABASE FILE TO iOS

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

- Interbase ToGo
- dbExpress Driver to InterBase
- The database file (dbdemos.gdb)

1. Open the Deployment Manager by selecting Project > Deployment.
2. Select All-Configurations - iOS Simulator 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
   - DBExpress InterBase Driver
5. Select **Add Files**, and select the database file (for example, C:\Users\Public\Documents\RAD Studio\11.0\Samples\Data\dbdemos.gdb).

6. Select **dbdemos.gdb**, and change **Remote Path** to StartUp\Documents\.

7. Select the **Platforms** column (double-click the ellipsis [...] in the row for dbdemos.gdb):
   1. Ensure that **iOS Simulator** and **iOS Device** 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).
   8. Select **All Configurations - iOS Device** platform, and make sure **dbdemos.gdb** is set to be deployed to StartUp\Documents\.

As you just configured, the database file (dbdemos.gdb) is to be deployed to the **Documents** folder in the sandbox area of your iOS app.
MODIFY YOUR CODE TO CONNECT TO A LOCAL DATABASE FILE ON iOS

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:

```pascal
procedure TForm1.SQLConnection1BeforeConnect(Sender: TObject);
begin
  {$IFDEF IOS}
  SQLConnection1.Params.Values['Database'] := GetHomePath + PathDelim + 'Documents' + PathDelim + 'dbdemos.gdb';
  {$ENDIF}
end;
```

The `GetHomePath` function gives you the actual home location of an iOS app. Using the `PathDelim` constant is recommended, as `PathDelim` specifically uses the path delimiter of the target platform (instead of hard-coded delimiters, such as \\ or /).
RUN YOUR APPLICATION ON THE iOS SIMULATOR OR AN iOS DEVICE

Now your application is ready to run. You should be able to browse data just as you can in the IDE. You can also narrow down the list using the Search Box as shown in the second image:

<table>
<thead>
<tr>
<th>Atlantic Spadefish</th>
<th>Atlantic Spadefish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaetodipterus faber</td>
<td>Chaetodipterus faber</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bat Ray</th>
<th>Blue Angelfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Myliobatis californica</td>
<td>Pomacanthus nauarchus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Blue Angelfish</th>
<th>Clown Triggerfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pomacanthus nauarchus</td>
<td>Ballistoides conspicillum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bluehead Wrasse</th>
<th>Firefish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thalassoma bifasciatum</td>
<td>Pterois volitans</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cabezon</th>
<th>Ornate Butterflyfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scorpaenichthys marmoratus</td>
<td>Chaetodon Ornatissimus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>California Moray</th>
<th>Redband Parrotfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gymnothorax mordax</td>
<td>Sparisoma Aurowrenatum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clown Triggerfish</th>
<th>Dog Snapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballistoides conspicillum</td>
<td>Lutjanus jocu</td>
</tr>
</tbody>
</table>

| Firefish | |
|----------| |
TROUBLESHOOTING

INTERBASE LICENSE ISSUES

If you get an error ("unavailable database") when you connect to the database in the development environment, this means you do not have a current license for InterBase.

- A license for InterBase Developer Edition is included as part of the product for some product editions.
- To activate the license of InterBase Developer Edition for a registered RAD Studio installation, go to the Embarcadero Product License Manager (click Start | All Programs | Embarcadero InterBase XE3).

EXCEPTION HANDLING ISSUES

If your application raises an exception without having proper exception handling code, your iOS 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:

```pascal
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;
```
## Typical Errors and Resolutions

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

<table>
<thead>
<tr>
<th>Error on iOS</th>
<th>Suggestion</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Error on iOS" /></td>
<td>Check whether the database file <em>(dbdemos.gdb)</em> is delivered to 'StartUp\Documents'.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Error on iOS" /></td>
<td>Check whether the license file is delivered for InterBase ToGo.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Error on iOS" /></td>
<td>Check whether you pointed to the local file (add an event handler for the <em>OnBeforeConnect</em> event of the <em>SQLConnection1</em> component).</td>
</tr>
</tbody>
</table>

### See Also

- InterBase ToGo with dbExpress
- iOS Tutorial: Using SQLite in an iOS Application
- iOS Tutorial: Connecting to an Enterprise Database from an iOS Client Application
**iOS Tutorial: Using SQLite in an iOS Application**

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

- iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application

This tutorial describes the basic steps to use SQLite as a local data storage on your iOS 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 iOS platform, dbExpress supports InterBase ToGo as well as SQLite. These database products can run on iOS 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 iOS devices.

Another tutorial discusses how to connect to Enterprise Database without using a client library on iOS device; see iOS Tutorial: Connecting to an Enterprise Database from an iOS Client Application.

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 Mobile Form Designer to design the user interface of your iOS 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.

5. Change the FailIfMissing property to False and then close the dialog:

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

6. Click the Test Connection button. With this operation, the new database file is created if no file existed:
CREATE TABLE ON DATAEXPLORER

1. On the Data Explorer, select the ShoppingList node under the SQLite section, and then select New Table from context menu.

![Data Explorer window with New Table option selected]

2. Specify a column titled ShopItem as value = TEXT.

![Table Name dialog box with ShopItem and TEXT selected]

3. Click the Save button and specify a table name (for example, Item.)
This tutorial uses one TListBox component as the UI element.

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

1. Create a FireMonkey Mobile application using **File > New > FireMonkey Mobile Application - Delphi**.
2. Drop a TToolBar on the form.
3. Drop a TButton on the ToolBar component.
4. Set the following properties in the **Object Inspector**:
   - Set the **Name** property to **ButtonAdd**.
   - Set the **StyleLookup** to **addtoolbuttonbordered**.
5. Drop a TButton on the ToolBar component.
6. Set the following properties in the **Object Inspector**:
   - Set the **Name** property to **ButtonDelete**.
   - Set the **StyleLookup** to **deletetoolbutton**.
   - Set the **Text** to **Delete**.
   - Set the **Visible** to **False**.
7. Drop a TLabel on the ToolBar component.
8. Set the following properties in the **Object Inspector**:
   - Set the **Align** to **alClient**.
   - Set the **StyleLookup** to **toollabel**.
   - Set the **Text** to **Shopping List**.
9. Drop a TListBox component on the form.
10. Set the following properties in the Object Inspector:
   - Set the TextAlign property to taCenter, so that the ListBox component uses the entire form.

CONNECTING TO THE DATA

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

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

   ![Data Explorer and Form Design](image)

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

2. Select the ShoppingList component on the form, and then change the Connected property to True.
3. Select the Item component on the form, and then change the Active property to True.
5. Select ShopItem in the Item component and drag ShopItem to 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.

```pascal
procedure TForm1.ListBox1ItemClick(const Sender: TCustomListBox; const Item: TListBoxItem);
begin
    ButtonDelete.Visible := ListBox1.Selected <> nil;
end;
```
CREATING THE EVENT HANDLER FOR THE ADD BUTTON TO ADD AN ENTRY TO THE LIST

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 ShoppingList.
   - 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 AddButton component. Add the following code to this event handler:

```pascal
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();
        Item.Refresh;
        ButtonDelete.Visible := ListBox1.Selected <> nil;
      end;
  except
    on e: Exception do
      begin
        ShowMessage(e.Message);
      end;
  end;
end;
```
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 **ShoppingList**.
   - Set the **SQL** property as follows:

   ```
   delete from Item where ShopItem = :ShopItem
   ```

   - Select the **Expand (...)** button on the **Params** property.
   - Select the **ShopItem** parameter and set **DataType** to **ftString**.
3. In the Form Designer, double-click the **DeleteButton** component. Add the following code to this event handler.

```delphi
procedure TForm1.ButtonDeleteClick(Sender: TObject);
var
  TaskName: String;
begin
  TaskName := ListBox1.Selected.Text;
  try
    SQLQueryDelete.ParamByName('ShopItem').AsString := TaskName;
    SQLQueryDelete.ExecSQL();
    Item.Refresh;
    ButtonDelete.Visible := ListBox1.Selected <> nil;
  except
    on e: Exception do
      begin
        ShowMessage(e.Message);
      end;
  end;
end;
```

MODIFYING YOUR CODE TO CONNECT TO A LOCAL DATABASE FILE ON iOS

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 iOS device unless you copy it to the iOS 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 iOS DEVICE**

1. In the Form Designer, select the **ShoppingList** component.
2. In the **Object Inspector**, double-click the **BeforeConnect** event.
3. Add the following code to this event handler:

```delphi
procedure TForm1.SQLConnectionSQLiteBeforeConnect(Sender: TObject);
begin
  {$IFDEF IOS}
  ShoppingList.Params.Values['Database'] := GetHomePath + PathDelim +
    'Documents' + PathDelim + 'shoplist.s3db';
  {$ENDIF}
end;
```

The **GetHomePath** function gives you the actual home location of an iOS app. Using the constant **System.SysUtils.PathDelim** is recommended, as **PathDelim** specifically uses the path delimiter of the target platform (instead of hard-coded delimiters, such as \ or /).

**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 **ShoppingList** component.
2. In the **Object Inspector**, double-click the **AfterConnect** event.
3. Add the following code to this event handler:

```delphi
procedure TForm1.ShoppingListAfterConnect(Sender: TObject);
begin
  ShoppingList.ExecuteDirect('CREATE TABLE IF NOT EXISTS Item (ShopItem TEXT NOT NULL)');
end;
```
RUNNING YOUR APPLICATION ON THE iOS SIMULATOR OR ON AN iOS DEVICE

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

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

SEE ALSO

- iOS Tutorial: Using InterBase ToGo in an iOS Application
- iOS Tutorial: Connecting to an Enterprise Database from an iOS Client Application
- SQLite support in XE3
iOS TUTORIAL: CONNECTING TO AN ENTERPRISE DATABASE FROM AN iOS CLIENT APPLICATION

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

- iOS Tutorial: Using ListBox Components to Display a Table View in an iOS Application
- iOS Tutorial: Using InterBase ToGo in an iOS Application

This tutorial describes how to connect to an Enterprise database from an iOS 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 iOS 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 iOS application can talk 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 iOS client.
CREATING THE MIDDLE TIER, A DATA SNAP 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 (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 DATA SNAP 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 in order to create a new Delphi project.
2. The **New DataSnap Server** wizard appears and you need to follow its steps without modifying too many parameters.

3. In the **New DataSnap Server** wizard:
   1. At first step, choose **VCL Forms Application** as application type.
   2. At the second step, choose the **TCP/IP** protocol, **Server Methods Class** and **Sample Methods** from the Server Features list.
   3. At the third 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.
   4. At the final step (number four) select **TDSServerModule** as the ancestor for the Server Methods.

4. Save the form unit as **DataSnapServerUnit.pas**.
5. Switch to **DataSnapServerUnit**, and change the **Name** property of the Form to **DSServerForm**.
6. Save the server methods unit (by default as created by the Wizard: **ServerMethodsUnit1**) as **ServerModuleUnit.pas**.
7. Save the server container unit (by default as created by the Wizard: **ServerContainerUnit1**) as **ServerContainerUnit.pas**.
8. Save the new project as **DataSnapServerProject.droj**.
9. Select ProjectGroup1 in the **Project Manager** and save the project as **DataSnapTutorialProjectGroup.groupproj**.
DEFINE A DATASET ON THE DATASNAP SERVER

1. Switch to the ServerContainerUnit.pas file and replace the uses clause in the implementation with: uses Winapi.Windows, ServerModuleUnit;
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:
     - Set the Name property to SQLConnection_EMPLOYEE.
     - Set the LoginPrompt property to False.
     - Set Driver to InterBase.
     - Expand the Driver node, and set the DataBase property to C:\Users\Public\Documents\RAD Studio\11.0\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:
     - Set the Name property to SQLDataSet_EMPLOYEE.
     - Set the SQLConnection property to SQLConnection_EMPLOYEE.

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.
- Expand the Driver node, and set the DataBase property to C:\Users\Public\Documents\RAD Studio\11.0\Samples\Data\EMPLOYEE.GDB.
- Change the Connected property to True. If you get an error, double-check the Driver 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:

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

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

**RUN THE DATA SNAP 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 AN iOS APPLICATION THAT CONNECTS TO THE DATA SNAP SERVER

The next step is creating the iOS 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.droj.
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)

  **TDSProviderConnection** component provides connectivity to the DataSnap server using dbExpress.

  - Set the **SQLConnection** property to **SQLConnection1**.
  - Set **ServerClassName** to **TDSServerModule_EMPLOYEE**. This name needs to match the name of the class of the Server Module of the DataSnap server.
  - Set the **Connected** property to **True**.

- **TClientDataSet** component (ClientDataSet1)

  **TClientDataSet** implements a database-independent dataset, and this can be used as a local in-memory buffer of the records from another dataset.

  - Set the **RemoteServer** property to **DSProviderConnection1**.
  - Set the **ProviderName** property to **DataSetProvider_EMPLOYEE**. This name needs to match the name of the provider for the DataSnap server.
  - Set the **Active** property to **True**.

- **TListBox** component

  - Set the **Align** property to **alClient**.
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:

   ![LiveBindings Designer](image1.png)

   2. Now you have created and configured the DataSnap Client on iOS. You should be able to see the data coming from the DataSnap server in the IDE:

   ![DataSnap Client on iOS](image2.png)
DEPLOY THE MIDAS LIBRARY TO iOS SIMULATOR

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

- MIDAS Library

1. Open the Deployment Manager by selecting Project > Deployment.
2. Select Add Featured Files ( ):

3. Select the following module, and then click OK to close the Deployment Manager:
   - MIDAS Library
RUN YOUR APPLICATION ON THE iOS SIMULATOR, OR ON AN iOS DEVICE

Now your application is ready to run.

In the Project Manager, select either the iOS Simulator or the iOS Device target platform, and run your application. You should be able to browse data just as you do within the IDE.

SEE ALSO

- iOS Tutorial: Using InterBase ToGo in an iOS Application
- iOS Tutorial: Using SQLite in an iOS Application
- Developing DataSnap Applications
- Understanding Multi-tiered Database Applications
- Datasnap.DSServer.TDSServer