Delphi
程式開發手冊
embarcadero

序

本書『Delphi 開發手冊』的目的是說明如何學習使用 Delphi 來開發 iOS 和 Android 的 App。本書的內容並不是說明 iOS/Android 專業程式設計，而是展示如何使用 Delphi 整合發展環境來開發 FireMonkey iOS/Android App。

本書分成 2 大部份，第一部份將說明如何使用 Delphi 開發 iOS 設備上的 App，這部份也將說明如何使用 Delphi 的 IDE。第 2 部份則是說明如何使用 Delphi 開發 Android 的 App，由於在第 1 部份已經說明了 IDE 的內容，因此第 2 部份將只著重在說明 Android 的開發。

本書將討論如下的內容:

版權所有 請勿翻印

- 第 1 部份
  - 如何安裝和設定 Delphi for iOS 整合發展環境
  - 使用 Delphi for iOS 整合發展環境開發您的第一個 iOS App
  - 使用 Delphi for iOS 整合發展環境的功能
  - 除錯您的 iOS App
  - 測試您的 iOS App
  - 部署您的 iOS App
  - 開發一個有趣的 iOS App 吧

- 第 2 部份
  - 如何安裝和設定 Delphi for Android 整合發展環境

- 開發 Delphi for Android App

- 第 3 部份
  - 如何安裝和設定 Linux
  - 連結 Delphi IDE 和 Linux 環境

在閱讀完本書的內容之後您就可以使用 Delphi 來開發您的 iOS/Android App 了，之後您就可以閱讀更深入的 iOS/Android 程式設計書籍並且使用 Delphi 做為學習和開發的工具。

由於 Delphi 支持移動平臺開發之後每年至少有 2 次的大版本更新以及數個 Patch 和 Hotfix 版本，為了讓本書能夠跟上這麼頻繁的更新，因此從本版開始將在本書最後部份陸續加入新版的功能說明，本書前面大部份的內容由於是基本功能因此將只進行維護的修正。

在本書隨後的內容中將以 Delphi for iOS 來代表 Delphi 中開發 iOS App 的功能，並以 Delphi for Android 代表 Delphi 中開發 Android App 的功能。

版權所有 請勿翻印

本書的範例程式請至下列網頁下載:

http://embarcadero.qcomgroup.com.tw/download/dpg.zip

目錄

1.安裝和設定 Delphi for iOS 11

1-1 安裝 PAServer 12

1-2 Delphi for iOS 整合發展環境中建立組態 15

2. 進入 Delphi for iOS 整合發展環境 18

2-1 如何建立 iOS App 新專案 20

2-2 Delphi for iOS 專案組成元素 23

3 開發您的第 1 個 iOS App 28

3-1 使用 MDD 設定 35

4. 使用 Delphi for iOS 整合發展環境 37

4-1 移動程式碼區塊 48

4-2 儲存/切換桌面設定 49

4-3 原始碼格式化 52

4-4 SyncEdit 63

4-5 程式碼重構(Refactor) 65

4-6 待辦清單(To-Do List) 68

4-7 程式區塊批註 72

完成類別功能(Class Completion) — CTRL+SHIFT+C 72

版權所有 請勿翻印

4-8 錯誤洞察(Error Insight) 73

4-9 宣告類別變數(Declare Field) 73

程式碼流覽 75

4- 10 設定和使用書簽 75

4-11 歷程管理員(History) 77

4-12 使用 LiveBinding 系結資料和視覺化元件 79

4-13 實作 SearchEditButton1Click 事件處理函式 81

5 除錯您的 iOS App 82

6. 為您的 iOS App 進行單元測試 89

進行單元測試 - 建立測試案例 92

7 開發和分發 iOS App 到 iOS 設備中 98

確定安裝了 XCode 的命令列工具 99

建立 iOS 設備的遠端組態 100

開發 iPad Mini 範例 App 104

取得 Apple 認證 108

使用部署管理員分發您的 iOS App 119

8 分發複雜的 iOS App 121

9 使用 iOS 的服務 125

10 分發額外檔案的 iOS App 130

11 用 Delphi RIO 寫個遊戲吧 135

11-1 讓泡泡充滿畫面吧 137

11-2 點選捏破泡泡 141

11-3 加入手勢功能 143

版權所有 請勿翻印

11-4 重新開始遊戲 152

11-5 處理遊戲資訊 153

11-6 把遊戲資訊儲存到資料庫吧 160

11-7 分享遊戲的樂趣吧 172

12.安裝和設定 Delphi for Android 開發環境 183

13.開發 Delphi for Android App 188

13-1 開發查詢臺北市旅館 App 190

13-2 加入聲控查詢功能 197

註冊 Delphi 程式碼中的回叫函式 200

藉由 JActivity 啟動 Google 的語音辨識功能 201

從回叫函式中取出結果並且使用它進行搜尋 203

13-3 啟動流覽器查詢旅館網站主頁 207

13-4 在雲端為旅館寫評價 209

13-5 寫個可啟動所有範例 App 的 App 吧 217

13-6 為您的 App 設計個圖像吧 220

13-7 為您的 App 加入啟動畫面吧 222

13-8 為您的 App 加入 Provisioning 資訊吧 224

設定 Build Configurations 為 Release 225

開啟 Target Platforms 設定為 Android 平臺並在 Configuration 中選擇 Application Store 225

到 Project > Options > Provisioning 頁面填寫必要的資訊 226

維護 App 版本資訊 228

14 RIO 新功能 229

版權所有 請勿翻印

14-1 MultiView 229

14-2 使用分散式版本控制工具-Git....................................... 231

14-3 使用 DUNITX 單元測試框架 249

14-3-1 DUNITX 單元測試框架簡介 250

14-3-2 如何成為 DUNITX 測試框架的測試類別 251

規則 1 任何的 Delphi 類別都可以成為測試類別 251

規則 2 撰寫測試方法 252

規則 3 如何使用 Test Fixture 252

規則 4 資料驅動測試 254

規則 5 使用 DUnitX.Assert.Assert 類別測試執行結果....... 255

14-3-3 使用 DUNITX 單元測試框架 265
版權所有 請勿翻印
14-4 App Tethering 267

14-4-1 手機端 TetherDBClient 如何工作 268

偵測和連結.. 269

14-4-2 Windows 端 TetherDatabase 如何工作 271

14-4-3 Windows 端和手機如何共同工作 273

TetherDBClient 專案端... 273

TetherDatabase 專案端 274

TetherDBClient 專案端... 275

TetherDatabase 專案端 275

14-5 藍牙開發 276

使用 TBlueTooth 元件.. 277

14-6 10.3 版 Delphi 程式語言新功能 294

14-6-1 inline 變數宣告 294

14-6-2 型態推論 296

16-4-3 其他語言新功能.. 297

15 呼叫 Android 系統功能 298

15-1 呼叫 Java 類別程式碼 299

步驟 1 把 Java 類別宣告轉成 Delphi 類別宣告 300

步驟 2 編譯並直接連結到最後的 App 中........................... 301

步驟 3 加入 Java 的.jar 檔並部署 302

15-2 呼叫 Delphi 尚未封裝的 Java 類別.............................. 305

使用 JObjectClass 介面封裝 Toast 類別內容.................... 308

使用 JObject 介面封裝 Toast 物件內容 309

使用 TJavaGenericImport 類別封裝完整 Java 類別.......... 310

使用 TJToast 類別 310

15-3 呼叫 Java API 存取 Android 連絡人資訊.................... 312

15-4 使用 Google GCM 實作推播功能................................ 324

15-4-1 啟動使用 GCM 功能 326

15-4-2 開發 GCM 用戶端 App 330

15-4-3 建立 Multi-Device 專案 330

15-4-4 開發 DataSnap 伺服器.................................... 340

15-4-5 向 DataSnap 伺服器註冊設備 ID 344

15-4-6 開發 Windows 用戶端...................................... 348

16 物聯網開發 350

16-1 什麼是 Beancon 技術 350

版權所有 請勿翻印

16-2 Beancon 種類 351

16-3 Beancon 資料格式和意義... 352

16-4 Beancon 接近狀態 354

16-5 Beancon 設備校正 354

16-6 使用 Apple iBeancon 355

16-7 開發 Beacon App 和物聯網應用架構............................ 356

16-7-1 自動偵測 Beacon 設備 357

16-7-2 開發已知 Beacon 設備 App 362

17 開發有趣的物聯網應用架構.. 367

17-1 範例資料表 368

17-2 仲介 DataSnap 伺服器... 369
版權所有 請勿翻印
17-3 移動用戶端 371

17-4 執行 DataSnap 伺服器和用戶端 App 372

18 開發 Android Service App 374

18-1 Android 服務種類 374

18-2 服務生命週期 375

18-3 App 生命週期 376

18-4 服務類別 377

18-5 開發服務流程 378

本機服務開發流程 378

遠端服務開發流程 379

18-6 使用 Delphi 開發 Android 服務 379

支援開發 Android 服務的 Delphi 類別和元件.................... 379

本機服務 App 381

19 新的 JSON 類別庫(XE11) 395

19-1 JSON Reader/Writer 類別 396

19-1-1 Writer 類別 397

19-1-2 Reader 類別 401

19-2 JSON Builder 類別 406

TJSONObjectBuilder 類別 407

TJSONArrayBuilder 類別 409

TJSONIterator 類別 411

19-3 BSON 類別 414

19-3-1 TBsonWriter 414

19-3-2 TBsonReader 類別 419

20 新的 AddressBook 組件 419

21 安裝和設定 Linux 開發環境 426

21-1 安裝 Linux 作業系統 427

21-2 傳遞 LinuxPAServer19.0.tar 到 Linux 429

21-3 設定 Delphi IDE 連結 Linux 作業系統 431

21-4 開發 Delphi Linux 應用程式並部署到 Linux 433

版權所有 請勿翻印

Delphi for iOS入門手冊

本手冊的目的是說明 Delphi for iOS 的入門使用者快速學習和瞭解如何使用 Delphi for iOS 整合發展環境來開發手機 App，本手冊將藉由開發一個小型手機 App 來說明 Delphi for iOS 整合發展環境中經常會被使用的功能，讓 Delphi for iOS 的新使用者能夠在閱讀本手冊之後能夠立刻開始使用 Delphi for iOS 整合發展環境來開發實際的手機 App。

1.安裝和設定 Delphi for iOS

版權所有 請勿翻印

在使用 Delphi for iOS 開發 iOS App 之前，讀者必須正確的安裝和設定相關的環境和輔助軟體 RAD PAServer，如此一來讀者才能夠在 Delphi for iOS 整合發展環境中除錯 iOS App 和部署 iOS App。

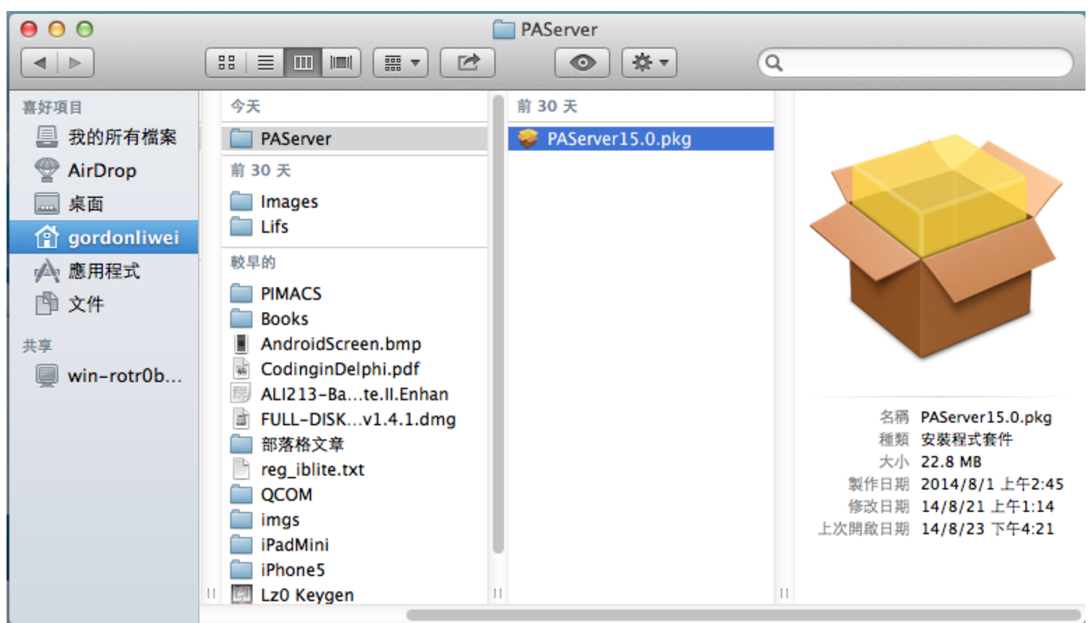
本書使用的開發環境是在 MacBook Pro 的機器中執行 OSX 10.8.2，XCode 4.6 以及 iOS SDK 6.1。並且使用 VMWare Fusion 5.0.2 執行 Windows 7 64 位元的繁體作業系統。

要完成 Delphi for iOS 的安裝，讀者需要完成下面的步驟:

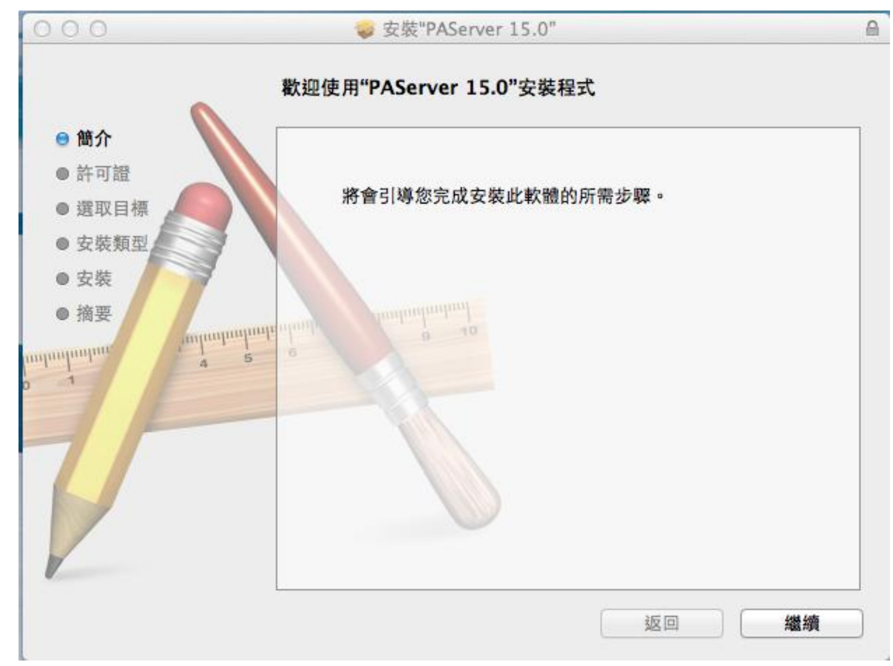
1. 安裝 Delphi for iOS 整合發展環境
2. 在 Mac OS 中安裝 Platform Assistant Server(RAD PAServer)軟體以便除錯部署和分發 iOS App
3. 在 Delphi for iOS 整合發展環境中建立相關的組態，例如建立 iOS 模擬器組態以便除錯 iOS App，建立 iOS Device 組態以便部署和分發 iOS App 到 iPhone 或是 iPad/iPad Mini 設備中。

1-1 安裝 PAServer

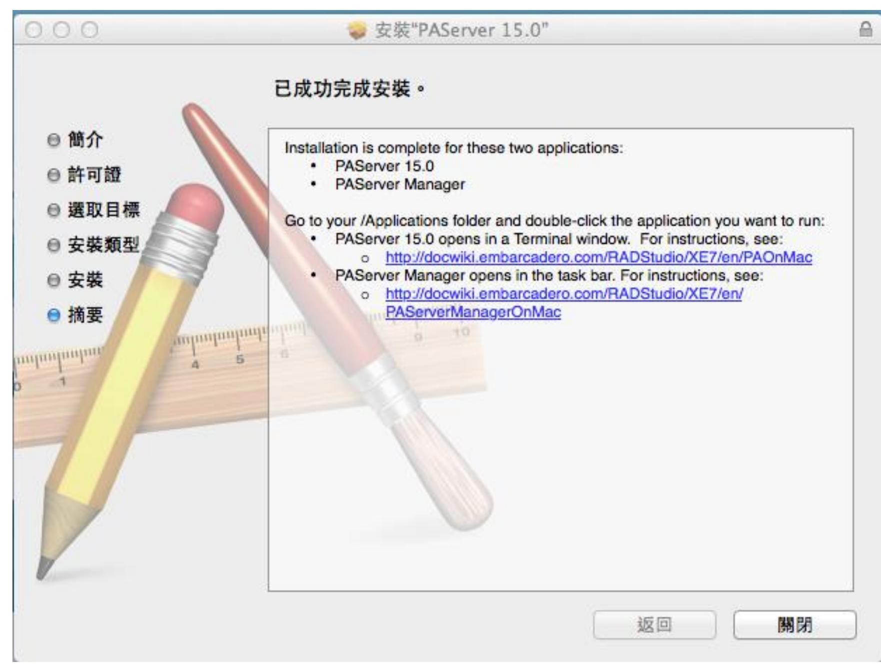
請使用 Delphi for iOS 的安裝 DVD 安裝 Delphi for iOS 整合發展環境，在安裝安之後，於 Delphi for iOS 的安裝目錄下會有一個『PAServer』子目錄，其中的『PAServer15.0.pkg』檔案即包含需要安裝在 Mac OS 主作業系統的 PAServer 應用程式，請把此檔案拷貝到 Mac OS 中。例如下圖顯示筆者把『PAServer15.0.pkg』拷貝到 Mac 的 SharediOS\PAServer 目錄下:



在 Mac 中按兩下『PAServer15.0.pkg』便會開發安裝 PAServer:



『RADPAServerRIO.pkg』執行完畢之後會把 PAServer 安裝在『/Applications』目錄中。

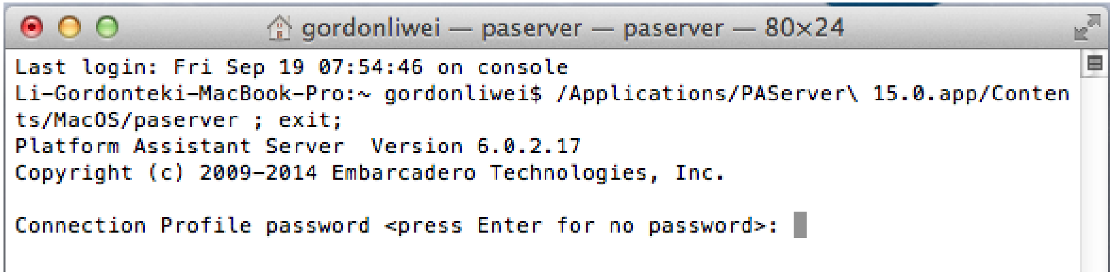


安裝完 PAServer 後讓我們執行它以便稍後進行下一個步驟。請到 Mac 的應用程式群組中執行 RAD PAServer RIO，如下所示:

版權所有 請勿翻印

或是在 Mac 中點選螢幕下方的 RAD PAServer RIO 圖像:

執行 PAServer 後，PAServer 會向讀者要求輸入一個連結的密碼，讀者可以自行給予密碼或是直接按下 Return 鍵不使用連結密碼，如下所示:



接著 PAServer 就會在執行狀態，例如下圖就顯示了 PAServer 執行在 64211 通信埠，等待連結:

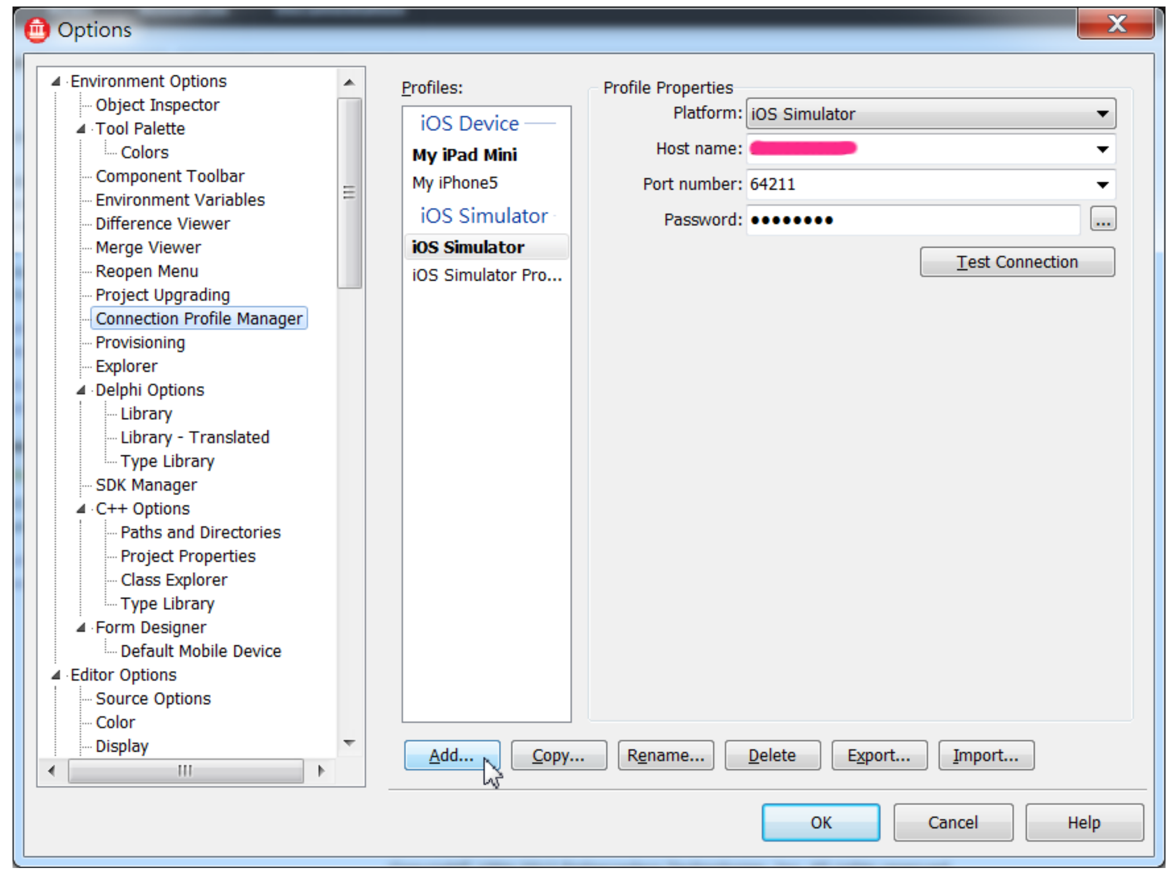
版權所有 請勿翻印

本書在撰寫時是使用 Delphi RIO 的 Beta 版，因此使用的 PAServer 是早期的版本，讀者使用的 PAServer 應該比本書的 PAServer 版本來得更高

1-2 Delphi for iOS 整合發展環境中建立組態

當開發人員在 Delphi for iOS 中建立 iOS App 專案後，需要指定如何部署和分發此專案的組態。例如如果您需要除錯 iOS App 專案，那麼 Delphi for iOS 整合發展環境必須把此 iOS App 分發到 Mac OS 中 XCode 的 iOS 模擬器中執行和除錯，因此 Delphi for iOS 整合發展環境必須知道您的 Mac OS 的相關資訊才能夠正確的把 iOS App 部署到 iOS 模擬器中，因此這就需要開發人員事先設定好組態資訊並且把它指定給 iOS App 專案。

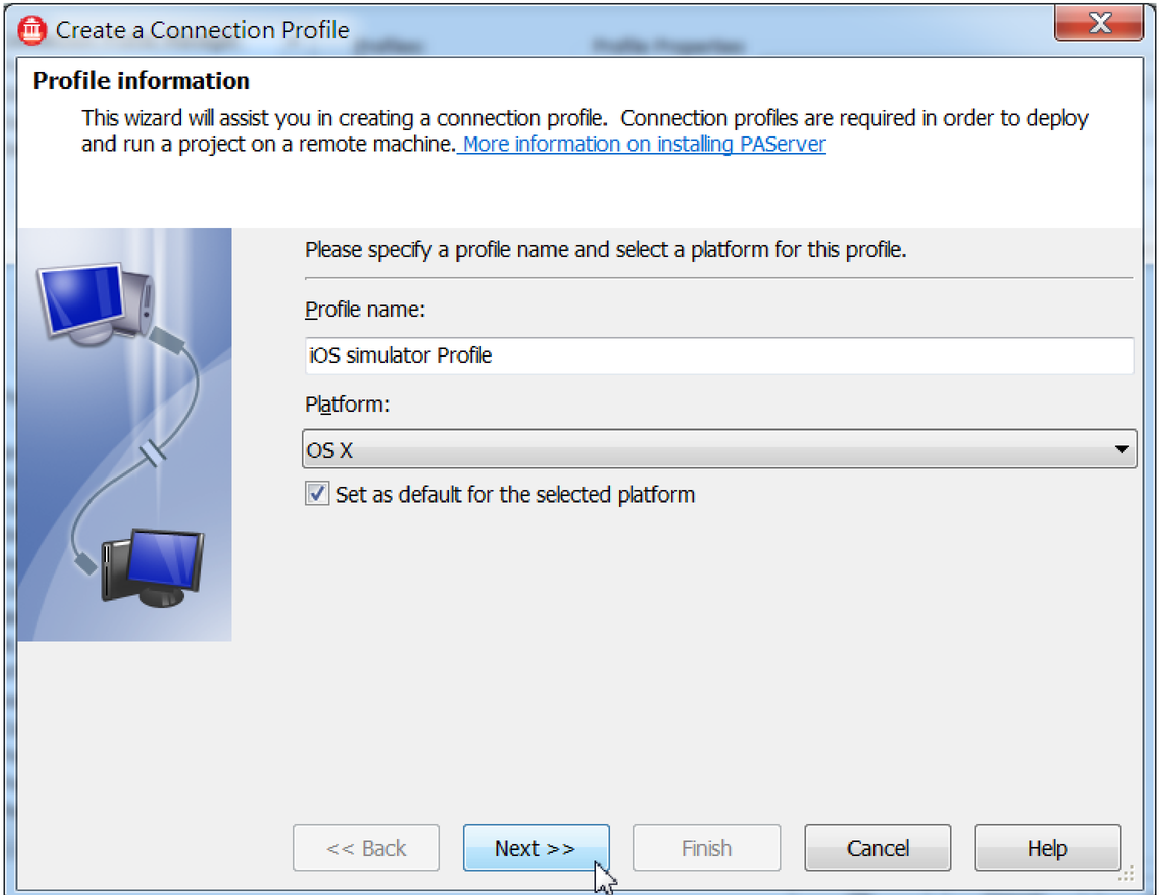
要建立組態資訊，請執行 Delphi for iOS，在整合發展環境中點選上方的 Tools | Option 功能表，在 Options 對話盒中的 Connection Profile Manager 專案中點選左下方的『Add』按鈕以新建組態，如下所示:



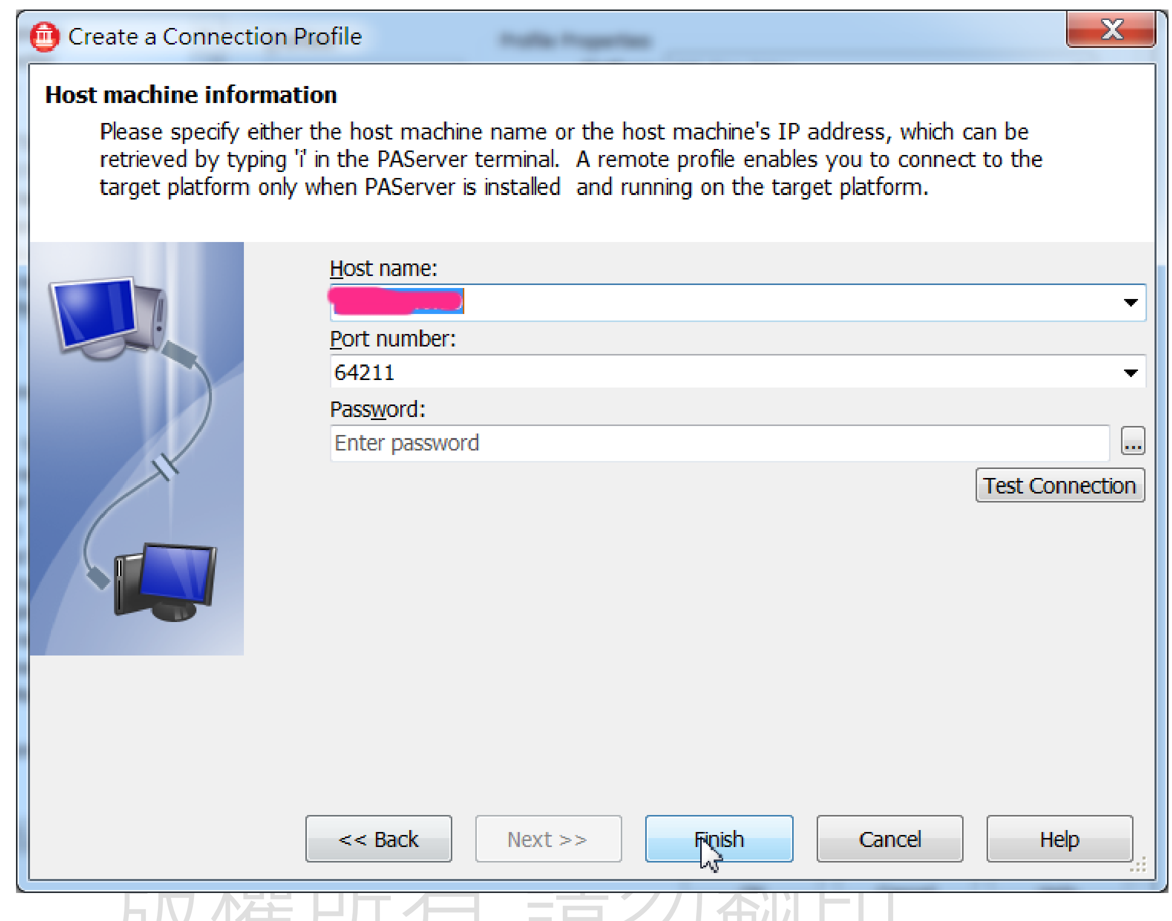
在點選『Add』按鈕後 IDE 會顯示 Create a Remote Profile 對話盒，您需要在此對話盒中輸入您需要在 Profile name 欄位中自行輸入此組態的名稱，接著在 Platform 欄位中選擇此組態需要部署的平臺，例如在下圖中筆者為此組態取名為『iOS Simulator Profile』並且選擇部署到 iOS 模擬器中以準備開發和測試 iOS App。在 Platform 欄位的下拉盒中還有其他的平臺，例如 iOS Device 平臺，讀者可以根據需要選擇欲使用的平臺。目前的範例中讓我們使用

『OS X』平臺，並且勾選下方的『Set as default for the selected platform』勾選盒以設定它為內定使用的組態。

接著點選『Next』按鈕以進入下一步:



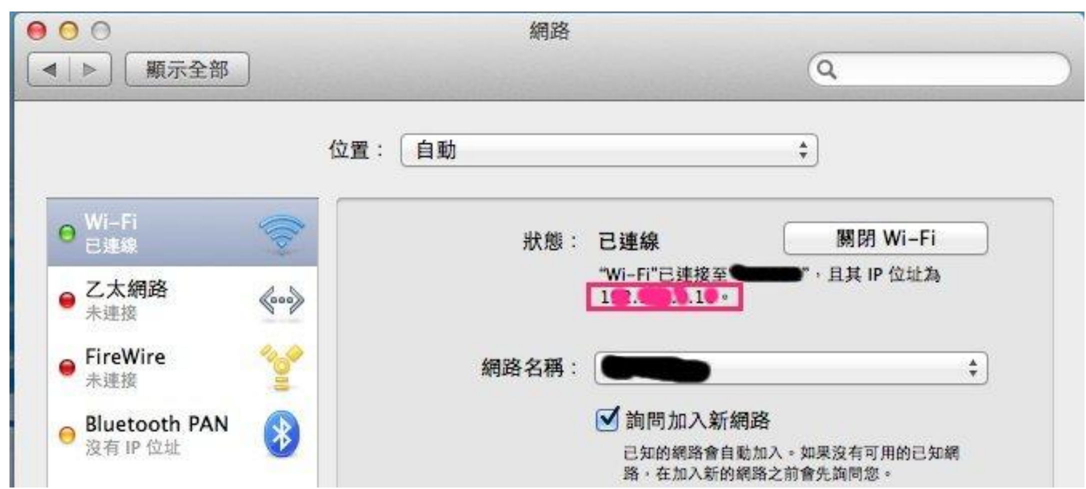
接下來您需要設定此組態的 Mac OS 的主機名稱或是 IP 位址，執行在 Mac OS 中 PAServer 使用的通信埠以及 PAServer 使用的連結密碼，如下所示:



版權所有 請勿翻印

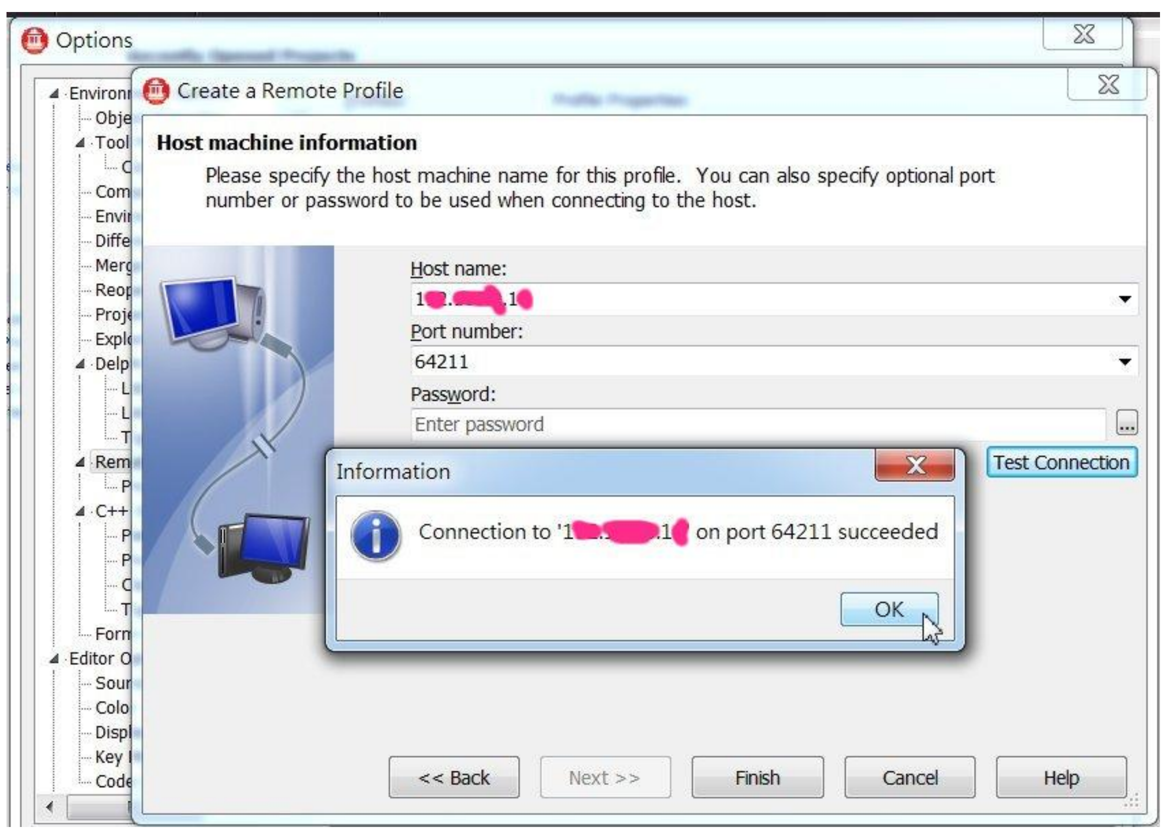
要取得您 Mac OS 的 IP 位址，您可以在您的 Mac OS 中點選螢幕左上方的蘋果圖像，選擇其中的『系統偏好設定』，再點選其中的『網路』圖像，接著 Mac OS 就會顯示如下的網路對話盒，其中就會顯示 IP 位址，例如筆者的 Mac IP 是『1xx.1xx.0.10』，因此在上圖的 Host Name 欄位中就輸入了此 IP。

此外由於在 1-1 小節中安裝 PAServer 時 PAServer 使用的通信埠是『64211』而且我們沒有設定連結密碼，因此在上圖中 Port number 欄位中輸入了 64211，Password 欄位則沒有輸入密碼。



在輸入了這些資訊後，請確定 PAServer 已經執行在 Mac OS 中，那麼您就可以點選『Create a Remote Profile』對話盒中的『Test Connection』按鈕來測試 Delphi for iOS IDE 是否能夠連結到 Mac OS 中的 PAServer。如果讀者設定的資訊都是正確的話，那麼應該會看到類似如下的結果畫面，IDE 顯示成功的連結您的 Mac OS 中的 PAServer，這代表稍後您就可以正確的部署您的 iOS App 到 iOS 的模擬中測試，除錯和執行了。

最後請點選『Finish』按鈕以完成建立組態的工作。



現在我們就可以準備進入 Delphi for iOS 整合發展環境來開發 iOS App 了。

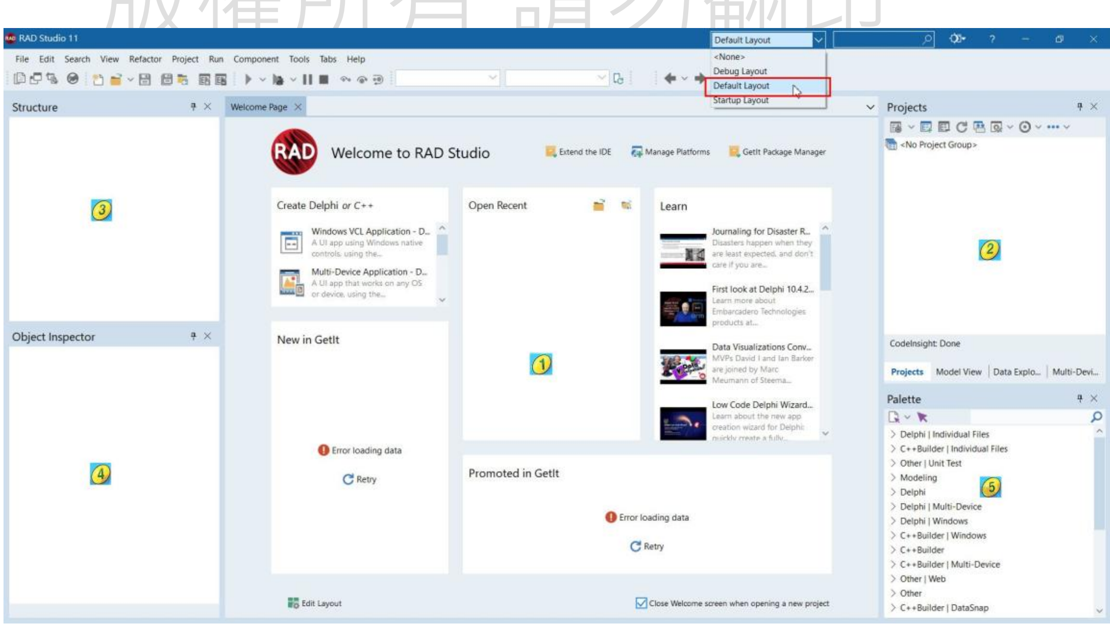
2. 進入 Delphi for iOS 整合發展環境

當您使用 Delphi for iOS DVD 安裝完之後，您可以藉由點選如下圖的 Delphi for iOS 圖像開始執行 Delphi for iOS 整合發展環境:

點選 Embarcadero RAD Studio RIO 或是 Delphi RIO 圖像執行 Delphi for iOS

一進入 Delphi for iOS 整合發展環境您看到的 IDE 是所謂的歡迎設定，請先點選加上方的下拉盒中並選擇切換到 Default Layout，Default Layout 是在開發過程中最常使用的設定，您會看到類似如下的畫面，Delphi for iOS 整合發展環境除了菜單和工具列之外(稍後說明)，整個整合發展環境分成 5 大區域，在下圖中我們以數字來標明這 5 大區域:

版權所有 請勿翻印



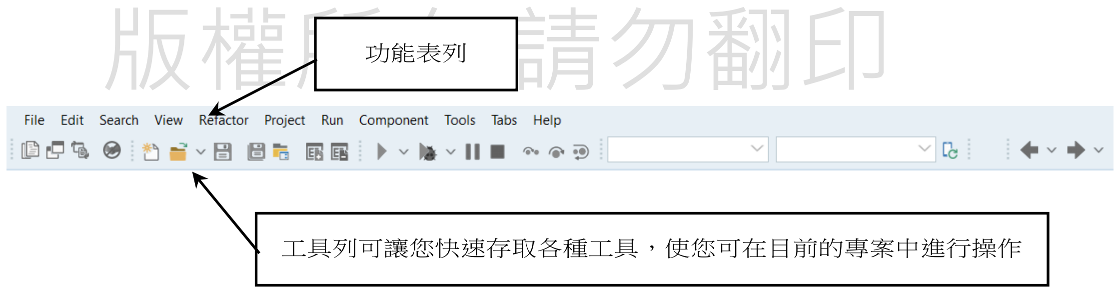
Delphi for iOS 整合發展環境進入畫面

下面的表格說明了這成 5 大區域的功能:

區域編號	說明
1	歡迎畫面(Welcome Page)，其中的內容會根據開發人員的設定來顯示，在內定上會顯示開發人員最近開啟和使用的專案。此外開發人員也可以在其中建立『最愛』，並且把專案歸類在不同的『最愛』中。
2	專案管理員(Project Manager)，管理專案中所有的檔案，由於在一開始沒有建立或是開啟任何的專案，因此它的內容暫時是空白的
3	專案樹狀架構(Structure)，可顯示目前開啟的表單(Form)中元件的樹狀架構，由於在一開始沒有建立或是開啟任何的表單，因此它的內容暫時是空白的
4	物件檢視器(Object Inspector)，在專案設計時期可檢視，設定組件的特性值，由於在一開始沒有建立或是開啟任何的表單和使用任何的元件，因此它的內容暫時是空白的
5	工具盤(Tool Palette)，列出目前可使用的工具，在一開始它的內容列出了目前在整合發展環境中可建立的專案種類。如果您建立了專案之後，它的內容就會改變，稍後我們會看到如果在專案中設計表格，那麼它的內容就會改變為表格可使用的元件

此外，在整合發展環境上方提供了工具列可幫助您快速完成特定的工作:

版權 請勿翻印



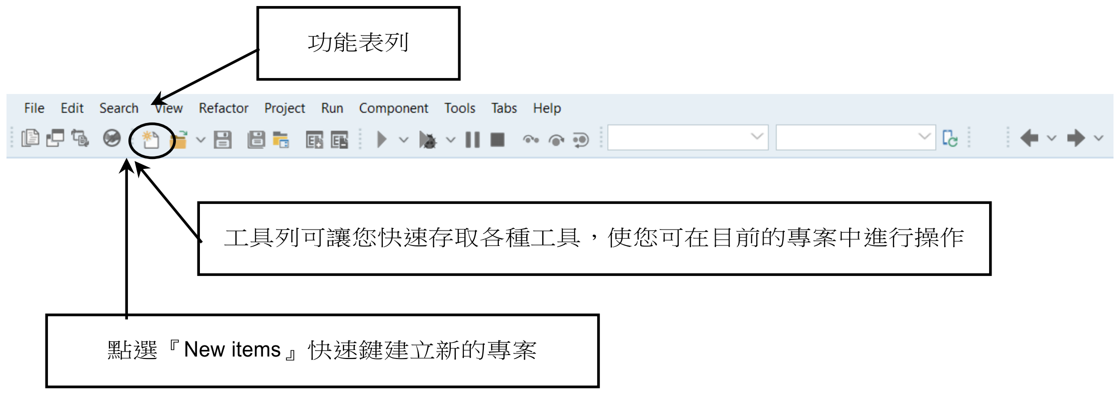
在稍後的教學內容會說明工具列中不同按鈕的功能，現在就讓我們開始建立一個新的 Delphi for iOS 項目。

2. 建立新專案

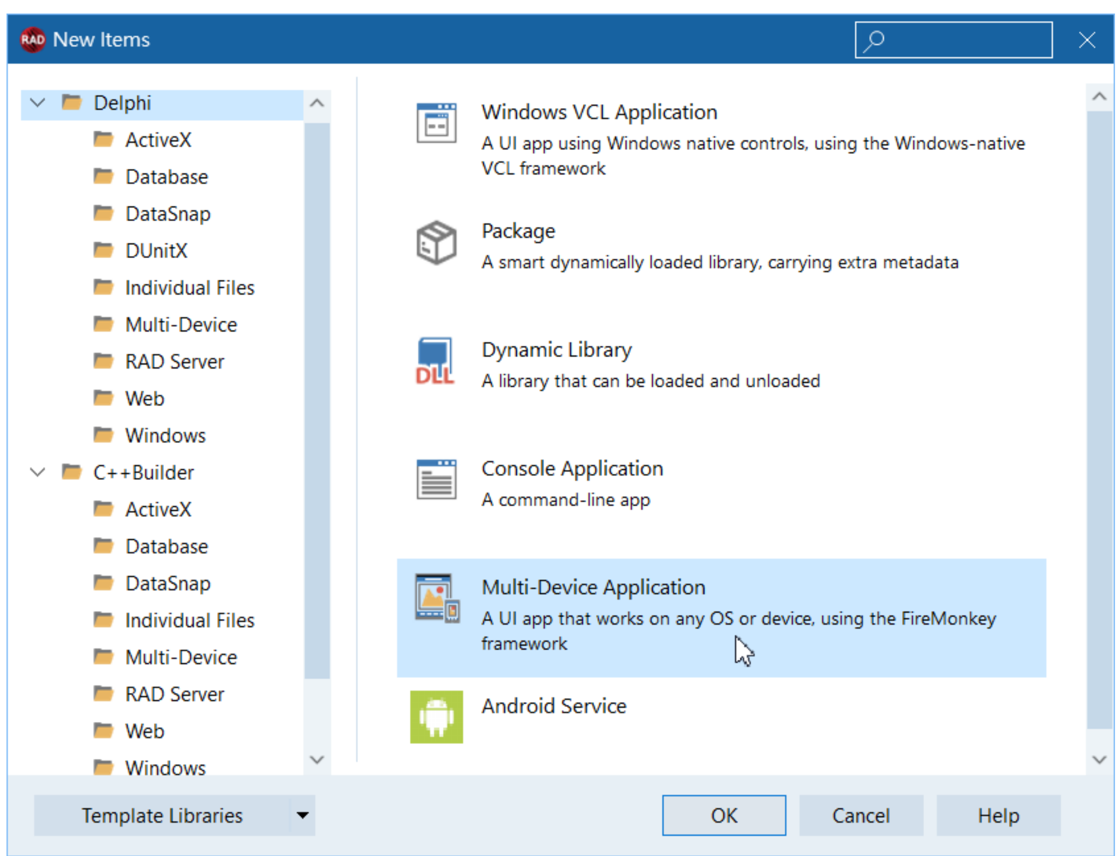
要使用 Delphi for iOS 開發新的 iOS App，您必須先從建立 iOS App 專案開始:

2-1 如何建立 iOS App 新專案

在 Delphi for iOS 整合發展環境中有許多方法可以建立新項目，首先您可以點選工具列中的『New items』快速鍵，它位於工具列從左方開始的第 4 個快速鍵:

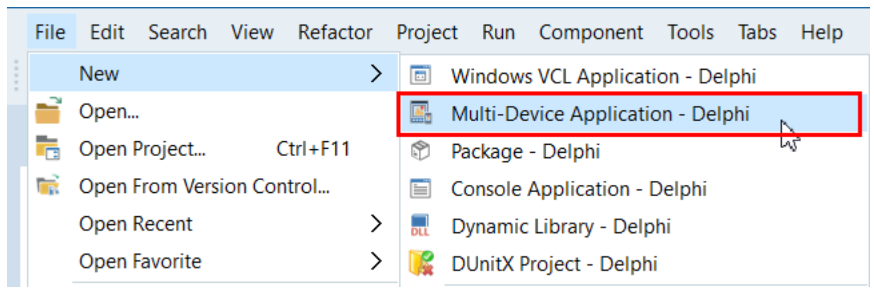


點選了工具列中的『New items』快速鍵之後，整合發展環境會顯示如下的 New Items 對話盒，讓您從其中選擇您想建立的專案型態，請在 Delphi Projects 專案中選擇『Multi-Device Application』圖像以建立 iOS App 專案，如下圖所示:



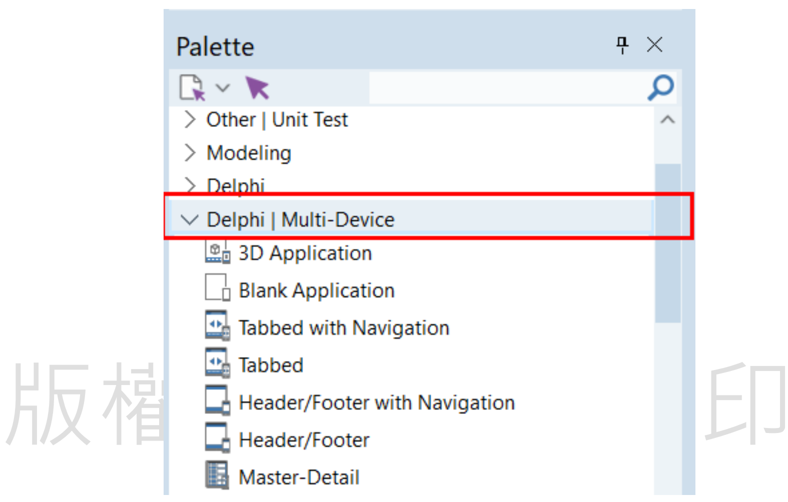
在 New Items 對話盒中選擇要建立的專案型態

第二種建立項目的方法是藉由整合發展環境上方的功能表，您可以藉由點選功能表中的 File | New 選項來選擇要建立的項目型態，如下所示:



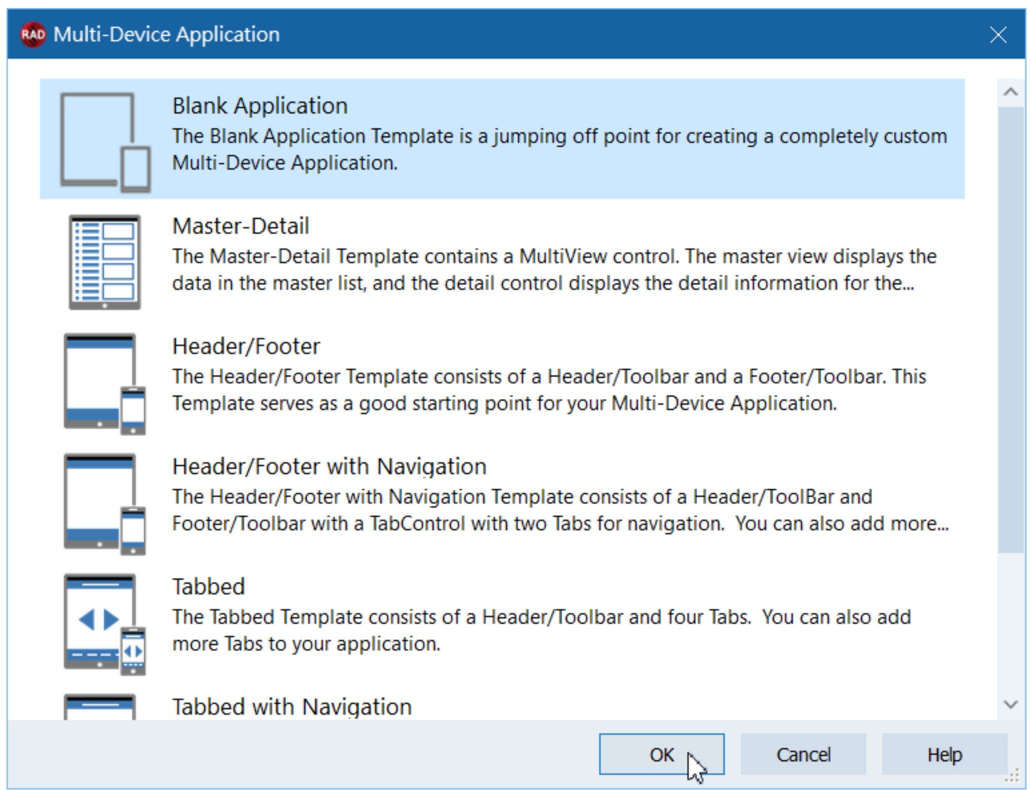
選擇功能表中的 File|New 選項並且選擇要建立的項目型態

第三種建立專案的方法是點選整合發展環境右下方的工具盤中的專案型態選項來選擇要建立的項目型態，如下圖所示:



點選整合發展環境中右下方工具盤中的選項來建立專案型態

不管您喜歡使用那一種方法，現在請選擇建立『Multi-Device Application』項目，Delphi for iOS 整合發展環境便會顯示如下的對話盒詢問您要建立的專案種類:



下面的表格說明了這麼專案的意義:

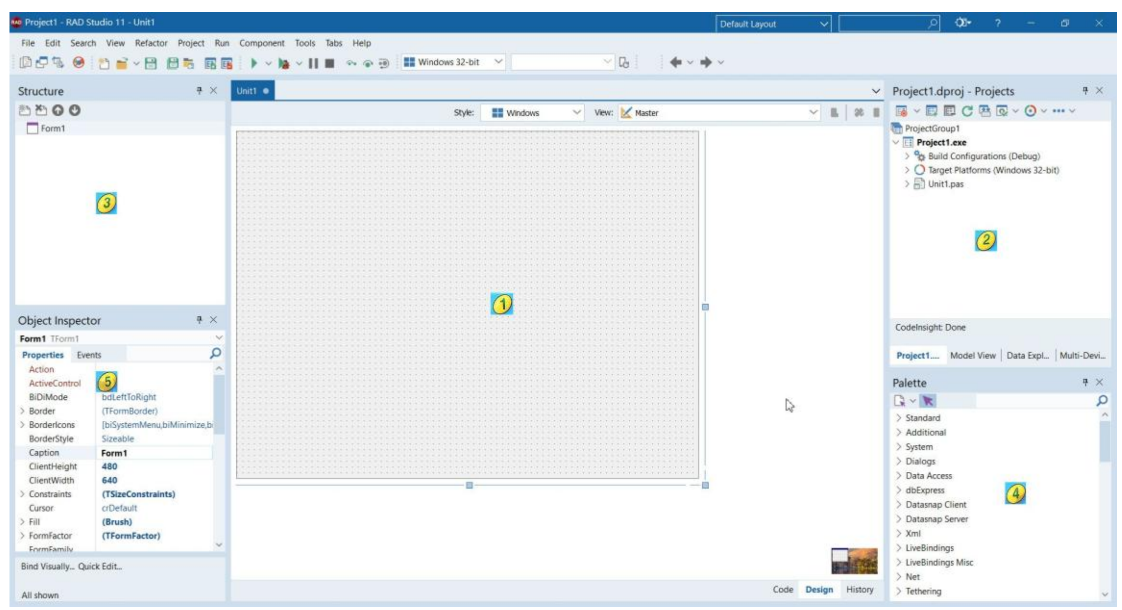
版權所有 請勿翻印

專案種類	說明
Blank Application	建立 2 維的空白 iOS App 專案
3D Application	建立 3 維的空白 iOS App 專案
其他種類的 iOS 項目	從內定的樣板 iOS App 中選擇要建立的專案

在我們的第 1 個 Mobile 範例中將使用 Blank Application 專案，因此請直接點選上圖中的 Blank Application 選項建立專案。

2-2 Delphi for iOS 專案組成元素

現在 Delphi for iOS 整合發展環境會為您建立 2 維的空白 iOS App 應用程式專案並且自動產生專案的內容檔案，此時整合發展環境也會發生一些變化，如下所示:

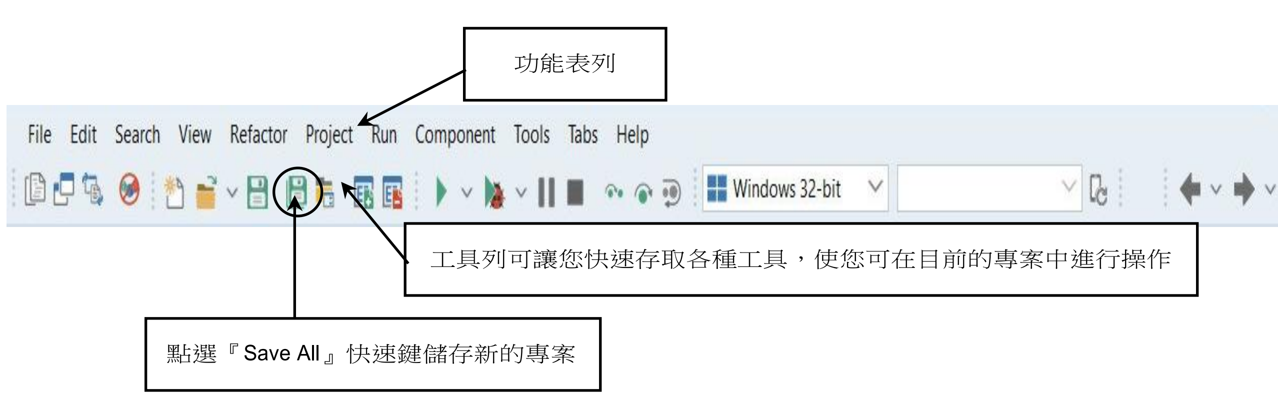


整合發展環境建立 FireMonkey 專案並且開啟專案主表單，準備讓您開始設計使用者介面

區域編號	說明
1	整合發展環境自動開啟專案中的主表單(Main Form)，準備讓您開始設計應用程式的圖形使用者介面
2	專案管理員顯示專案中所有的檔案和相關的設定
3	專案樹狀架構顯示主表單中的元件架構，但由於現在主表單中沒有任何元件，因此專案樹狀架構中只顯示了主表單(即畫面中的 Form1)
4	物件檢視器顯示主表單所有的特性和它的特性值，現在您就可以在物件檢視器中對特性值進行任何的設定或是改變
5	工具盤現在顯示了所有此專案型態可使用的元件，現在您就可以拖曳這些元件到主表單中進行應用程式的設計

版權所有 請勿翻印

在繼續說明之前，讓我們先儲存這個範例專案，請點選工具列中的『Save All』快速鍵儲存新的專案:



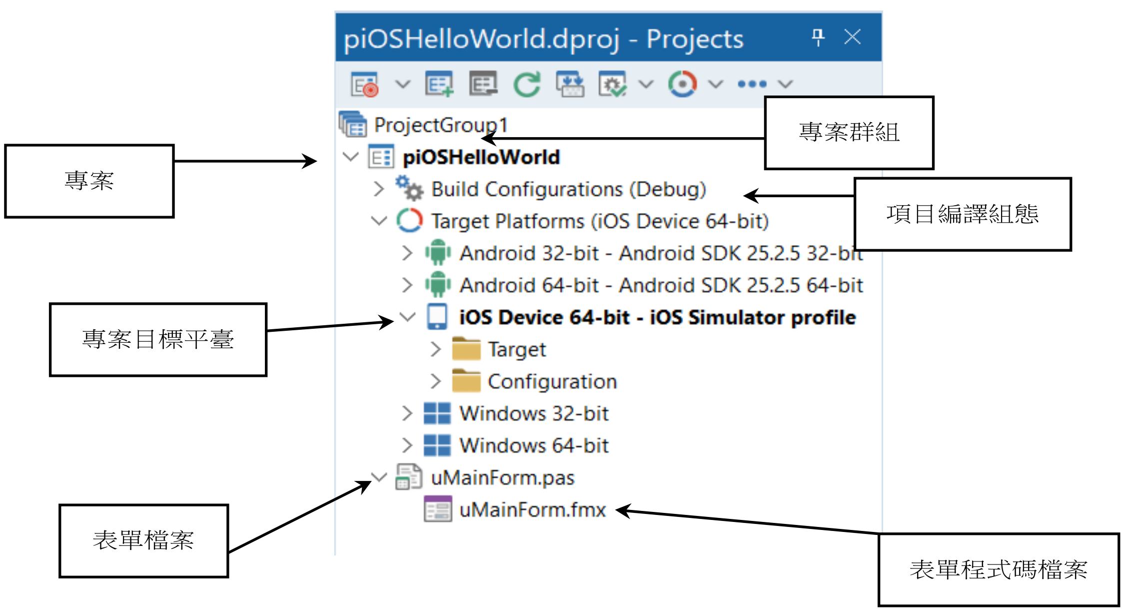
或是同時按下『Shift+Control+S』3 個按鍵儲存專案，當儲存專案時整合發展環境會詢問以什麼名稱儲存表單和專案，請以『MainForm』儲存主表單，以『piOSHelloWorld』儲存專案。

> 在儲存專案時，您可以選擇儲存在一個特定的目錄中，例如 C:\MyRIODemos，如果您直接儲存專案而沒有選擇特定的目錄，那麼 DelphiIDE 會把您的專案儲存在 C:\Users\您的名稱\Documents\RAD Studio\Projects\的內定目錄之下

Delphi for iOS 的專案是由不同的檔案和相關的設定所組成，其中主要的元素就是表單(Form)和程式碼檔案，其中表單是您設計您的應用程式圖形使用者介面的元素，而程式碼檔案則是您撰寫您應用程式邏輯程式碼的元素。在 FireMonkey 型態專案中，表單是以『.fmx』結尾的檔案，而程式碼檔案則是以『.pas』結尾的檔案，每一個表單都會擁有一個對應的程式碼檔案。

除了表單和程式碼檔案之外，專案中尚有其他的元素，這些元素都會顯示在專案管理員中。其中的『專案編譯組態』可設定目前專案是使用『除錯』模式，『部署』模式或是『客制化』模式。而『專案目標平臺』則可設定專案編譯的目標平臺，FireMonkey 專案可支援 Win32，Win64，Mac OSX 或是 iOS 平臺，由於我們是使用 Delphi for iOS 開發，而且在 1-2 小節中我們設定『iOS Simulator』為內定平臺，因此在項目中的 Target Platforms 節點中可以看到 iOS Simulator 被設定為目前專案使用的平臺:

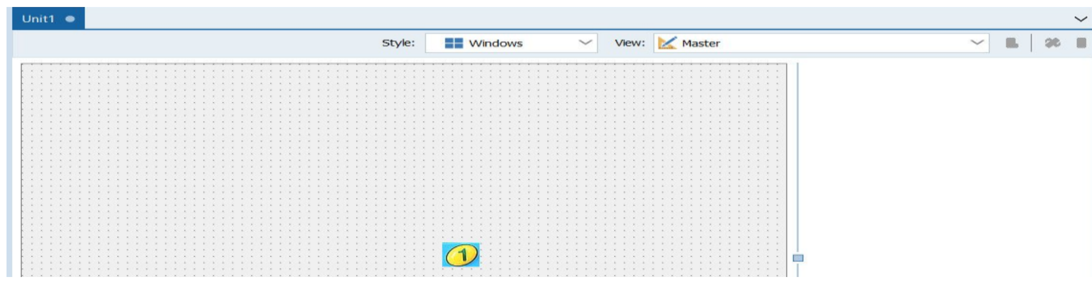
版權所有 請勿翻印



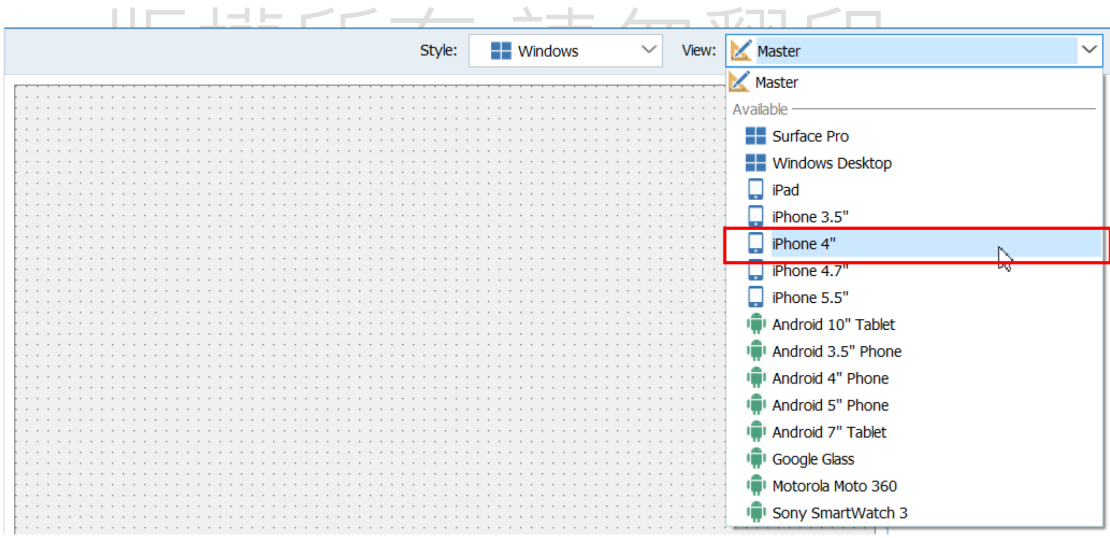
項目管理員

RIO 在 IDE 中新增了所謂的 MDD(Multi-Device Designer)的功能，基本上 MDD 是藉由多個 View 來設計不同平臺的 UI，當開發人員在 Delphi IDE 中建立 MDD Application 專案後 IDE 會顯示所謂的 Master View，Master View 是所有其他子 View 的父 View，我們馬上會建立一個 iPhone 的子 View。

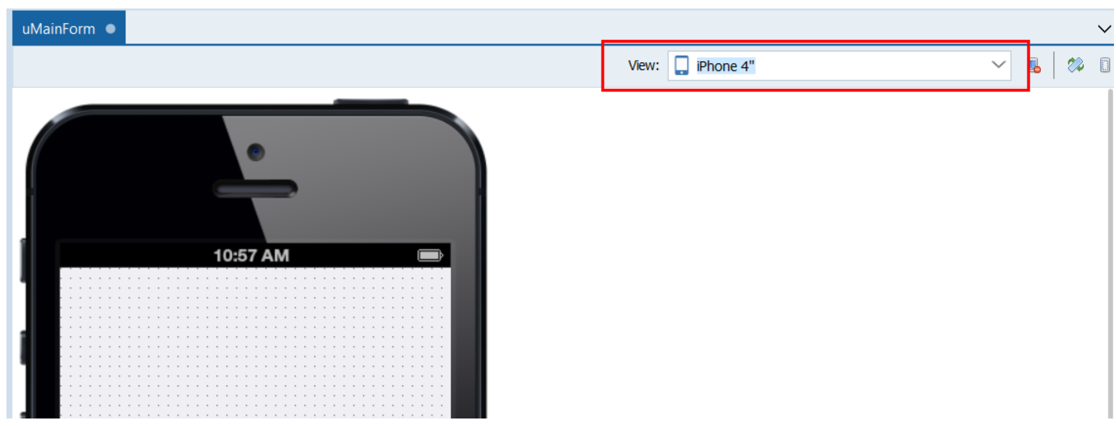
開發人員在 Master View 上加入的任何元件都會自動繼承到其他子妁 View 之中。當然開發人員也可以在子 View 中直接進行 UI 設計，



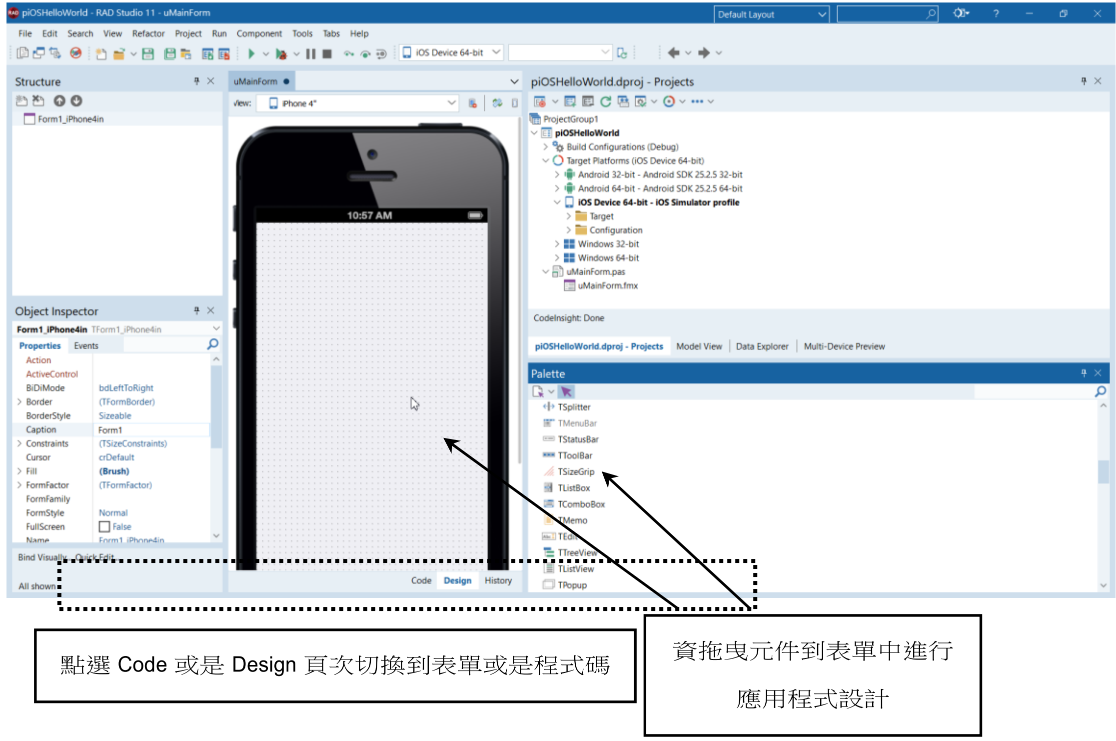
現在就讓我們在此項目中建立一個 iPhone 子 View，請點選 IDE 右上方的 Views 下拉盒並選擇其中的 iPhone 4，如下所示：



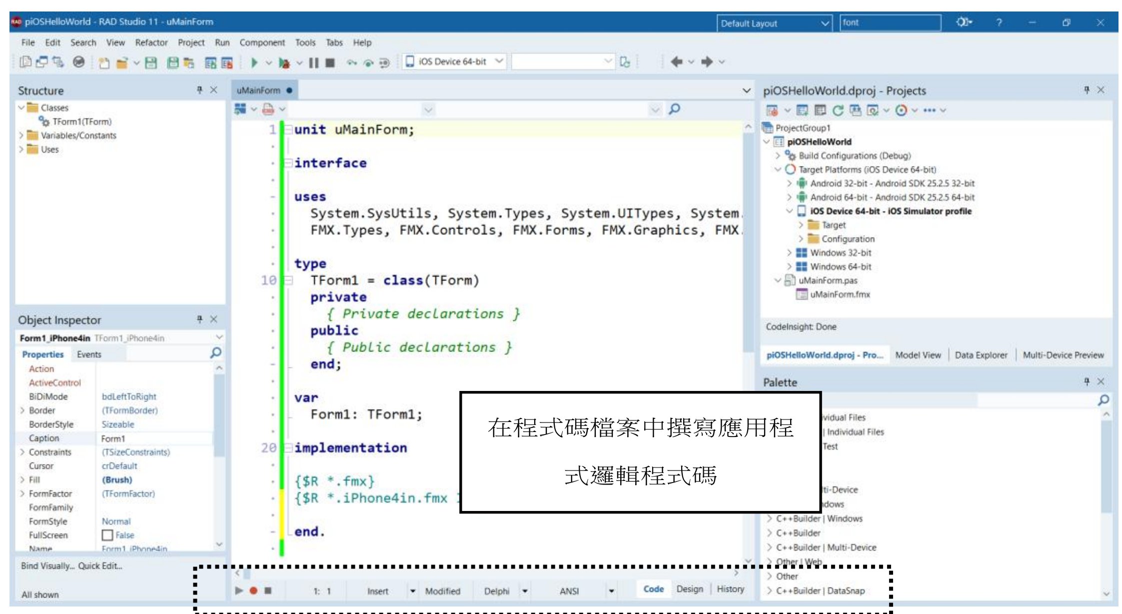
點選了 iPhone 4 子 View 之後 IDE 便會建立一個 iPhone 的設計表單如下所示並且在 IDE 右上方的 Views 下拉盒中可以看到現在項目有 2 個 View：Master View 和 iPhone 4：



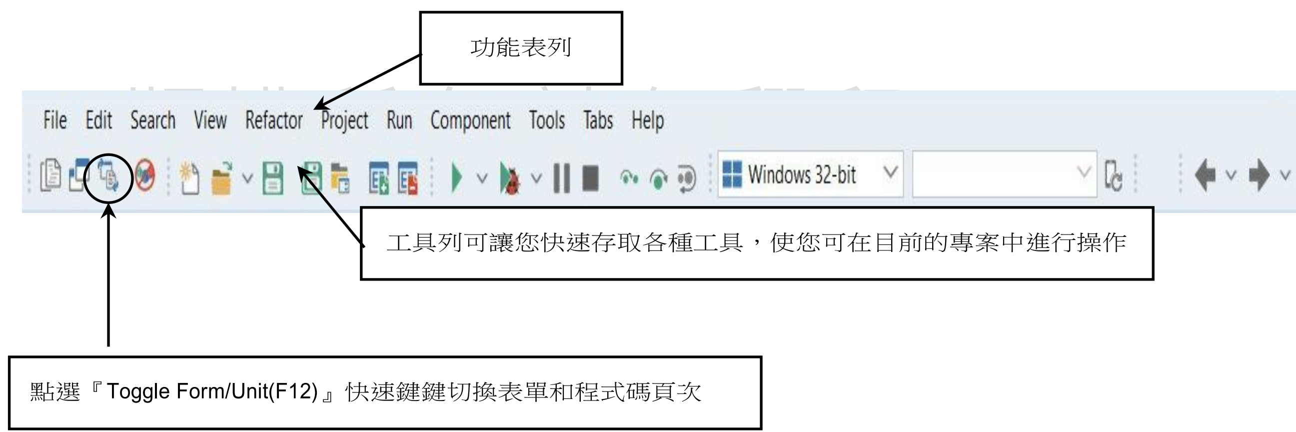
當項目建立之後，整合發展環境會自動開啟主表單，之後您就可以藉由拖曳整合發展環境右下方的元件到表單中開始進行應用程式設計:



或是切換到下圖到表單的程式碼檔案中開始撰寫程式碼。要在表單和它的程式碼檔案之間切換，您可以點選整合發展環境中下方的『Code』或是『Design』頁次切換到表單或是程式碼。點選『Code』可切換到程式碼，點選『Design』可切換到表單。



或是按下『F12』鍵切換表單和程式碼頁次，或是點選工具列中的『Toggle Form/Unit(F12)』快速鍵來切換:

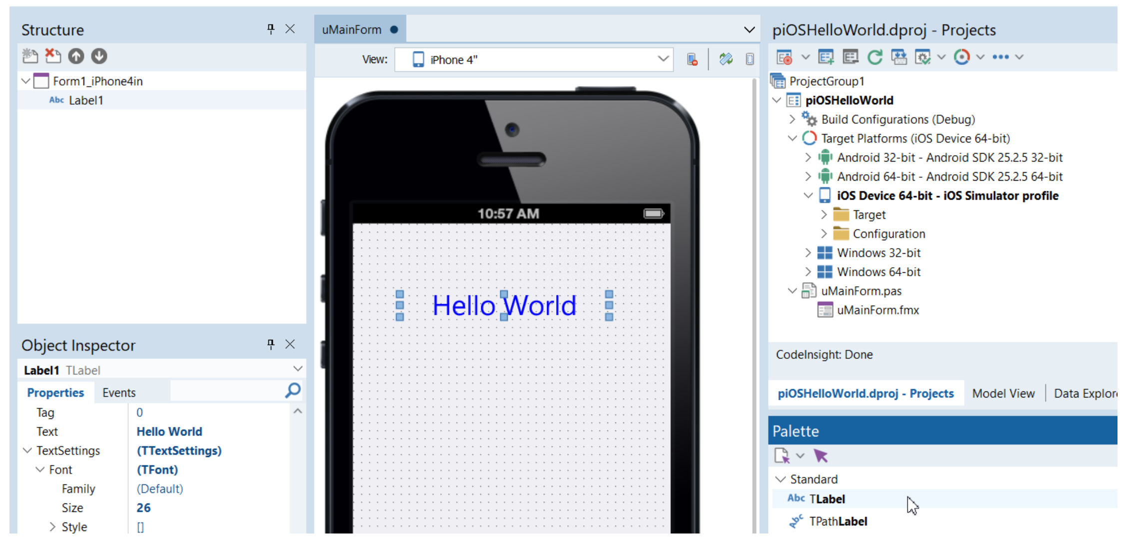


現在我們就可以開始開發您的 iOS 應用程式了。

3 開發您的第 1 個 iOS App

筆者在學習開發 iOS App 時閱讀過許多使用 XCode 開發的書籍，許多書籍在介紹如何開發第一個 iOS App 時都是使用 Hello World 這個範例(事實上這幾乎成了所有開發書籍的第一個範例了，不是嗎?)，但對於習慣使用 Delphi 的筆者來說，XCode 實在過於繁瑣，讓我們看看 Delphi for iOS 是多簡單，快速就可以完成更好的 Hello World iOS App。

現在讀者繼續前面儲存的『piOSHelloWorld』專案，在 Delphi for iOS IDE 加下方的工具盤上方的 Search 欄位中輸入 tlabel 以搜尋 TLabel 元件，接著拖曳工具盤中的 TLabel 元件到 iPhone 表單中，如下所示:

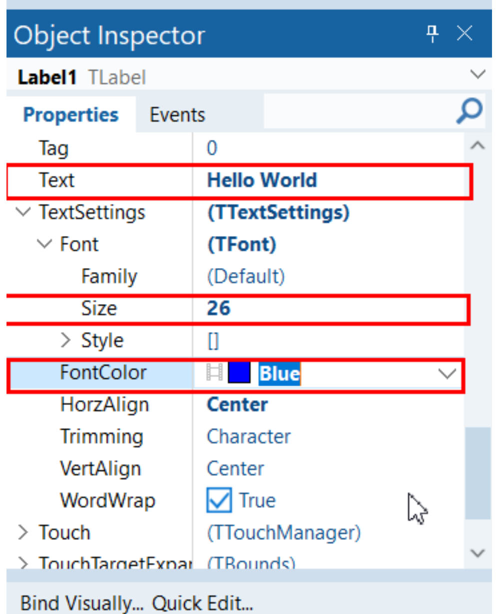


接著到 IDE 左下方的物件檢視器中設定 TLabel 如下的特性值:

特性名稱	特性值
TestSettings 中的 Font	26
TestSettings 中的 FontColor	Blue
Text	Hello World

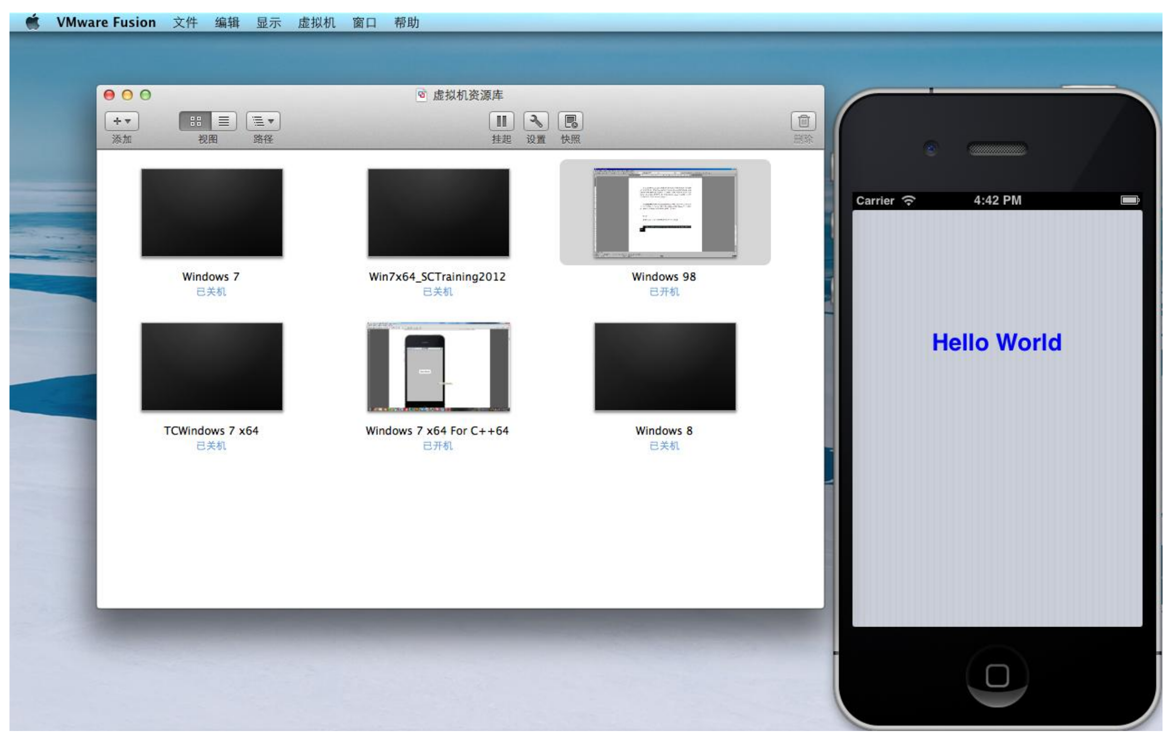
版權所有 請勿翻印

物件檢視器如下所示:



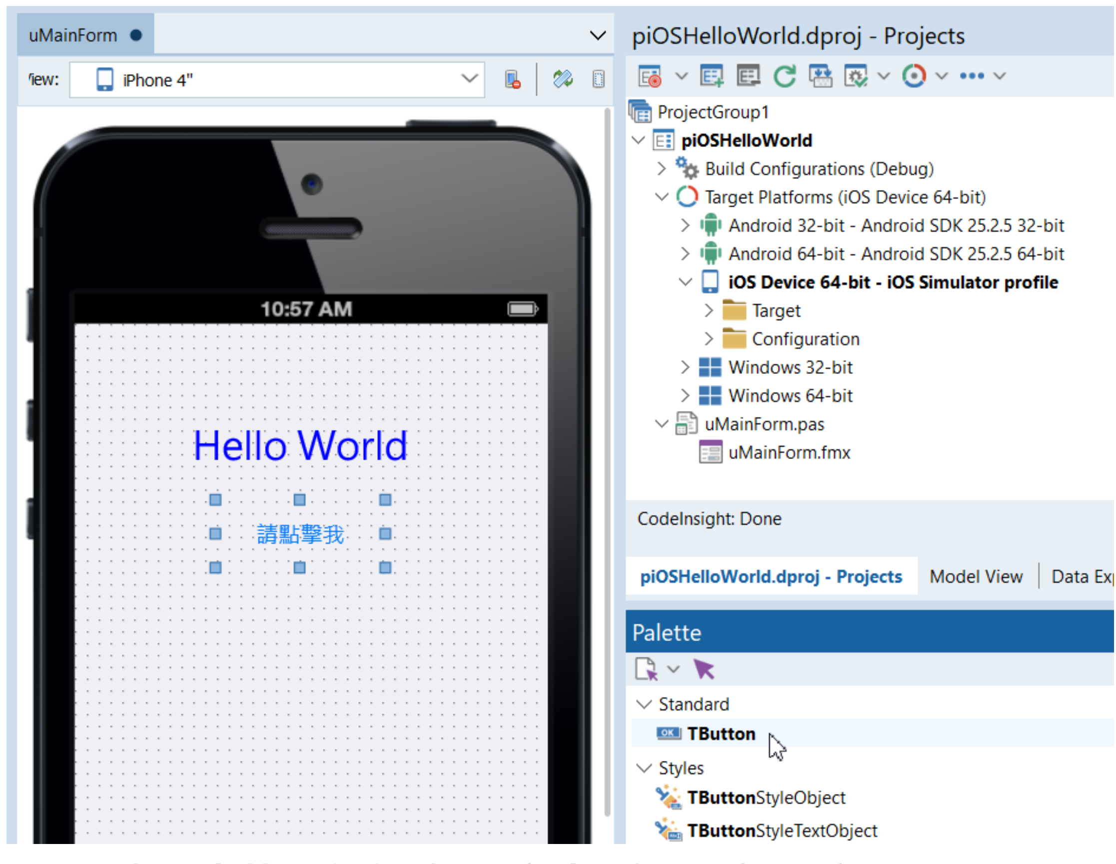
之後 iPhone 表單 TLabel 元件中的 Hello World 文字就會變成上圖顯示的效果。

現在您只需要按下 F9 或是 Shift+Ctrl+F9，Delphi for iOS 就會編譯，並且分發『piOSHelloWorld』專案到 Mac 平臺的 iOS 模擬器中執行了，例如下圖就是『piOSHelloWorld』專案在 Mac 平臺的 iOS 模擬器中執行的畫面(請確定 PAServer 已經執行在 Mac OS 中並且 IDE 的組態已經正確設定):



如何? 使用 Delphi for iOS 開發 iOS App 是不是太方便了呢?

讓我們繼續改善我們的第一個 iOS App 吧。讓我們在工具盤中搜尋 TButton 元件然後拖曳到 iPhone 表單中，然後在物件檢視器中設定它的 Text 特性值為『請點擊我』，如下所示:



版權所有 請勿翻印

然後使用滑鼠按兩下 iPhone 表單中的 TButton 元件，IDE 便會把我們帶到程式碼編輯器中，請在其中撰寫如下的程式碼:

```
procedure TForm5.Button1Click(Sender: TObject);
begin
  ShowMessage('歡迎使用 Delphi for iOS!');
end;
```

在稍後本書會說明，這段程式碼稱為事件處理函式，在其中我們呼叫 ShowMessage 函式顯示訊息，ShowMessage 函式是 Delphi for iOS 的執行時期函式館中的函式。

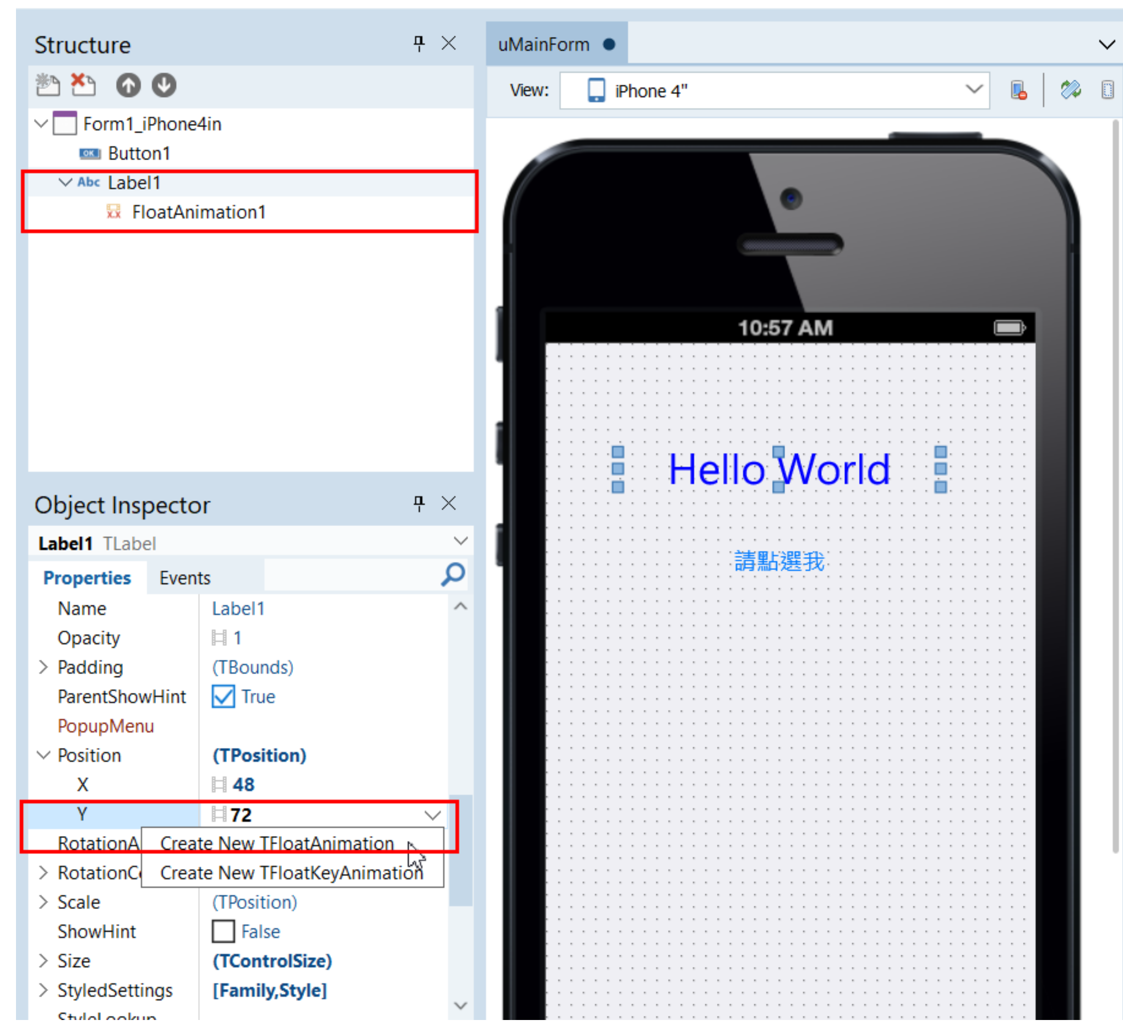
現在請再次按下 F9 執行『piOSHelloWorld』項目，我們就可以在 iOS 模擬中看到修改過的 piOSHelloWorld App，如果讀者使用滑鼠點選表單中的『請點選我』按鈕，就可以看到 piOSHelloWorld App 使用 iPhone 的原生訊息盒顯示訊息，如下所示:

版權所有 請勿翻印

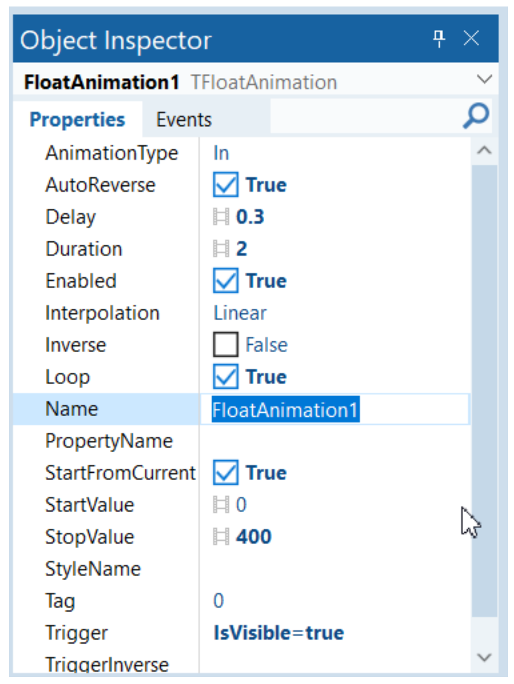
讀者可以看到使用 Delphi for iOS 撰寫 iOS 程式碼並且連結視覺化元件是多麼的容易。在我們離開第一個 iOS App 之前，最後再讓我們為 Hello World 這個文字加入動畫的功能。

讓我們在執行 piOSHelloWorld App 把『Hello World』這個文字從目前的位置動態的往下移動，再往上移動回原位置，如此反復不停的移動。要達到這個視覺化效果非常的簡單，因為這基本上就是把 iPhone 表單中的 TLabel 元件在 Y 軸移動和變化。

因此請回到 Delphi for iOS IDE，點選 iPhone 表單中的 TLabel 元件，然後在物件檢視器中打開它的 Position 特性，我們可以在其中看到它的 X 和 Y 兩個子特性，再使用滑鼠點選 Y 特性，就會如下圖看到物件檢視器顯示一個下拉式功能表，請點選其中的『Create New TFloatAnimation』以便為 TLabel 組件的 Y 軸建立動畫效果物件。



此時物件檢視器便會顯示此新建立的 TFloatAnimation 物件的特性，如下所示:



請在物件檢視器中設定 TFloatAnimation 物件如下的特性值:

特性	特性值	說明
AutoReverse	True	當動畫完成時自動以反方相再執行一次
Delay	0.3	每一動畫動作之間延遲 0.3 秒
Duration	2	此動畫效果一共執行 2 秒
Enabled	True	啟動動畫功能
Loop	True	不斷重複執行此動畫功能
StartFromCurrent	True	從目前 TLabel 元件的 Y 軸位置開始動畫功能
StopValue	400	動畫功能到達 TLabel 組件的 Y 軸位置 400 的地方時就完成
Trigger	IsVisible=true	當 TLabel 元件顯示時就觸發執行動畫功能

版權所有 請勿翻印

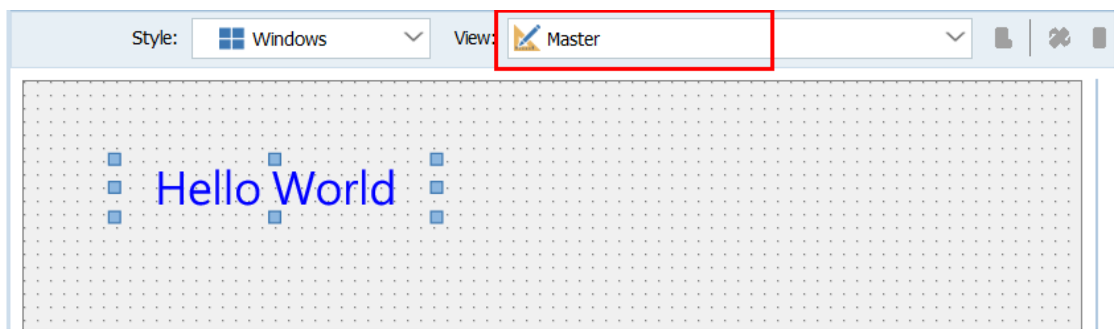
現在請再次按下 F9 執行『piOSHelloWorld』項目我們，就可以在 iOS 模擬中看到修改過的 piOSHelloWorld App，此時 iPhone 表單中的 TLabel 元件就會不停的自動上下移動，如下所示:

版權所有 請勿翻印

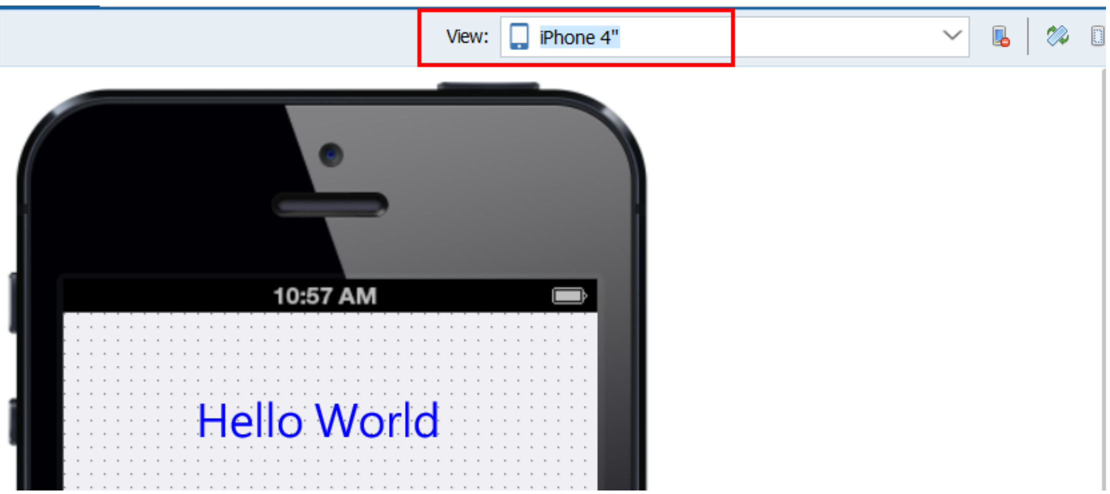
OK，現在我們完成了第一個 iOS App，這個具備動態視覺化效果的 Hello World iOS App 使用 Delphi for iOS 開發起來不但比其他 iOS 開發工具更簡單，提供的功能也遠遠超過了只是靜態的顯示 Hello World 文字。從這個簡單的範例就可以證明 Delphi for iOS 比其他 iOS 開發工具更強大，更易於使用，也具體更高的生產力。

3-1 使用 MDD 設定

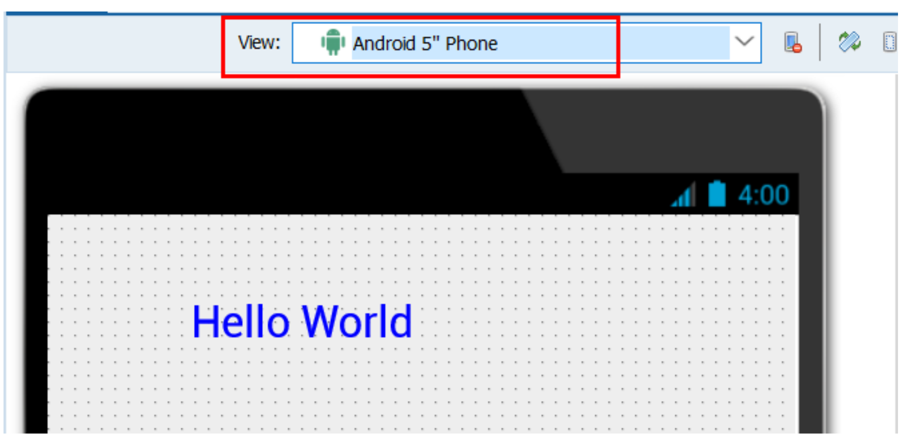
在前面的說明中我們是直接在 iPhone 的子 View 中進行 UI 設計，當然您也可以在 Master View 中進行共用設計再到特定的子 View 中進行調整，例如我們可以先在 Master View 中加入 Hello World 的 TLabel 元件：



再切換到 iPhone View 就可以看到 iPhone View 繼承了 Master View 中的元件以及設定：



如果此時我們再加入開發 Android View 也可以看到 Android View 也繼承了 Master View 中的元件和設定：



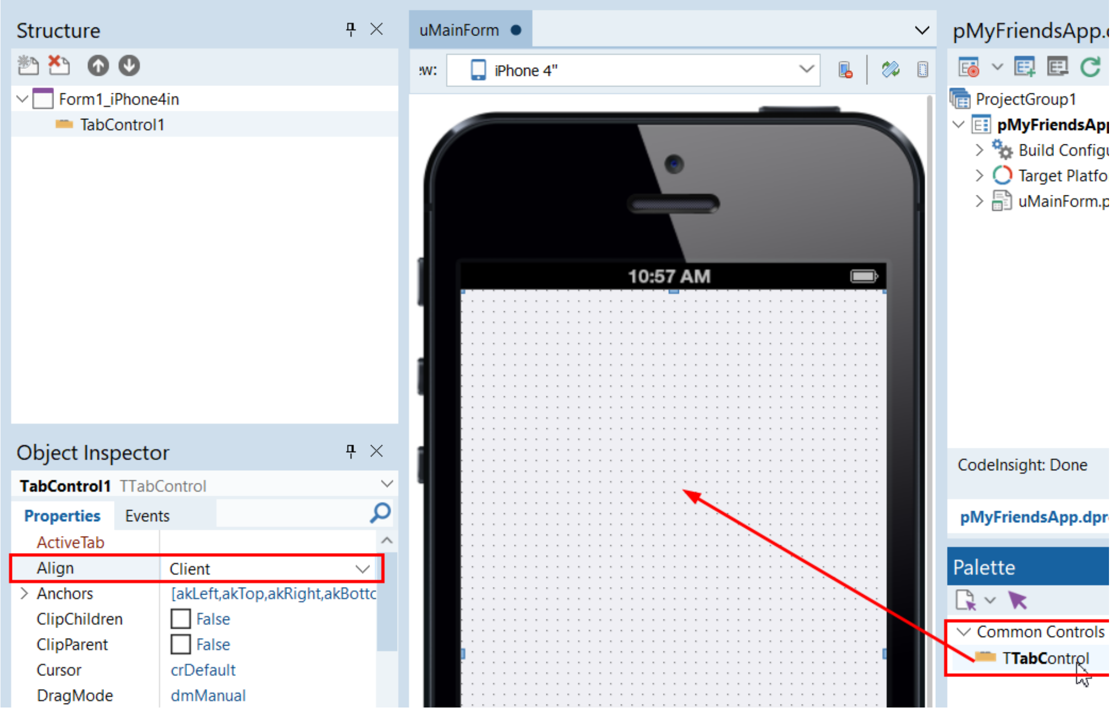
所以使用 MDD 的使用時機是把所有子平臺共同需要的 UI 元件在 Master View 中先放入完成，再分別切換到不同的子平臺進行特定平臺的細節設定。

接下來讀者將開始學習如何有效的使用 Delphi for iOS IDE 來開發 iOS App，我們將使用 FireMonkey For Mobile 框架開發個人通訊錄 App，在開發的過程中讀者將學習許多 Delphi for iOS 的使用技巧以及 FireMonkey For Mobile 框架的元件。

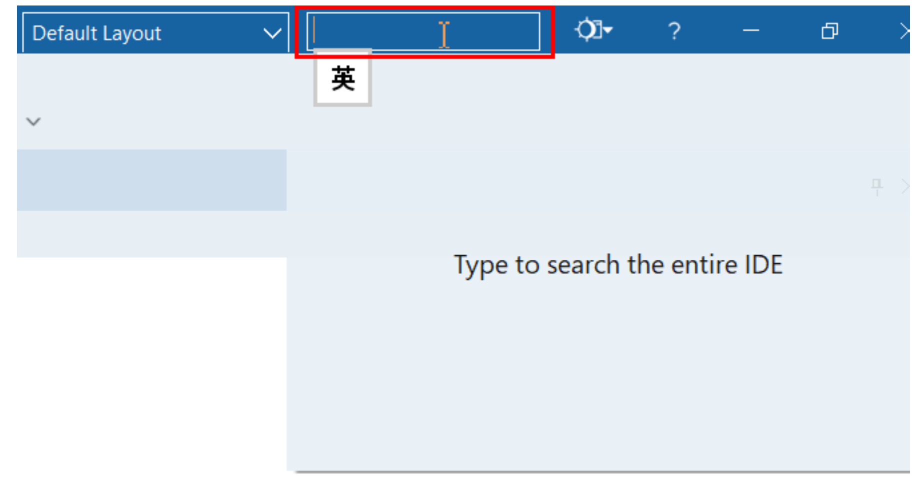
4. 使用 Delphi for iOS 整合發展環境

請執行 Delphi for iOS IDE，建立一個『Multi-Device Application』空白專案，並且以『pMyFriendsApp』為名稱儲存此 iOS App。

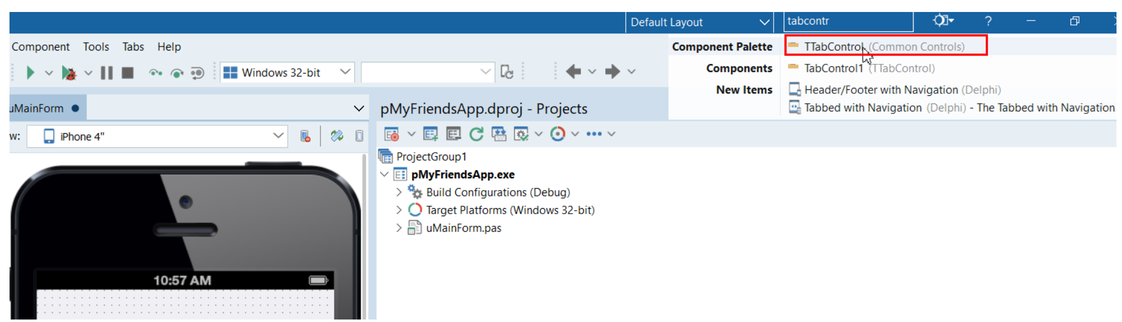
在 IDE 左下方的工具盤中搜尋 TTabControl 元件，拖曳到 iPhone 主表單中(當然您也可以把 TTabControl 放在 Master View 中再切換到 iPhone View 中)，在物件檢視器中設定 TTabControl 組件的 Align 特性值為『alClient』，如下所示:



由於 FireMonkey For Mobile 框架提供的元件非常的多，因此您也可以按下 F6 鍵，整合發展環境便把游標定位在 IDE 右上方的搜尋欄位中等待您輸入的搜尋的關鍵字：

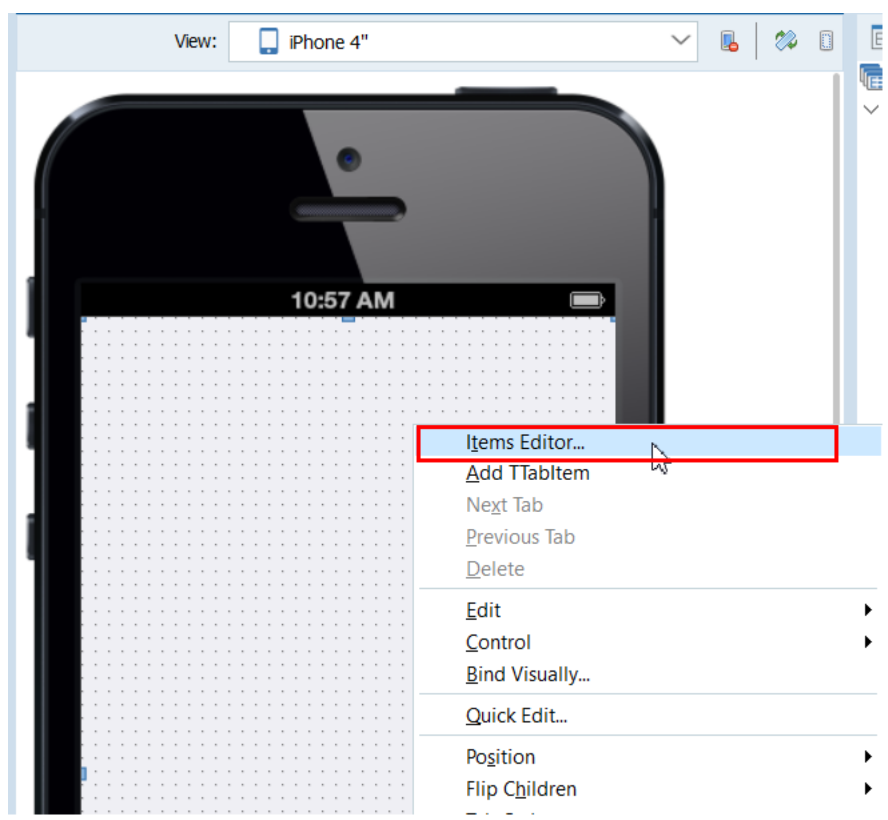


您只需要輸入要搜尋的關鍵字內容即可在整合發展環境中搜尋，例如下圖輸入 TTabControl 即可找到 TTabControl 元件，接著使用滑鼠按兩下 TTabControl 即可自動把 TTabControl 加入表單中。或是您要搜尋的元件的部份名稱，例如『tabcont』，『bcontrol』等都可以找到符合輸入名稱的元件。



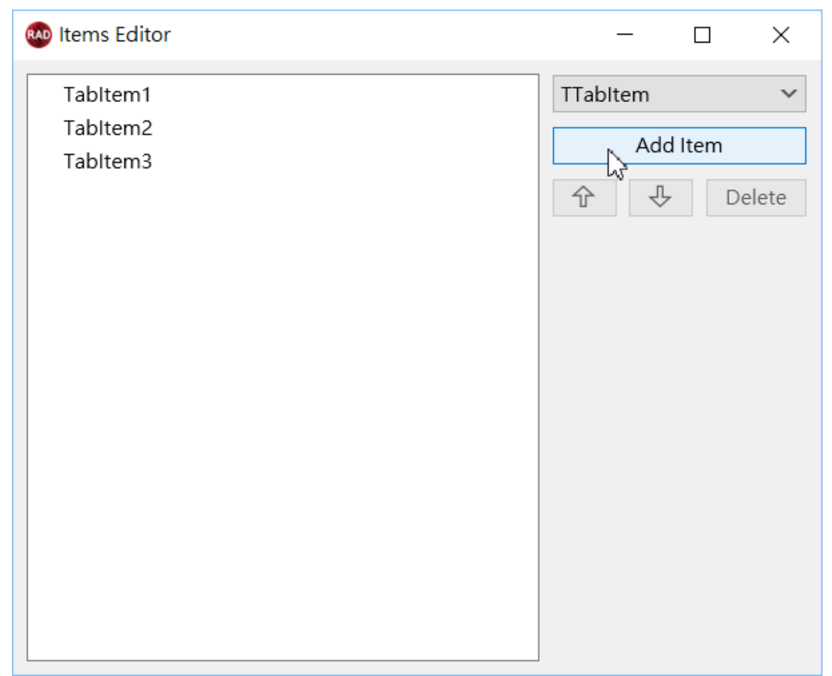
請注意『IDE Insight』會根據您現在使用的模式來搜尋內容，例如如果您是在程式碼編輯器中按下 F6 啟動『IDE Insight』並且搜尋 TTabControl，那麼便搜尋不到，因為 TTabControl 組件無法加入到程式碼之中。

現在讓我們在這個 TTabControl 元件中加入數個頁面，請點選表單設計家中的 TTabControl 元件並且點選滑鼠右鍵，此時 IDE 便會顯示一個快顯功能表，請選擇其中的『Items Editor…』選項以啟動選項設計者對話盒:

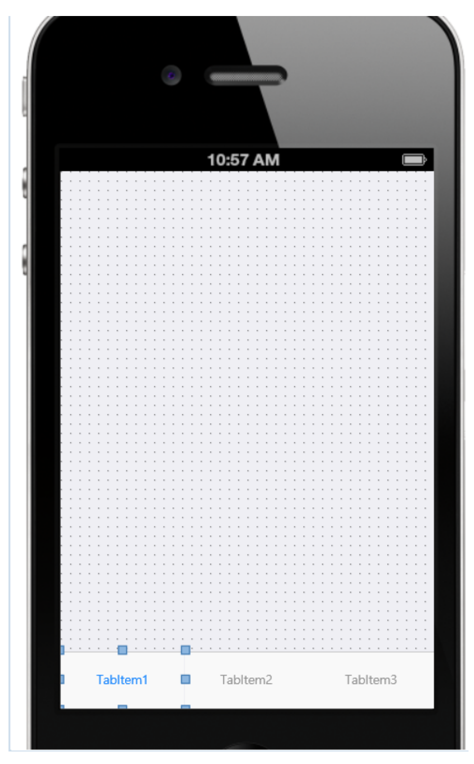


接著點選對話盒中的『Add Item』按鈕在 TTabControl 元件中加入四個 TTabItem 子組件，如下所示:

版權所有 請勿翻印

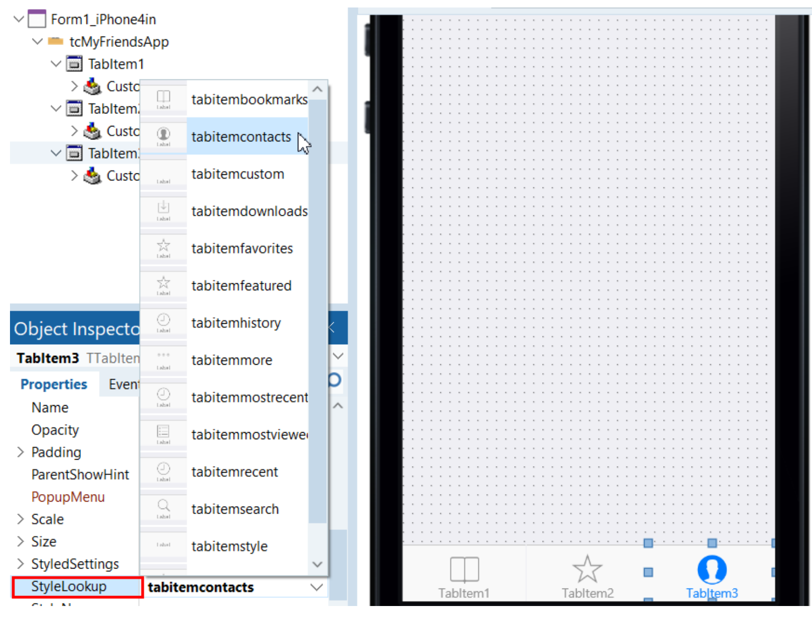


接著在物件檢視器中設定 TTabControl 組件的 TabPosition 特性值為『tpBottom』，此時表單設計者便類似如下所示:



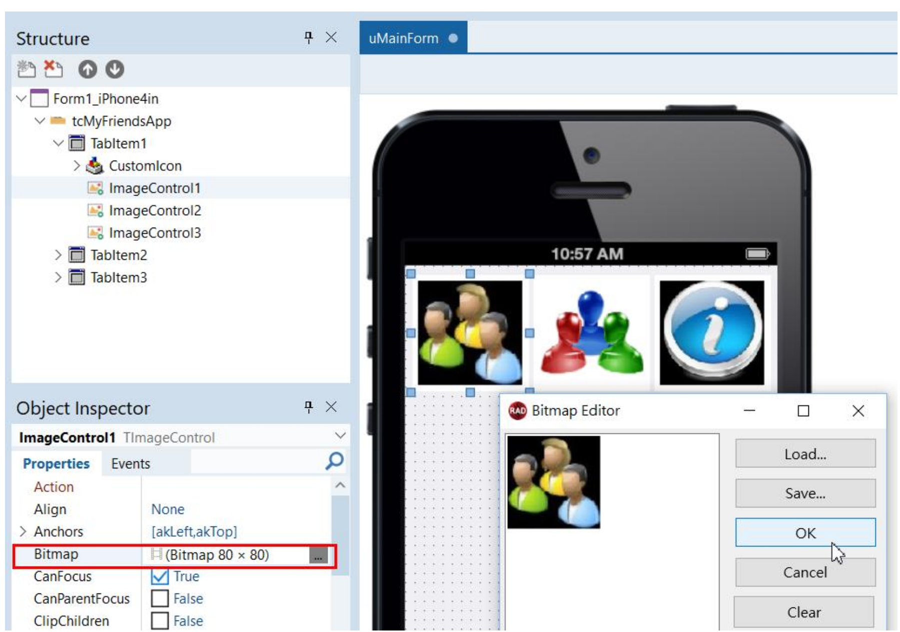
現在在 TTabControl 元件的下方便會出現頁面的按鈕，讓我們改變這些按鈕的外觀，讓這個 FireMonkey 應用程式更像典型的 iOS App，請點選 TTabControl 元件下方的按鈕，在物件檢視器中點選 StyleLookup 特性，您就可以從其中為按鈕選擇不同的外觀風格。例如下圖就是為 4 個頁面按鈕選擇並且設定不同外觀風格的結果:

版權所有 請勿翻印



接著讓我們在第一個頁面加入三個 TImageControl 元件以載入和顯示 3 個圖像。要在 TImageControl 元件中載入圖像，我們可以點選 TImageControl

元件，然後在物件檢視器中按兩下它的 Bitmap 特性，IDE 便會顯示 Bitmap 特性的特性值編輯器，在這個『Bitmap Editor』對話盒中我們就可以點選『Load…』按鈕以載入需要的圖像，如下所示:



現在讓我們為這個 FireMonkey App 加入一些有趣的功能，這個功能就是當這個 FireMonkey App 執行時，如果使用者點選上圖中最右方的圖像時就動態顯示一個此 App 的資訊文字，這個動態文字會從畫面下方往上出現，接著在一定的時間之後這個資訊文字就會消失。

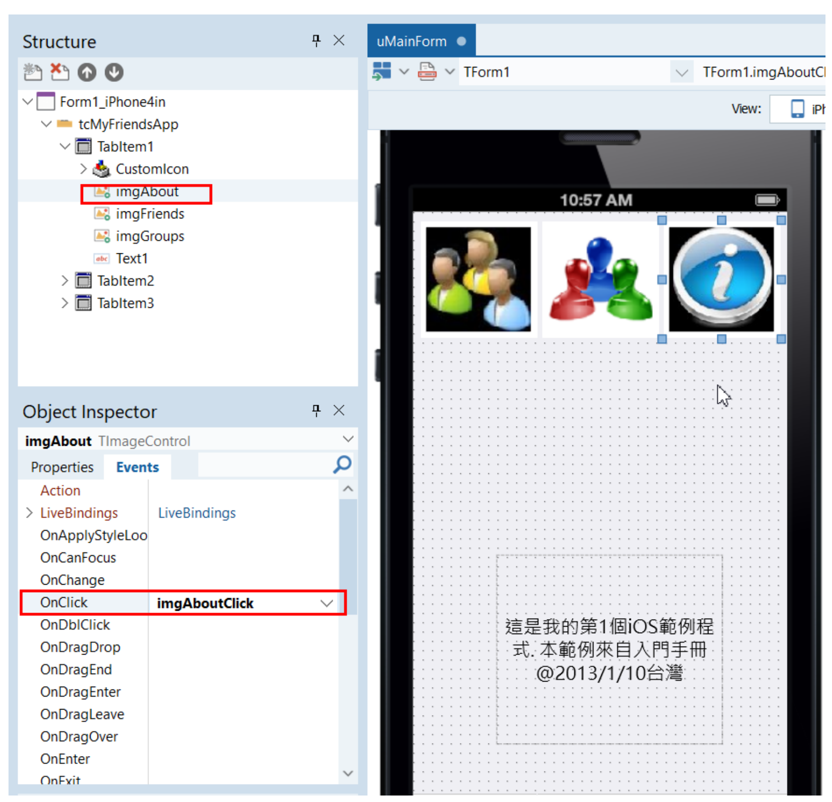
版權所有 請勿翻印

這種動態顯示文字的功能可以使用 FireMonkey 框架中的動畫功能(Animation)輕易的完成。因此首先請在工具盤中找到 TText 元件，拖曳到 TTabControl 的第一個頁面的下方，並且請在 TText 元件的 Text 特性中輸入一些資訊文字，如下所示:

我們希望點選主表單中最右方的驚嘆號圖像時動態顯示 TText 元件的內容，因此請點選驚嘆號圖像，在物件檢視器的 Events 頁次中按兩下它的 OnClick 專案以自動產生 OnClick 事件處理函式．

版權所有 請勿翻印

現在我們開始需要撰寫一些程式碼了，請點選 TTabControl 的第一個頁面中最右方的圖像元件，接著點選物件檢視器中的『Events』頁次，並且按兩下其中的 OnClick 事件，如下所示:

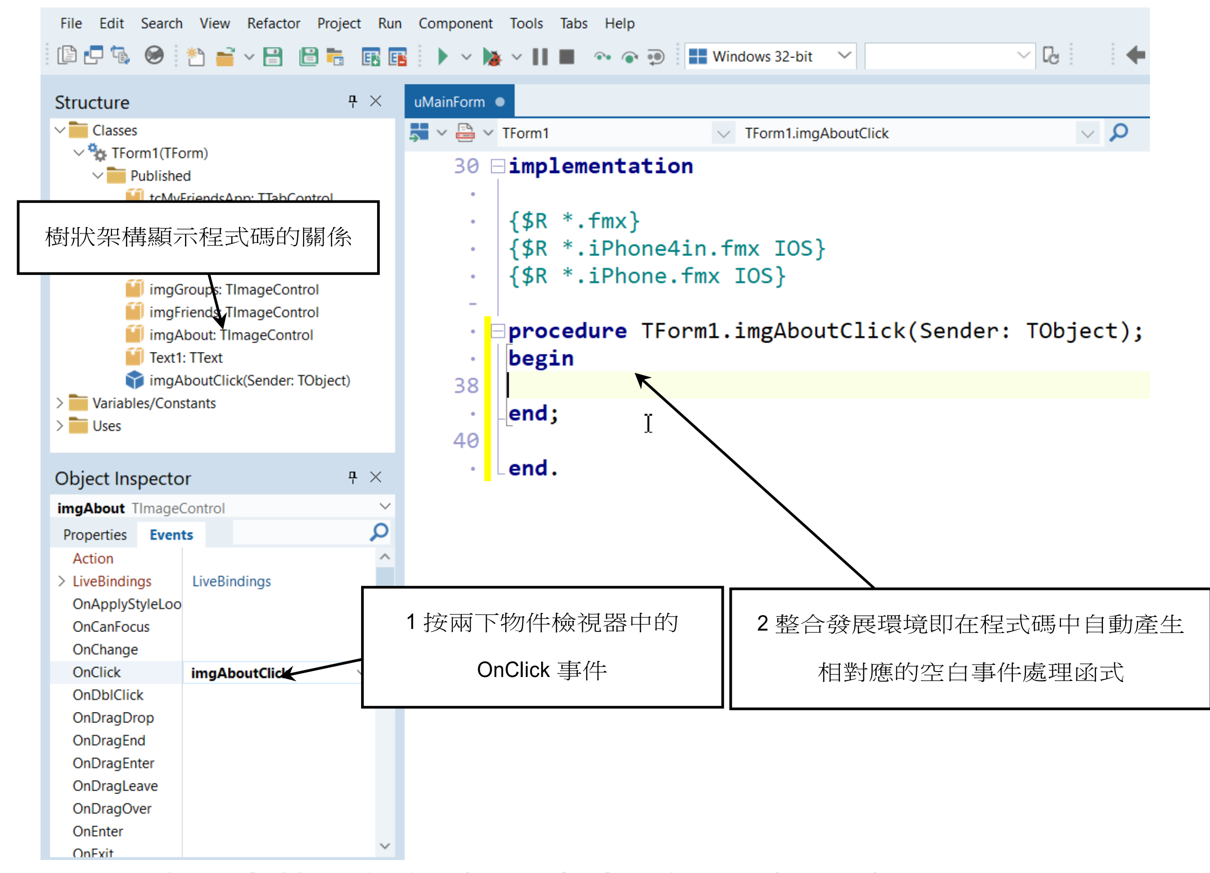


點選表單，在物件檢視器的『Events』頁次按兩下事件以自動產生事件處理函式

版權所有 請勿翻印

這個動作會讓整合發展環境自動在程式碼頁次中產生空白的事件處理函式，接著您就可以在程式碼頁次中開始撰寫程式碼了，這個產生事件處理函式的動作是您之後使用整合發展環境最常使用的功能之一。下圖即顯示了執行這個動作之後整合發展環境會自動切換到程式碼頁次並且讓游標自動停駐在空白的事件處理函式程式區塊中，準備讓您撰寫程式碼。

請注意現在整合發展環境左上方的樹狀架構視窗也從顯示元件關係切換為顯示程式碼的關係:



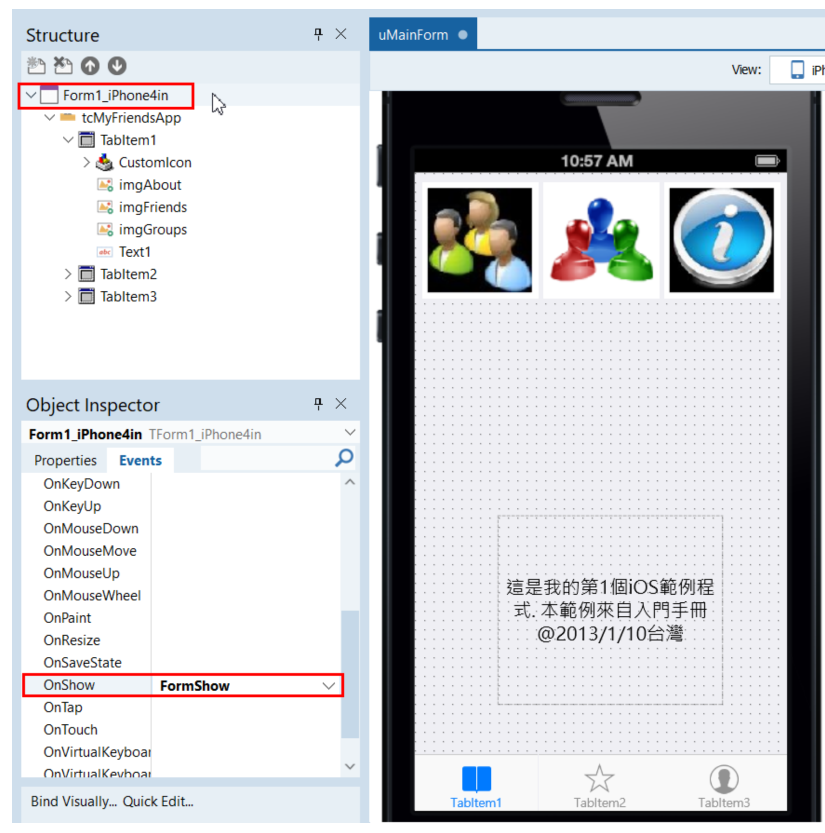
版權所有 請勿翻印

按兩下事件整合發展環境即可自動建立相對應的事件處理函式

現在請您在 OnClick 事件處理函式中撰寫如下的程式碼(請勿輸入每行程式碼之前的行數號，例如 001, 002 等，行數號只為說明程式碼的意義使用):

```
001 ShowWelcomeText;
```

OnClick 事件處理函式呼叫了 ShowWelcomeText 方法來顯示前面加入的 TText 元件，但在說明 ShowWelcomeText 方法之前，我們需要再撰寫一些初始化程式碼。請點選整合發展環境左上方的樹狀架構視窗，選擇主表單物件『fmMainForm』，再於『Events』頁次按兩下 OnShow 以自動產生主表單的 OnShow 事件處理函式，如下所示:



主表單的 OnShow 事件處理函式在 003 行設定主表單中的 TTabControl 元件顯示第一個頁面，接著 004 行呼叫 SetupWelcomeText 方法對於 TText 元件進行初始化設定。

版權所有 請勿翻印

SetupWelcomeText 方法於 009 行設定 TText 元件(txtWelcome)的 Visible 特性值為 False 以隱藏此組件，並且於 010 行改變它的 Y 軸位置，

```
txtWelcome.Position.Y := Self.Height + 10;
```

的功能就是把此 Text 元件移動到主表單可顯示的區域之外，讓它無法顯示在主表單的區域中。

```
001    procedure TfmMainForm.FormShow(Sender: TObject);
002    begin
003      tcMyFriedsApp.ActiveTab := tiFirstPage;
004      SetupWelcomeText;
005    end;
006
007    procedure TfmMainForm.SetupWelcomeText;
008    begin
009      txtWelcome.Visible := False;
010      txtWelcome.Position.Y := Self.Height + 10;
```

```
011    end;
```

OK，接下來就簡單了。由一開始我們就把 TText 元件隱藏而且移動到顯示區域之外，因此在 ShowWelcomeText 方法中我們只需需要把它移回顯示區域並且顯示它即可。但只是簡單的如此做沒什麼意思，讓我們使用 FireMonkey 框架的動態顯示功能來顯示此 Text 元件。

在 ShowWelcomeText 方法的 003 行先設定 txtWelcome 的 Visible 特性值為 True 以顯示它，004 行呼叫 txtWelcome 的 AnimateFloat 方法動態的顯示出來。AnimateFloat 方法的宣告原型如下:

```
procedure AnimateFloat(const APropertyName: string; const NewValue:
Single; Duration: Single = 0.2;
  AType: TAnimationType = TAnimationType.atIn;
  AInterpolation: TInterpolationType = TInterpolationType.itLinear);
```

AnimateFloat 的第一個參數是需要動態效果的特性名稱，第二個參數是此特性動態效果之後的新數值，第三個參數是動態效果持續的時間，最後的 2 個參數是動態效果的種類，由於最後 2 個參數都擁有內定值，因此我們只需要傳入前 3 個參數即可。

版權所有 請勿翻印

因此 004 行傳入 AnimateFloat 方法的第一個參數是'Position.Y'，代表要對 TText 元件的垂直位置進行動態效果，第二個傳入的參數值是

```
   ((txtWelcome.Parent) as TControl).Height - txtWelcome.Height + 10
```

它代表 TText 組件的新垂直位置，也就是 TTabControl 元件第一個頁面的高度減去 TText 組件的高度，再加上上 10 個圖元的高度。

最後的參數值 2 代表整個動態效果的時間會維持 2 秒鐘。

```
001    procedure TfmMainForm.ShowWelcomeText;
002    begin
003      txtWelcome.Visible := True;
004      txtWelcome.AnimateFloat('Position.Y', ((txtWelcome.Parent) as
TControl).Height - txtWelcome.Height + 10, 2);
005      Timer1.Enabled := True;
006    end;
```

最後在主表單中加入一個 TTimer 元件，設定它的 Interval 特性值為 5000，在它的 OnTimer 事件處理函式中撰寫如下的程式碼:

```
001    procedure TfmMainForm.Timer1Timer(Sender: TObject);
002    begin
```

```
003       SetupWelcomeText;
004       Timer1.Enabled := False;
005    end;
```

OnTimer 事件處理函式的工作很簡單，就是在 TText 元件顯示了 5 秒之後就在 003 行呼叫 SetupWelcomeText 方法把 TText 元件回復成初始化的狀態，004 行再停止 TTimer 元件的運作。

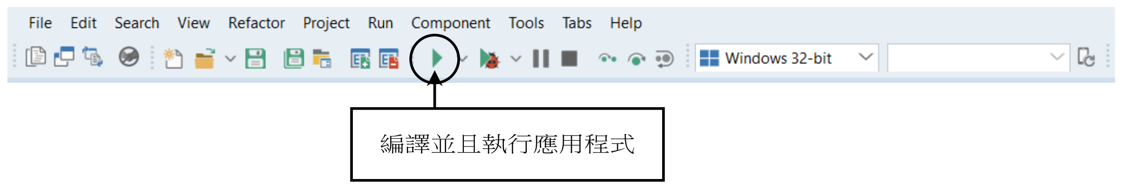
現在您的程式碼頁次應該看起來類似如下所示:

```
110  procedure TfmMainForm.FormShow(Sender: TObject);
     begin
     tcMyFriedsApp.ActiveTab := tiFirstPage;
113  SetupWelcomeText;
     end;

     procedure TfmMainForm.imgcAboutClick(Sender: TObject);
     begin
     ShowWelcomeText;
     end;
120
     procedure TfmMainForm.SetupWelcomeText;
     begin
     txtWelcome.Visible := False;
     txtWelcome.Position.Y := Self.Height + 10;
     end;
```

整合發展環境編輯器的變更長條

請注意在程式碼頁次的左方有一條綠色和黃色的線條，這個線條稱為『變更長條』，綠色代表從上次儲存這個程式碼頁次之後沒有被改變的程式碼，而黃色則代表已經被改變的程式碼，由於剛才我們加入了 3 行程式碼，因此新加入的程式碼之前的『變更長條』都是黃色的線條。如果您此時儲存此專案，那麼這 3 行程式碼就會變成綠色，您可以試試看。

現在您就可以試著執行此範例應用程式了，您可以點選工具列中的『Run Without Debugging』快速鍵編譯並且執行此應用程式，如下所示:



或是同時按下『Shift+Ctrl+F9』鍵。

如果您沒有打錯程式碼的話，那麼您就可以在 iOS 的 Simulator 中看到下面的 2 個執行畫面，當您點選最右邊的驚嘆號圖像時就可以看到文字動態的慢慢的從下往上出現非常的意思，讀者使用 FireMonkey 框架可以輕易的在 iPhone/iPad 中實作出精彩的動態效果，您使用 DelphiFor iOS 整合發展環境開發的第一個應用程式已經正確的執行了，使用 DelphiFor iOS 開發 iOS App 真的又簡單生產力又高，不是嗎?

4-1 移動程式碼區塊

再看看剛才輸入的程式碼，它們都靠在編輯器的最左邊，這其實是不太好的程式碼風格，讓我們看看如何使用按鍵來移動程式碼區塊。首先把游標移動到下圖 55 行左邊第一個位置，按下 Shift 鍵，再按下向下鍵『↓』把 2 行程式碼選擇，此時編輯器會以反白顯示被選擇的程式區塊，或是直接使用滑鼠選擇這 2 行程式區塊，接著同時按下『Ctrl+Shift+I』，您就可以看到被選擇的程式區塊整個往右方移動，如下圖所示:

當然您也可以把程式區塊往左移動，您只要按下『Ctrl+Shift+U』就可以把整個程式區塊往左方移動。

```
procedure TForm1.SetupWelcomeText;
begin
  txtWelcome.Visible := False;
  txtWelcome.Position.Y := Self.Height + 10;
end;

procedure TForm1.ShowWelcomeText;
begin
    txtWelcome.Visible := True;
    txtWelcome.AnimateFloat('Position.Y', ((txtWelcome.Parent) as TContr
    Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  SetupWelcomeText;
  Timer1.Enabled := False;
end;
```

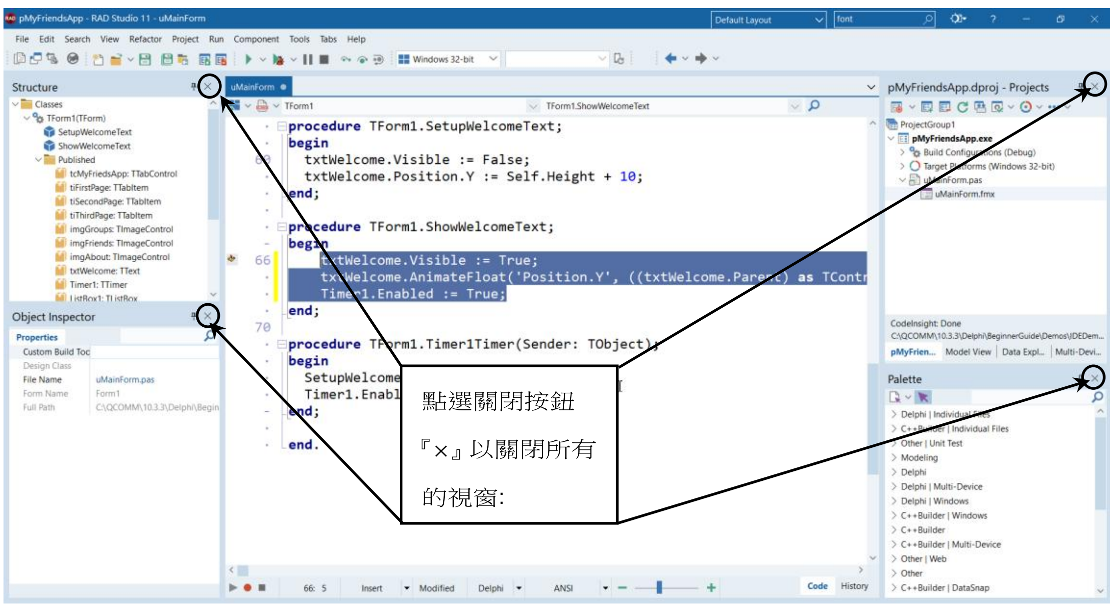
同時按下 Ctrl+Shift+I 按鍵把程式區塊往右移動

現在讓我們繼續開發這個範例應用程式，讓它更有趣一點。

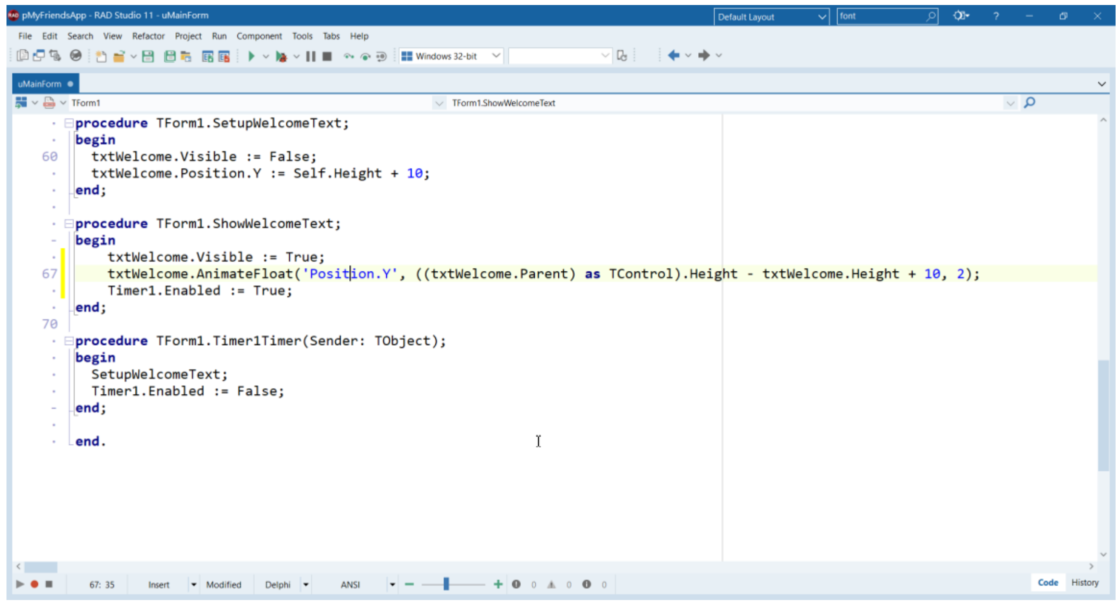
4-2 儲存/切換桌面設定

好了我們終於可以開始撰寫一些程式碼了，現在我們要集中焦點在程式碼而不是視覺化設計家，因此請按下『F12』切換回編輯器畫面，接著如下圖所示按一下『樹狀架構』，『物件檢視器』，『專案管理員』和『工具盤』視窗右上方的關閉按鈕『×』以關閉所有的視窗:

版權所有 請勿翻印



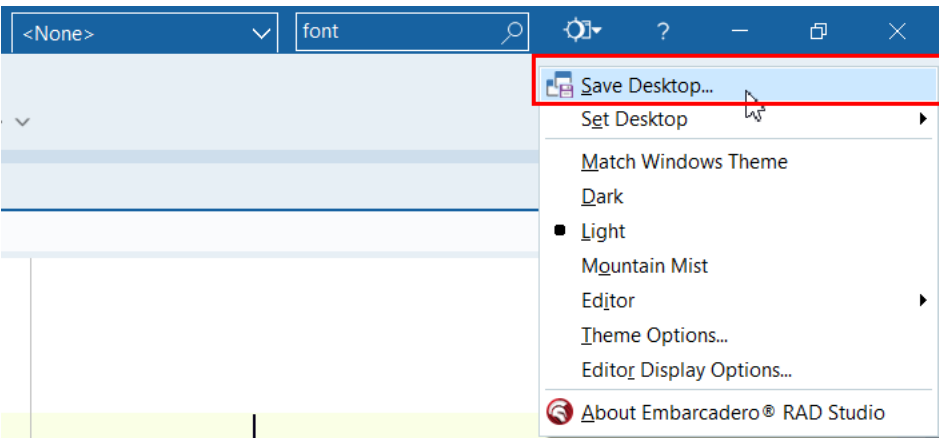
現在應該如下圖只剩下編輯器視窗:



當您在專心撰寫程式碼時，可能不希望看到其他不相關的視窗，因此可以使用上面說明的方法只使用編輯器來撰寫程式碼，但每次都需要關閉 4 個視窗非常的麻煩，因此您可以把現在只使用編輯器來撰寫程式碼的桌面組態儲存起來，那麼當下次您只需要編輯器時，就只要選擇這個組態就可以把整合發展環境回復成現在的組態。

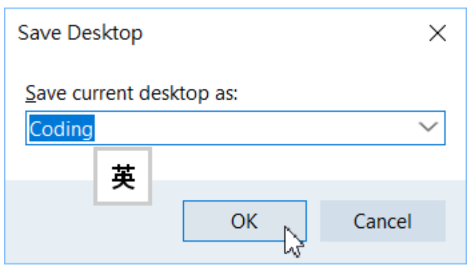
版權所有 請勿翻印

請點選整合發展環境右上方的『Save current desktop』按鈕，如下圖所示:



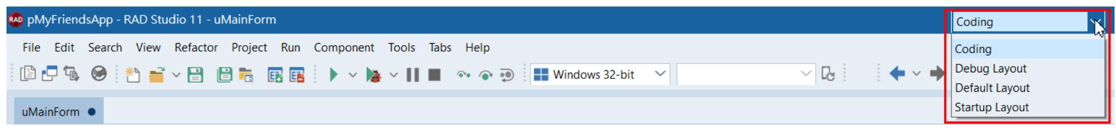
儲存目前桌面的設定組態

接著 Delphi for iOS 整合發展環境會顯示如下的對話盒，詢問您以什麼名稱儲存目前整合發展環境的組態，請輸入『Coding』以代表這個桌面組態是專門使用于撰寫程式碼時使用的:



設定儲存的桌面組態名稱為 Coding

點選上圖中的『OK』按鈕之後，現在您如果再點選整合發展環境右上方『Help』右邊的下拉盒，就可以看到如下的結果，剛才您儲存的『Coding』組態已經存在於其中:



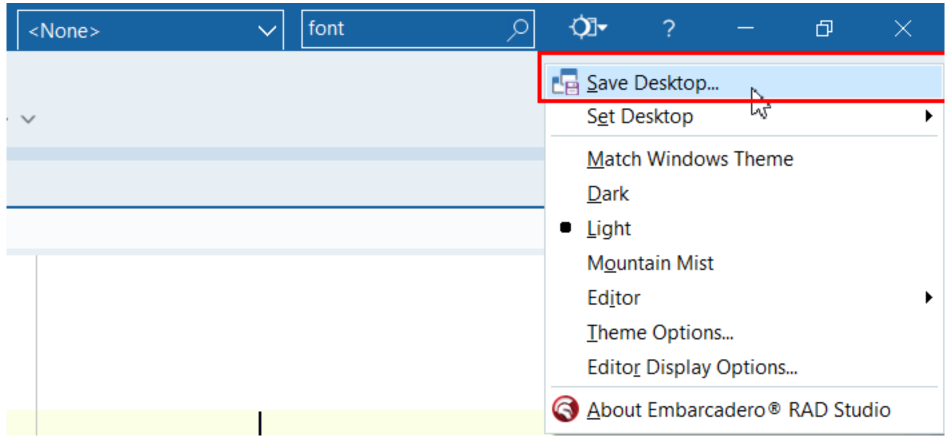
現在桌面組態下拉盒中就存在了您儲存的桌面組態名稱

請試著在此下拉盒中選擇不同的組態，例如先選擇『Default Layout』，您會發現此時整合發展環境回復到一開始 5 個不同的視窗都顯示的組態，再選擇您儲存的『Coding』組態，那麼整合發展環境又會回復到只開啟編輯器的組態了。下面的表格說明了下拉盒中不同組態的意義:

版權所有 請勿翻印

組態	說明
<None>	不使用任何設定的組態，維持目前的整合發展環境
<Startup Layout>	顯示歡迎頁面的桌面組態
<Debug Layout>	使用除錯桌面組態，稍後本書會說明如何使用除錯桌面組態
< Default Layout>	使用整合發展環境內定的桌面組態，同時開啟 5 個視窗

此外您也可以在下面的下拉盒中選擇 IDE 使用的風格或是在 Editor 項目中選擇編輯器使用的風格:

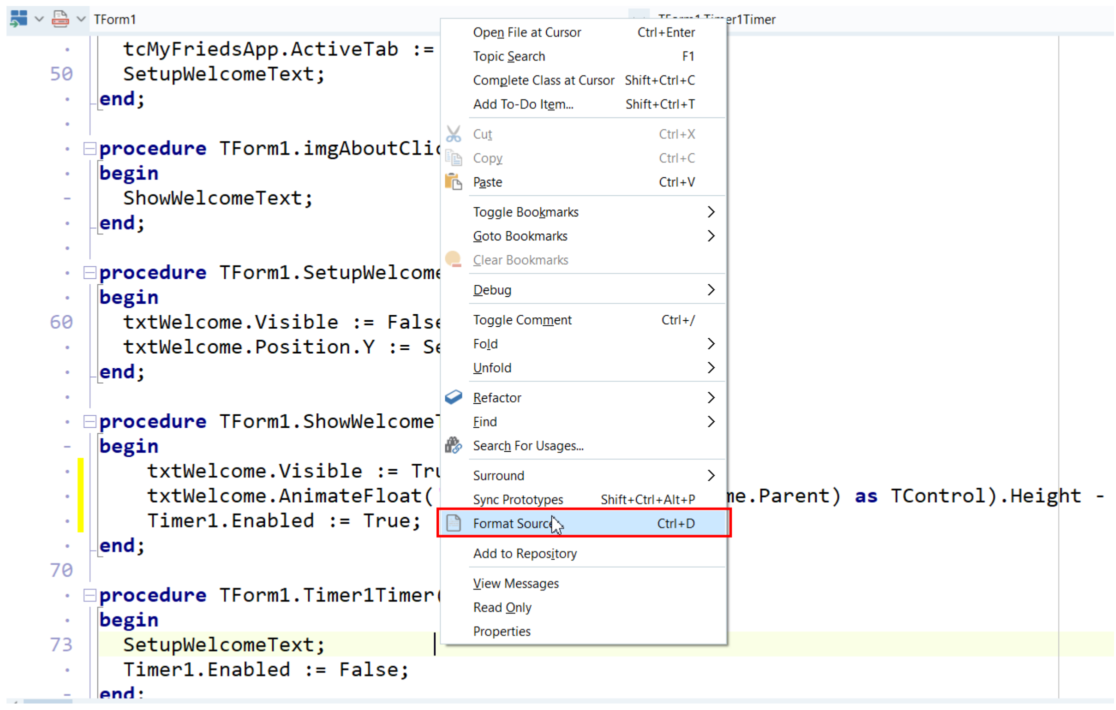


4-3 原始碼格式化

請回頭看看撰寫的程式碼，又是所有的程式碼都在編輯器的最左邊，當然您可以使用前面學習到的技巧使用『Ctrl+Shift+I』把程式碼逐一的右移，但當程式碼很多的時候這可能需要花上一些時間，在這種情形中，您可以直接使用整合發展環境的『格式化程式碼』的功能來自動格式化您的程式碼。

```
procedure TfmMainForm.FormCreate(Sender: TObject);
begin
CreateGroupDataSet;
FillGroupDataSet;
end;

procedure TfmMainForm.FormShow(Sender: TObject);
begin
tcMyFriedsApp.ActiveTab := tiFirstPage;
SetupWelcomeText;
end;

procedure TfmMainForm.imgcAboutClick(Sender: TObject);
begin
ShowWelcomeText;
end;

procedure TfmMainForm.SetupWelcomeText;
begin
txtWelcome.Visible := False;
txtWelcome.Position.Y := Self.Height + 10;
end;
```

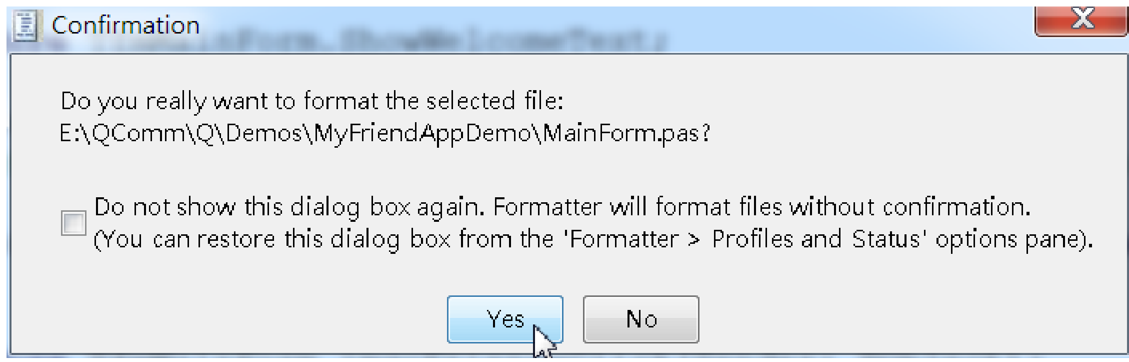
請回到編輯器中剛才實作 imgcAboutClick 的程式中，接著點選滑鼠右鍵，IDE 便會顯示一個快顯功能表，請在其中選擇『Format Source』選項，如下圖所示，或是直接在編輯器中同時按下『Ctrl+D』鍵:



在編輯器的快顯功能表中選擇『Format Source』選項

接著 IDE 會顯示下面的對話盒詢問您是否確定要格式化原始程式碼，請點選『Yes』按鈕，或是順便勾選對話盒中的勾選盒，避免每次要格式化原始程式碼是 IDE 都會詢問您。

版權所有 請勿翻印

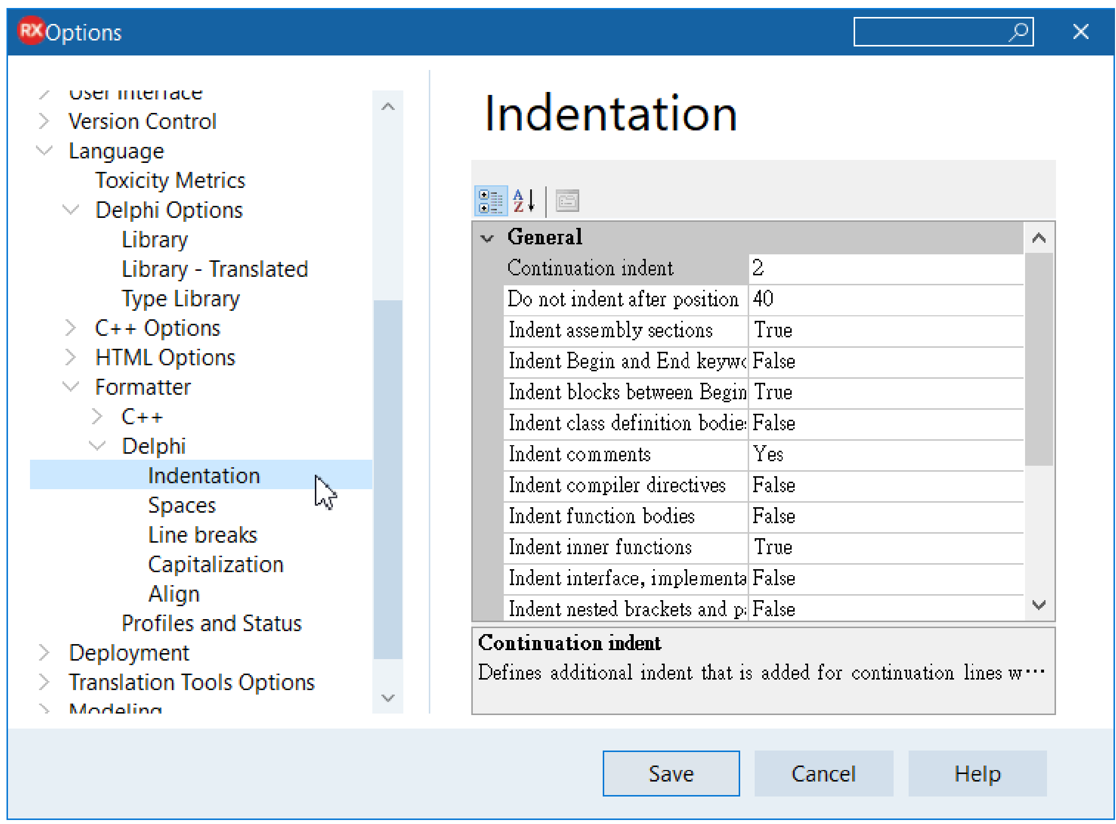


點選『Yes』按鈕之後，IDE 便會開始格式化您的原始程式碼，下圖就是格式化之後的結果，您可以看到程式碼風格好多了，也更容易明瞭，當然您也可以養成良好的習慣，在撰寫程式碼時就使用類似的風格。

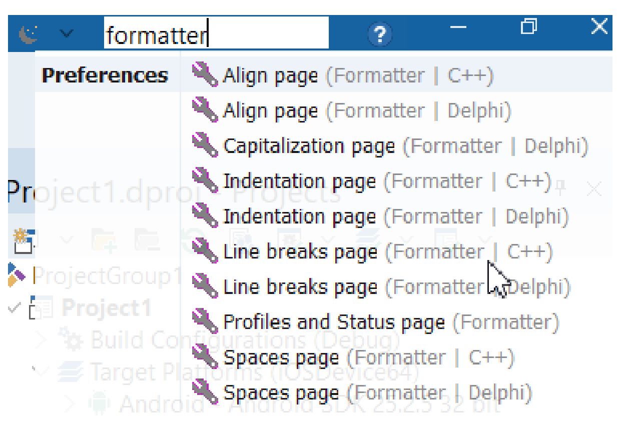
```
procedure TfmMainForm.FormCreate(Sender: TObject);
begin
  CreateGroupDataSet;
  FillGroupDataSet;
end;

procedure TfmMainForm.FormShow(Sender: TObject);
begin
  tcMyFriedsApp.ActiveTab := tiFirstPage;
  SetupWelcomeText;
end;

procedure TfmMainForm.imgcAboutClick(Sender: TObject);
begin
  ShowWelcomeText;
end;

procedure TfmMainForm.SetupWelcomeText;
begin
  txtWelcome.Visible := False;
  txtWelcome.Position.Y := Self.Height + 10;
end;
```

格式化原始程式碼功能是根據 IDE 中如何格式程式碼的設定而執行的結果，您當然也可以控制如何格式化您的程式碼，或是使用什麼風格來格式化您的程式碼。您可以點選 IDE 上方功能表中的 Tools|Options...選項啟動 Options 對話盒，然後在其中找尋 Formatter|Delphi 選項，在這個選項之中有數 10 個如何格式化原始程式碼的設定，您可以使用這些設定來定義您自己喜歡使用的風格，如下所示:

版權所有 請勿翻印



或是在 IDE 中按下 F6，在 IDE Insight 對話盒中直接輸入 formatter 就可以搜尋到格式化原始程式碼的設定選項，如下所示:

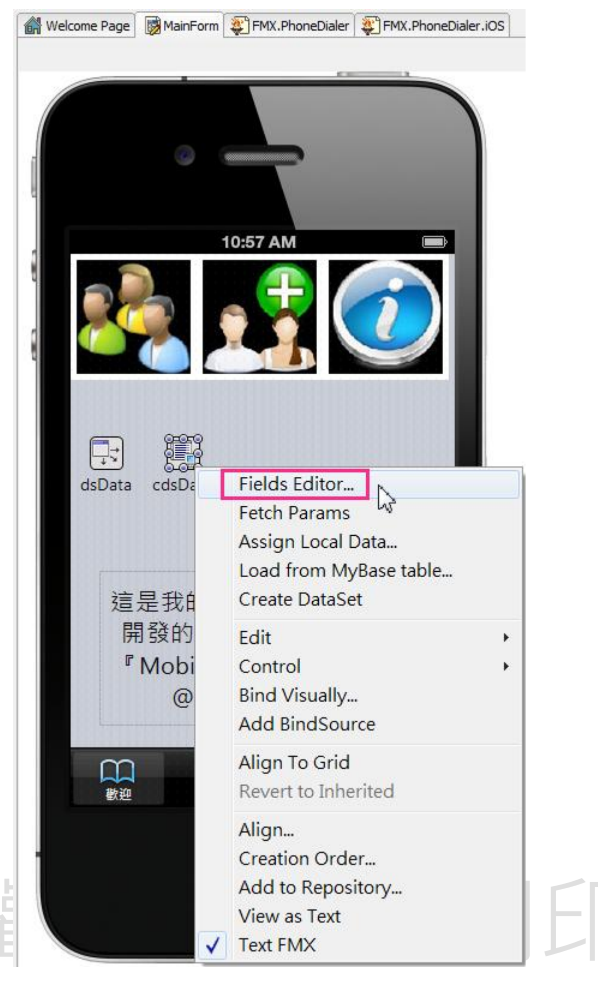


現在您就可以試著改變一些設定然後再次重新格式化您的原始程式碼，看看改變這些設定之後有什麼效果了。

讓我們為這個範例 iOS 加入一些更為複雜和有趣的功能，那就是開始加入存取資料的能力。如果您是使用其他 iOS 開發工具而需要為 iOS App 加入資料處理的功能的話，您需要花費許多的時間去撰寫大量的程式碼，但使用 Delphi for iOS 卻非常的簡單。

版權所有 請勿翻印

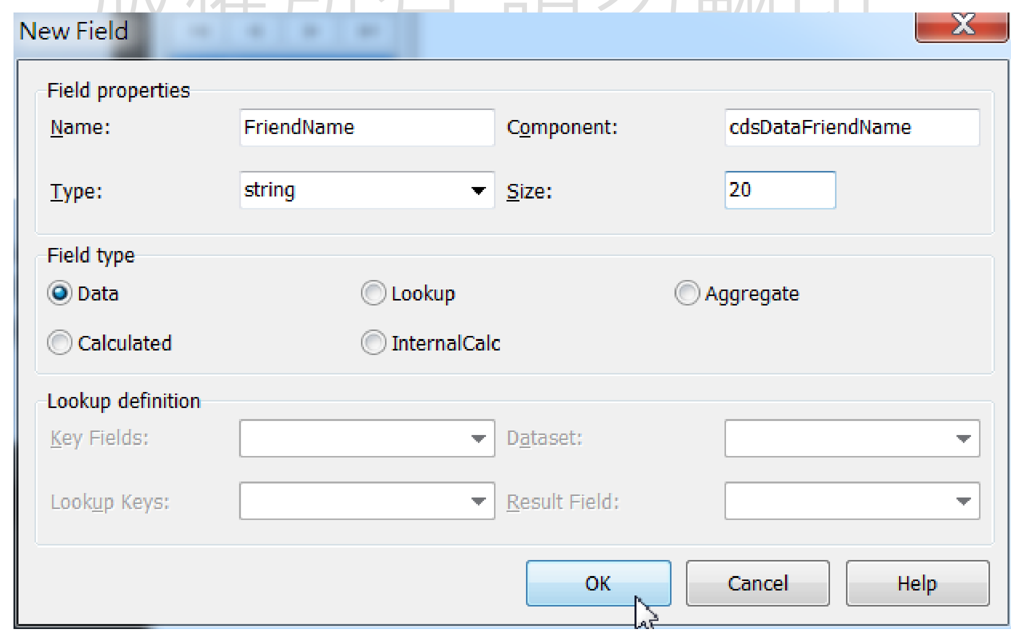
首先請在主表單中加入 TClientDataSet 設定它的 Name 特性值為 cdsData，加入 TDataSource 組件，設定它的 Name 特性值為 dsData。接著點選主表單中的 cdsData，點選滑鼠右鍵從快顯功能表中選擇『Fields Editor…』選項以啟動欄位編輯器，我們將使用它為 cdsData 加入兩個欄位。現在主表單應該看起來類似如下:

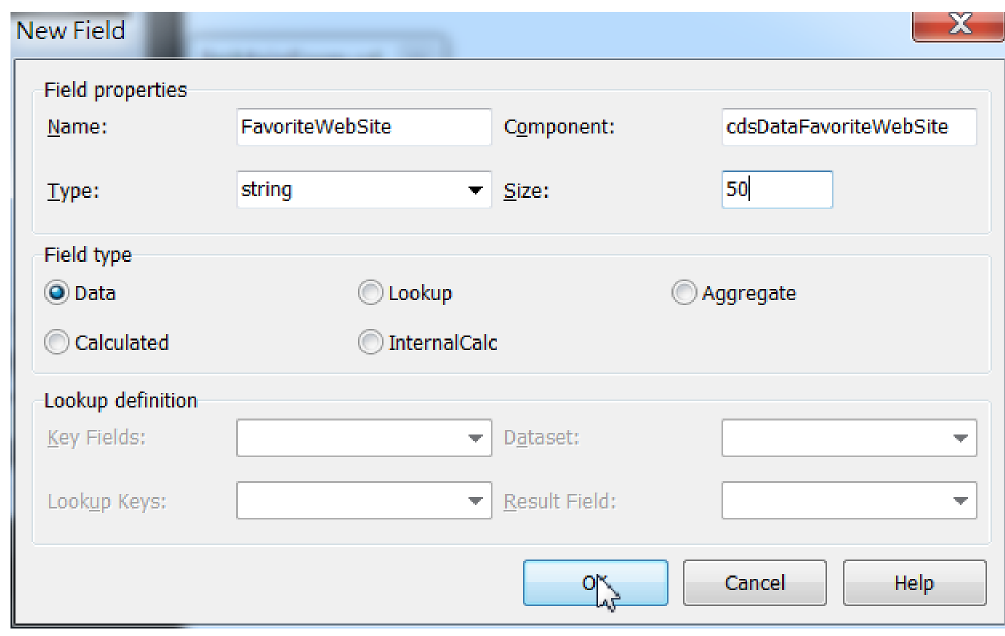


在 cdsData 的欄位編輯器啟動之後，再於其中點選滑鼠右鍵，從快顯功能表中選擇『New field…』選項以加入欄位物件，如下所示:

接著使用 **New Field** 對話盒加入如下的兩個欄位物件:

版權所有 請勿翻印





現在 cdsData 組件就擁有 FriendName 和 FavoriteWebSite 這兩個字串欄位物件了，接著切換到編輯器，在程式碼中加入的兩個方法 CreateGroupDataSet 和 FillGroupDataSet:

```
procedure TfmMainForm.CreateGroupDataSet;
var
  aField : TFieldDef;
  anIndex : TIndexDef;
begin
  anIndex := cdsData.IndexDefs.AddIndexDef;
  anIndex.Fields := 'FriendName';
  anIndex.Name := 'idxFriendName';

  cdsData.CreateDataSet;
  cdsData.Active := True;
end;

procedure TfmMainForm.FillGroupDataSet;
begin
  cdsData.Insert;
  cdsData.FieldByName('FriendName').AsString := 'Jackson Wang';
  cdsData.FieldByName('FavoriteWebSite').AsString :=
'www.youtube.com';
  cdsData.Post;
```

```
  cdsData.Insert;
  cdsData.FieldByName('FriendName').AsString := 'Hua Lee';
  cdsData.FieldByName('FavoriteWebSite').AsString :=
'www.embarcadero.com';
  cdsData.Post;

  cdsData.Insert;
  cdsData.FieldByName('FriendName').AsString := 'DongHseng Cheng';
  cdsData.FieldByName('FavoriteWebSite').AsString := 'www.msn.com.tw';
  cdsData.Post;

  cdsData.Insert;
  cdsData.FieldByName('FriendName').AsString := 'YuHei Chu';
  cdsData.FieldByName('FavoriteWebSite').AsString :=
'Delphi.ktop.com.tw';
  cdsData.Post;
end;
```

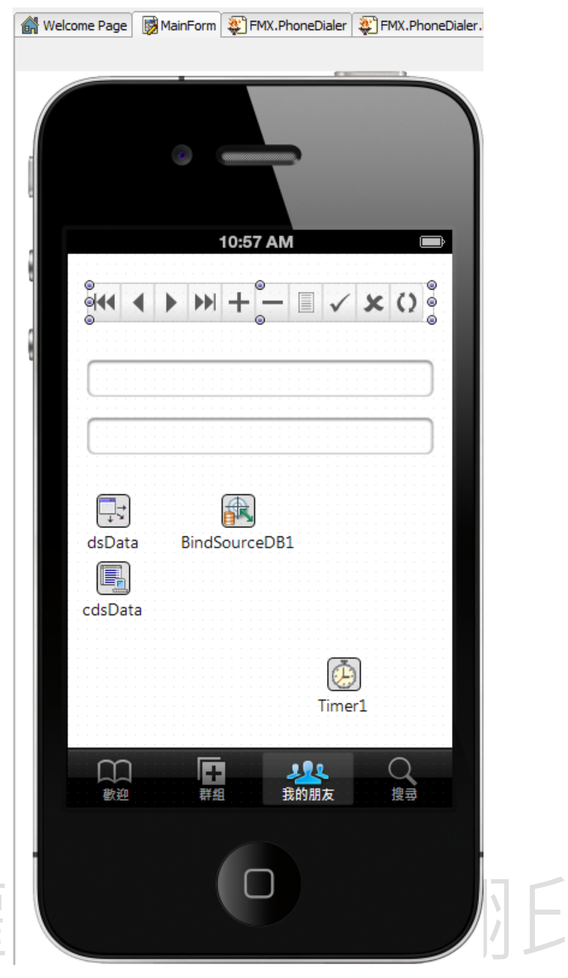
CreateGroupDataSet 在 cdsData 中加入一個索引物件，再呼叫 CreateDataSet 方法建立 cdsData 中的資料集。而 FillGroupDataSet 則是在 cdsData 元件中新增一些資料。

版權所有 請勿翻印

CreateGroupDataSet 在 cdsData 中加入一個索引物件，再呼叫 CreateDataSet 方法建立 cdsData 中的資料集。而 FillGroupDataSet 則是在 cdsData 元件中新增一些資料。

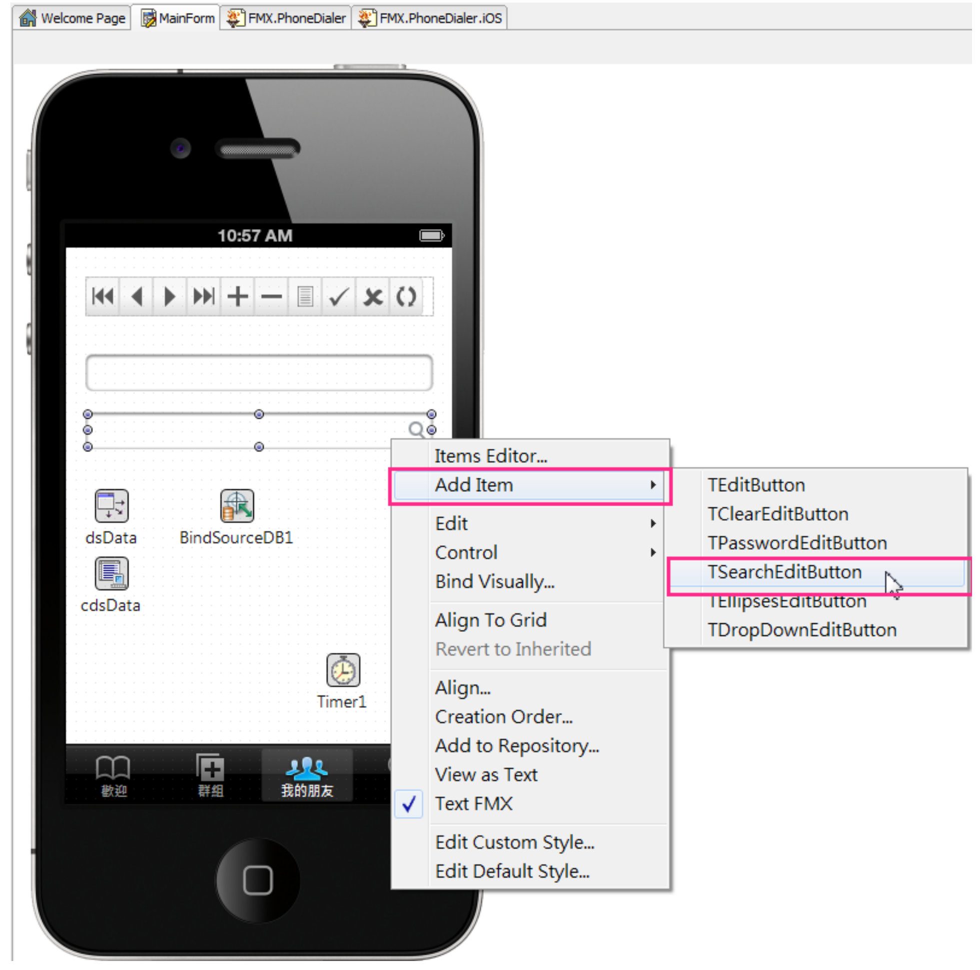
```
procedure TfmMainForm.FormCreate(Sender: TObject);
begin
  CreateGroupDataSet;
  FillGroupDataSet;
end;
```

接著切換到主表單中 TTabControl 元件的第 3 個頁次，在其中加入 TBindNavigator 組件，設定它的 DataSource 特性值為 cdsData，再加入兩個 TEdit 組件，如下所示:

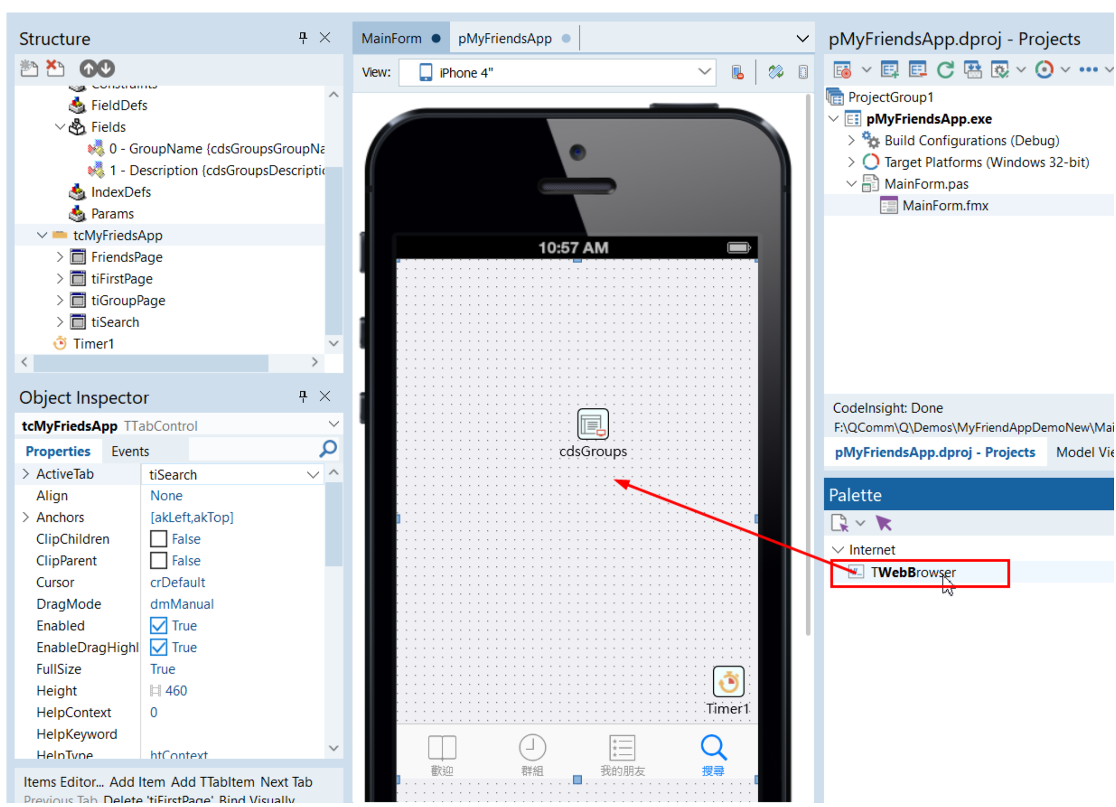


版權

再讓我們為剛才加入的第二個 TEdit 元件加入一個搜尋按鈕，請點選第二個 TEdit 元件，再點選滑鼠右鍵，從快顯功能表中選擇『Add Item』選項，再從其中選擇『TSearchEditButton』，如下所示:

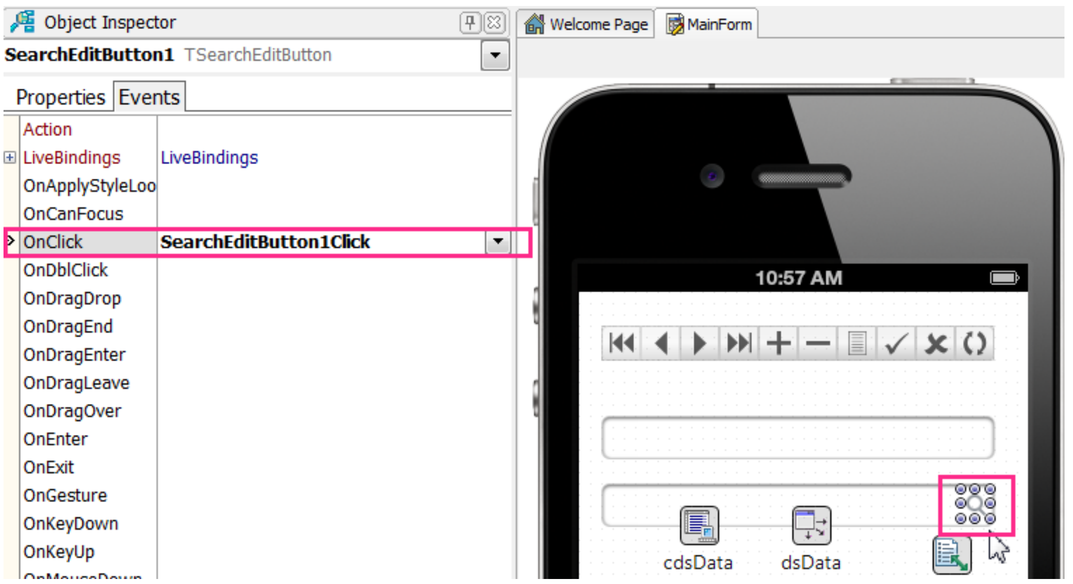


切換到 TTabControl 元件的第 4 個頁次，在工具盤中搜尋 TWebBrowser 元件再拖曳到第 4 個頁次中，設定 TWebBrowser 組件的 Align 特性值為 alClient，如下所示:



最後再回到 TTabControl 組件的第 3 個頁次為 TSearchEditButton 組件建立 OnClick 事件處理函式，如下所示:

版權所有 請勿翻印




```
procedure TfmMainForm.SearchEditButton1Click(Sender: TObject);
begin
  //
end;
```

接下來我們就可以開始試著系結 cdsData 中的資料和主表單中的視覺化在一起了，但在這之前讓我們再學習數個整合發展環境的技巧。

4-4 SyncEdit

在繼續改善程式碼之前，您需要學習一個非常有用的編輯器技巧，那就是所謂的 SyncEdit。

首先請回到主表單把原先的 TClientDataSet 的名稱 Name 特性值從 cdsData 改變成 cdsMyData。由於我們改變了元件名稱因此程式碼中使用的 cdsData 物件變數名稱也必須修改。但如果我們到 FillGroupDataSet 方法會發現其中使用了多次 cdsData，因此我們必須進行多次的修改，這實在非常的麻煩而且容易出錯:

```
procedure TfmMainForm.FillGroupDataSet;
begin
  cdsData.Insert;
  cdsData.FieldByName('FriendName').AsString := 'Jackson Wang';
  cdsData.FieldByName('FavoriteWebSite').AsString :=
'www.youtube.com';
  cdsData.Post;
  ...
```

這個時候就是使用 SyncEdit 的好時機，現在就讓我們使用 SyncEdit 來修改 cdsData 為 cdsMyData。

首先請如下圖使用滑鼠或是鍵盤選擇整個使用 cdsData 物件變數名稱的程式碼部份(使用滑鼠或是使用 Shift+↓鍵)，請注意此時在編輯器最左方會出現一個類似雙鉛筆的圖像，如下所示:

```
98    cdsData.CreateDataSet;
      cdsData.Active := True;
100 end;

    procedure TfmMainForm.FillGroupDataSet;
    begin
      cdsData.Insert;
      cdsData.FieldByName('FriendName').AsString := 'Jackson Wang';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'www.youtube.com';
      cdsData.Post;

      cdsData.Insert;
110   cdsData.FieldByName('FriendName').AsString := 'Hua Lee';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'www.embarcadero.com';
      cdsData.Post;

      cdsData.Insert;
      cdsData.FieldByName('FriendName').AsString := 'DongHseng Cheng';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'www.msn.com.tw';
      cdsData.Post;

      cdsData.Insert;
120   cdsData.FieldByName('FriendName').AsString := 'YuHei Chu';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'Delphi.ktop.com.tw';
      cdsData.Post;
    end;
```

Sync Edit Mode (Shift+Ctrl+J)

選擇程式碼並且啟動 SyncEdit 功能

現在請使用滑鼠按兩下編輯器中圖像，此時編輯器會立刻把整個使用 cdsData 物件變數名稱的程式碼部份中所有的變數標示出來，由於 cdsData 是第 1 個變數，因此它是以方框標示，此時所有的 cdsData 變數中的第 1 個 cdsData 更以方框反白標示，如下圖所示:

版權所有 請勿翻印

```
98    cdsData.CreateDataSet;
      cdsData.Active := True;
100 end;

    procedure TfmMainForm.FillGroupDataSet;
    begin
      cdsData.Insert;
      cdsData.FieldByName('FriendName').AsString := 'Jackson Wang';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'www.youtube.com';
      cdsData.Post;

      cdsData.Insert;
110   cdsData.FieldByName('FriendName').AsString := 'Hua Lee';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'www.embarcadero.com';
      cdsData.Post;

      cdsData.Insert;
      cdsData.FieldByName('FriendName').AsString := 'DongHseng Cheng';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'www.msn.com.tw';
      cdsData.Post;

      cdsData.Insert;
120   cdsData.FieldByName('FriendName').AsString := 'YuHei Chu';
      cdsData.FieldByName('FavoriteWebSite').AsString := 'Delphi.ktop.com.tw';
      cdsData.Post;
    end;
```

Undeclared identifier 'cdsData'

現在請您直接在編輯器中輸入 cdsMyData，請注意這時編輯器中所有變數 cdsData 都立刻修改為 cdsMyData，如下所示:

```
  cdsMyData.CreateDataSet;
  cdsMyData.Active := True;
end;

procedure TfmMainForm.FillGroupDataSet;
begin
  cdsMyData.Insert;
  cdsMyData.FieldByName('FriendName').AsString := 'Jackson Wang';
  cdsMyData.FieldByName('FavoriteWebSite').AsString := 'www.youtube.com';
  cdsMyData.Post;

  cdsMyData.Insert;
  cdsMyData.FieldByName('FriendName').AsString := 'Hua Lee';
  cdsMyData.FieldByName('FavoriteWebSite').AsString := 'www.embarcadero.com';
  cdsMyData.Post;

  cdsMyData.Insert;
  cdsMyData.FieldByName('FriendName').AsString := 'DongHseng Cheng';
  cdsMyData.FieldByName('FavoriteWebSite').AsString := 'www.msn.com.tw';
  cdsMyData.Post;

  cdsMyData.Insert;
  cdsMyData.FieldByName('FriendName').AsString := 'YuHei Chu';
  cdsMyData.FieldByName('FavoriteWebSite').AsString := 'Delphi.ktop.com.tw';
  cdsMyData.Post;
end;
```

如果此時您不斷的按下『Tab』鍵，整個使用 **cdsData** 物件變數名稱的程式碼部份中會逐一的以方框反白標示每一個可修改的變數，類別名稱或是方法名稱，例如您第 1 次按下『Tab』鍵方框反白標示就會停駐在變數 **Insert** 上，再按下一次就會停駐在類別 **FieldByName** 上，您就可以像剛才修改 **cdsData** 為 **cdsMyData** 一樣改變 **Insert** 或是 **FieldByName** 的名稱。

版權所有 請勿翻印

最後要關閉 **SyncEdit** 功能，您只需要再次使用滑鼠點選圖像，或是直接振下 **ESC** 鍵即可。

4-5 程式碼重構(Refactor)

再把焦點放在 **FillGroupDataSet** 程式，我們會發現在其中擁有重複的程式碼，這些重複的程式碼只有傳遞的參數不同，因此我們應該對它們進行一些簡單的重構以簡潔化這些重複的程式碼。

這些重複的程式碼都是以:

```
cdsMyData.Insert;
cdsMyData.FieldByName('FriendName').Value := 'Jackson Wang';
cdsMyData.FieldByName('FavoriteWebSite').Value := 'www.youtube.com';
cdsMyData.Post;
```

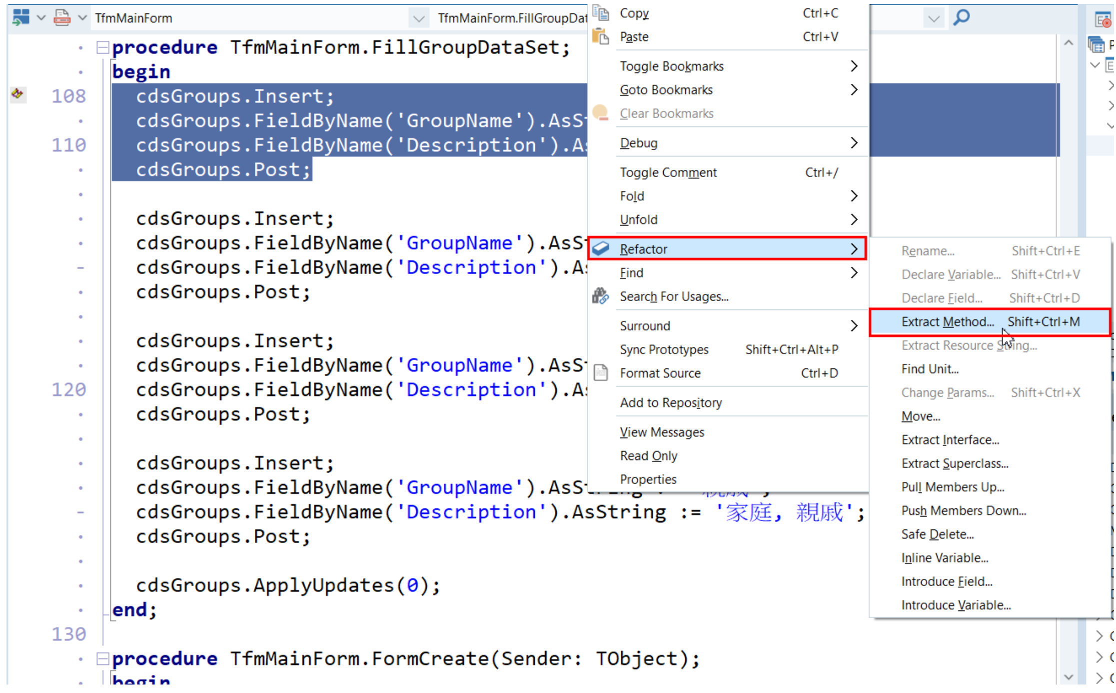
程式區塊組成，如果我們可以簡單的使用

```
InsertData('Jackson Wang', 'www.youtube.com');
```

來代替就好了。

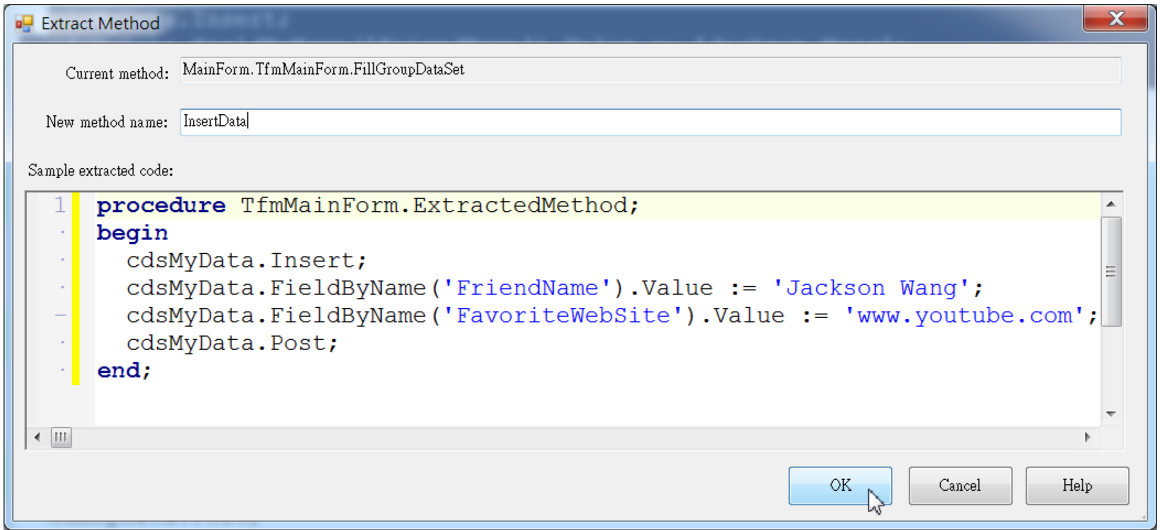
當然我們可以自行在編輯器中修改這些程式碼，但 Delphi for iOS 整合發展環境提供了重構功能可以幫助我們簡化這個工作。

現在請使用滑鼠或鍵盤選擇 FillGroupDataSet 中的程式碼，然後右擊滑鼠右鍵，編輯器會顯示快顯功能表，請選擇其中的 Refactor 選項，再從 Refactor 選項顯示的快顯功能表中選擇『Extract Method...』選項，如下所示:



或著您也可以直接點選 IDE 上方的 Refactor | Extract Method...功能表，或是同時按下『Shift+Ctrl+M』3 個鍵。

接著 IDE 會顯示 Extract Method 對話盒，對話盒下方會顯示所有會被重構的程式碼，當然也就是您選擇的程式碼，在對話盒上方會請您為新擷取出來的程式碼定義一個方法名稱，如下所示，請在 New method name 欄位中輸入 InsertData 如下所示:



點選對話盒之中的『OK』按鈕之後，IDE 會立刻的重構您的程式碼，最後的結果如下所示:

```
procedure TfmMainForm.FillGroupDataSet;
begin
  InsertData;


  cdsMyData.Insert;
  ...
end;


procedure TfmMainForm.InsertData;
begin
  cdsMyData.Insert;
  cdsMyData.FieldByName('FriendName').Value := 'Jackson Wang';
  cdsMyData.FieldByName('FavoriteWebSite').Value := 'www.youtube.com';
  cdsMyData.Post;
end;
```

接著修改 InsertData 讓它接受兩個參數，然後我們就可以修改 FillGroupDataSet 成一個只需 4 行程式碼的程式了，FillGroupDataSet 程式變得相當的簡潔。

```
procedure TfmMainForm.FillGroupDataSet;
begin
  InsertData('Jackson Wang', 'www.youtube.com');
  InsertData('Hua Lee', 'www.embarcadero.com');
  InsertData('DongHseng Cheng', 'www.msn.com.tw');
```

```
  InsertData('YuHei Chu', 'Delphi.ktop.com.tw');
end;

procedure TfmMainForm.InsertData(const sName : String; const sWebSite :
String);
begin
  cdsMyData.Insert;
  cdsMyData.FieldByName('FriendName').Value := sName;
  cdsMyData.FieldByName('FavoriteWebSite').Value := sWebSite;
  cdsMyData.Post;
end;
```

在讀者使用 Delphi for iOS 開發時也可以多多使用整合發展環境提供的程式碼重構功能來精簡程式碼。

4-6 待辦清單(To-Do List)

在撰寫程式碼時，您可以先在程式碼中列出待辦清單，接著一一的撰寫程式碼實現這些待辦清單，您也可以追蹤，管理程式碼中的待辦清單以便瞭解您是否完成了所有的待辦清單，以免遺漏應該完成的功能，這在撰寫大量的功能和程式碼時是非常實用的功能。

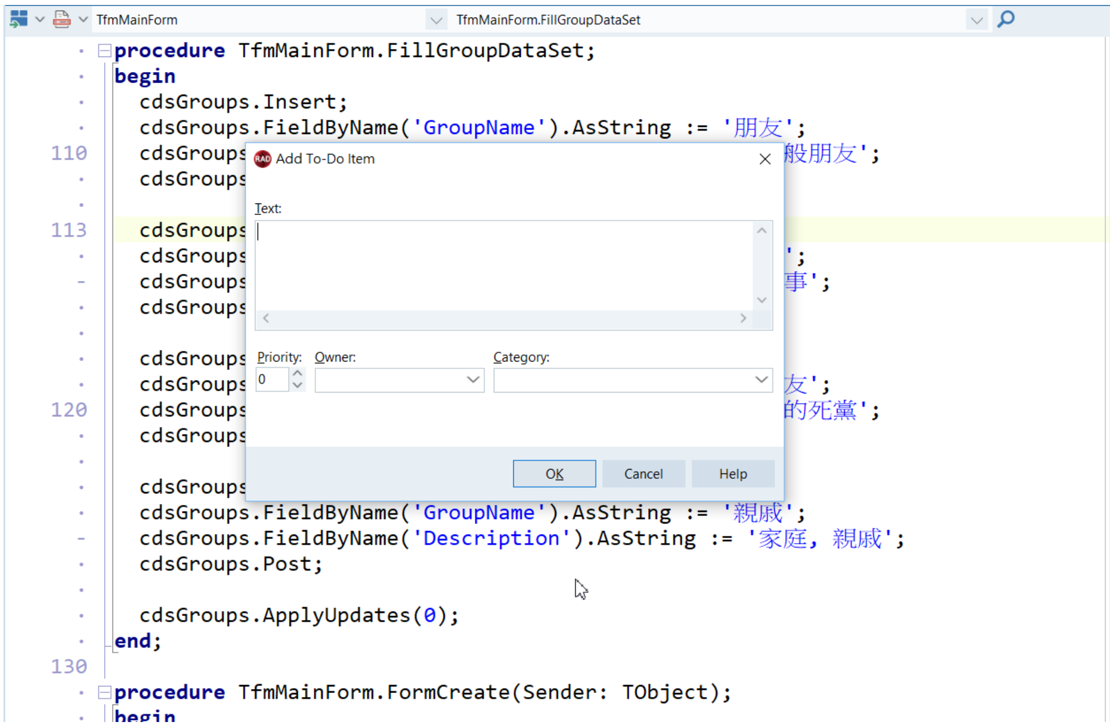
版權所有 請勿翻印

例如現在 TSearchEditButton 元件的 OnClick 事件處理函式還沒有實作，但我們可以在編輯器加入這些待辦清單，並且追蹤這些待辦清單以便我們最終能夠完成這些功能。

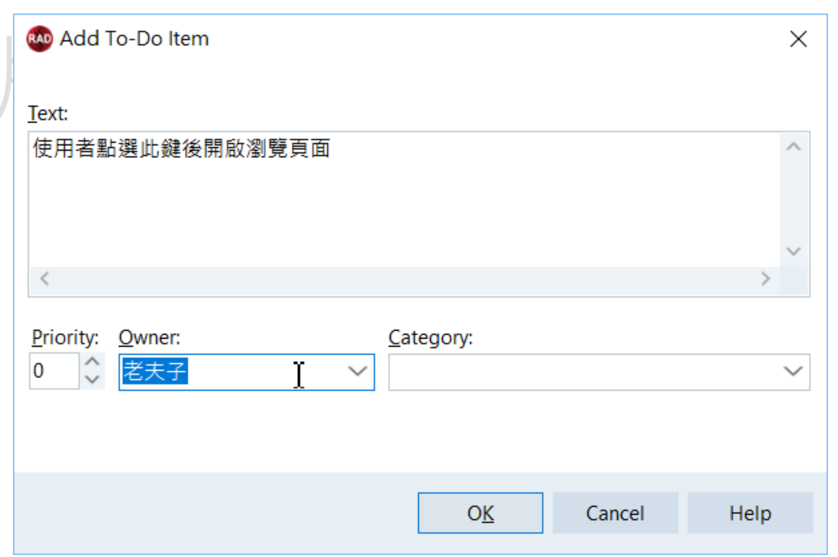
要在編輯器中加入待辦清單，您可以在程式碼中適當的地方同時按下 Shift+Ctrl+T，或是在編輯器中點選滑鼠右鍵，在快顯功能表中選擇『Add To-Do item...』選項。例如 TSearchEditButton 元件建立 OnClick 事件處理函式尚未實作，因此讓我們移動滑鼠到 SearchEditButton1Click 程式的第 1 行程式碼之上，同時按下 Shift+Ctrl+T 準備加入待辦清單，此時 IDE 就會顯示一個 Add To-Do Item 對話盒如下，請您輸入待辦清單的詳細資訊，對話盒中欄位的意義說明如下:

欄位	說明
Text	待辦列表說明文字
Priority	待辦清單優先次序
Owner	待辦清單負責人
Category	待辦清單歸納分類

例如下圖就是在 SearchEditButton1Click 程式中啟動待辦清單功能:



接著讓我們在 Add To-Do Item 對話盒中輸入如下的資訊:



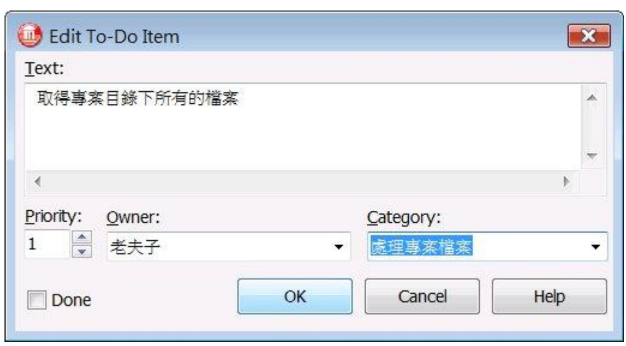
當您點選 Add To-Do Item 對話盒中的 OK 按鈕之後，您就可以在程式碼中看到 IDE 在您的程式碼中加入了如下的待辦清單注釋:

```
procedure TfmMainForm.SearchEditButton1Click(Sender: TObject);
begin
  { TODO -o老夫子 : 使用者點選此鍵後開啟流覽器頁面 }
end;
```

接著請您在 btnGetFilesClick 程式中使用剛才相同的方法加入另外一個待辦列表，這個待辦列表是壓縮專案檔案的待辦事項。完成了之後您可以點選 IDE 上方的 View|To-Do List 功能表，您就可以看到類似如下的畫面，IDE 可以顯示所有您的待辦事項，當您按兩下其中任何的待辦事項時，編輯器就會移動到您的程式碼到對應的待辦事項和程式碼處，如下所示:

```
MainForm   pMyFriendsApp
TfmMainForm                    TfmMainForm.FillGroupDataSet
      cdsGroups.Insert;
      cdsGroups.FieldByName('GroupName').AsString := '好朋友';
120   cdsGroups.FieldByName('Description').AsString := '我的死黨';
      cdsGroups.Post;

      cdsGroups.Insert;
      cdsGroups.FieldByName('GroupName').AsString := '親戚';
      cdsGroups.FieldByName('Description').AsString := '家庭, 親戚';
      cdsGroups.Post;

      cdsGroups.ApplyUpdates(0);
    end;
130
    procedure TfmMainForm.FormCreate(Sender: TObject);
    begin
    //   CreateGroupDataSet;
    //   FillGroupDataSet;
    end;

    procedure TfmMainForm.FormDestroy(Sender: TObject);
    begin
113: 78   Insert   Modified   Delphi   ANSI      Code   Design   History
To-Do List - pMyFriendsApp
Action Item                    !  Module              Owner    Category
使用者點選此鍵後開啟瀏覽頁面      F:\QCom...\MainForm.pas  老夫子
```

當然您也可以修改待辦事項，現在請使用滑鼠點選 IDE 下方的『取得專案目錄下所有的檔案』待辦事項，點選滑鼠右鍵，從快顯功能表中選擇『Edit...』選項，或是直接按下 F2 鍵，那麼 Edit To-Do Item 對話盒就會顯示，您就可以進行修改，例如我們增加了『待辦清單優先次序』和『待辦清單歸納分類』資訊，如下圖所示。

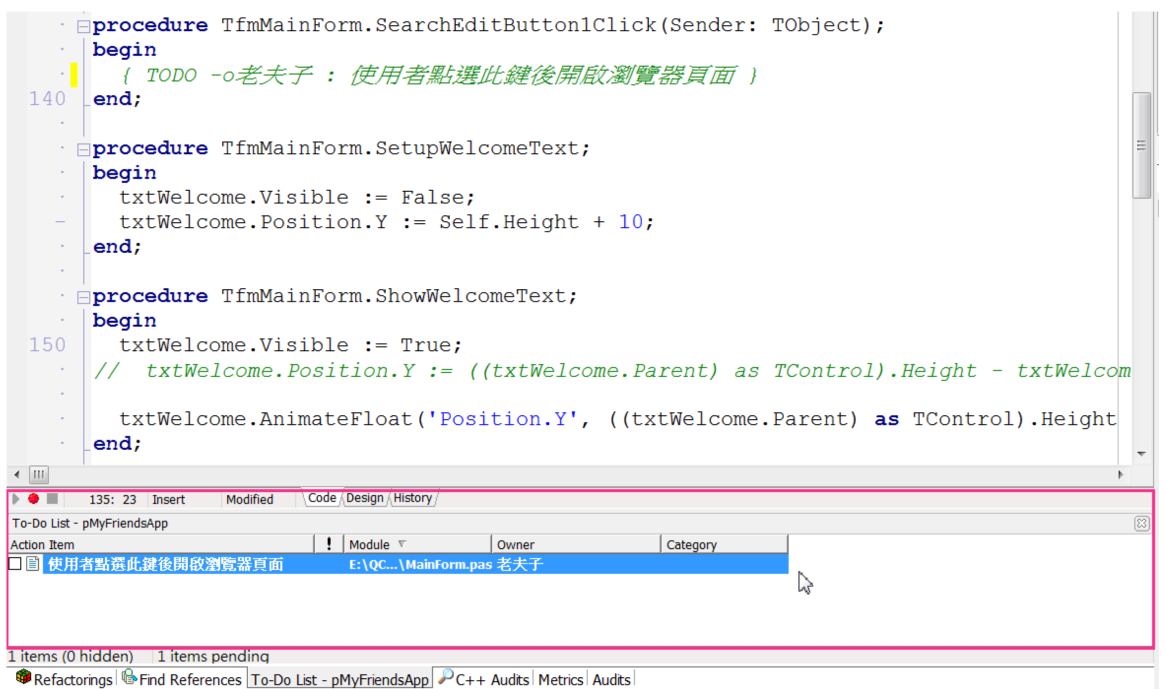


點選 OK 按鈕之後程式碼就會被修改如下:

```
{ TODO 1 -c處理專案檔案 -o老夫子 : 取得專案目錄下所有的檔案 }
```

```
 filesarray :=
TDirectory.GetFileSystemEntries(TDirectory.GetCurrentDirectory +
'/../..');
```

如果您點選 View|To-Do List 功能表就會在整合發展環境中看到 To-Do List 視窗，其中會顯示所有程式碼中的待辦事項列表:



當然，如果我們完成了某項待辦清單，那麼我們可以更正待辦事項為『完成』的狀態，我們只需要勾選待辦事項最左方的勾選盒，那麼 IDE 就會修改程式碼中的待辦事項為 DONE，如下圖所示:

```
procedure TfmMainForm.SearchEditButton1Click(Sender: TObject);
begin
  { DONE -o老夫子 : 使用者點選此鍵後開啟瀏覽器頁面 }
end;

procedure TfmMainForm.SetupWelcomeText;
begin
  txtWelcome.Visible := False;
  txtWelcome.Position.Y := Self.Height + 10;
end;

procedure TfmMainForm.ShowWelcomeText;
begin
  txtWelcome.Visible := True;
//  txtWelcome.Position.Y := ((txtWelcome.Parent) as TControl).Height - txtWelcom

  txtWelcome.AnimateFloat('Position.Y', ((txtWelcome.Parent) as TControl).Height
end;
```

```
149: 42   Insert   Modified   Code  Design  History
To-Do List - pMyFriendsApp
Action Item | ! | Module | Owner | Category
使用者點選此鍵後開啟瀏覽器頁面   E:\QC...\MainForm.pas 老夫子
1 items (0 hidden)   0 items pending
```

4-7 程式區塊批註

在 Delphi for iOS 程式語言中，您可以使用『//』批註單行程式碼的意義，或是使用『{...}』來批註多行程式碼的意義。在 IDE 中如果您想批註多行的程式碼，您可以使用滑鼠選擇想批註的程式碼，然後同時按下『Ctrl+/』，那麼 IDE 就會自動幫您批註這些程式碼，如下所示:

版權所有 請勿翻印

```
procedure TfmMainForm.InsertData(const sName : String; const sWebSite : String);
begin
//  cdsMyData.Insert;
//  cdsMyData.FieldByName('FriendName').Value := sName;
//  cdsMyData.FieldByName('FavoriteWebSite').Value := sWebSite;
//  cdsMyData.Post;
end;
```

如果您想取消批註的程式碼，那麼同樣的您只要選擇已經批註程式碼，再同時按下『Ctrl+/』就可以取消選擇的程式碼的批註了。

完成類別功能(Class Completion) — CTRL+SHIFT+C

使用使用 Delphi for iOS 開發時，我們會需要撰寫許多的程式碼，也經常需要宣告和實作程式，因此整合發展環境提供了完成類別功能，開發人員只需要在類別宣告部份撰寫程式原型，再使用完成類別功能那麼編輯器就會自動說明您產生程式程式區塊。例如在範例 App 中如果我們需要一個 SearchPerson 程式，那麼我們只需要如下所示在類別中宣告 SearchPerson 程式的原型:

```
private
  { Private declarations }
  procedure SetupWelcomeText;
  procedure ShowWelcomeText;
  procedure CreateGroupDataSet;
  procedure FillGroupDataSet;
  procedure InsertData(const sName : String; const sWebSite : String);
  procedure SearchPerson(const sName : String);
```

接著同時按下『CTRL+SHIFT+C』3 個鍵，IDE 就會啟動 Class Completion 功能，在您的程式碼中自動加入 SearchPerson 程式，如下所示:

```
procedure TfmMainForm.SearchPerson(const sName: String);
begin

end;
```

4-8 錯誤洞察(Error Insight)

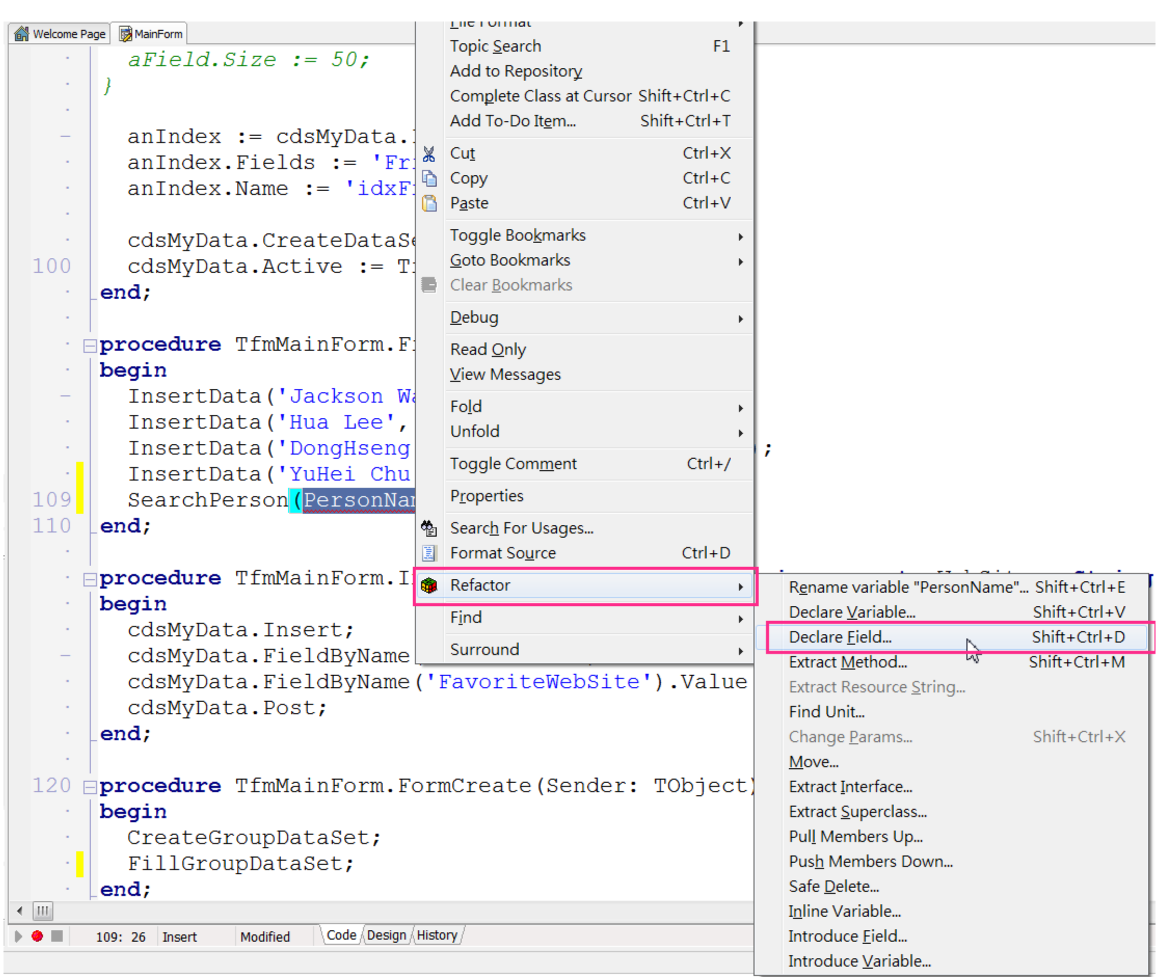
如果我們在 FillGroupDataSet 程式中撰寫一行呼叫 SearchPerson 的程式碼如下所示:

```
procedure TfmMainForm.FillGroupDataSet;
begin
  InsertData('Jackson Wang', 'www.youtube.com');
  InsertData('Hua Lee', 'www.embarcadero.com');
  InsertData('DongHseng Cheng', 'www.msn.com.tw');
  InsertData('YuHei Chu', 'Delphi.ktop.com.tw');
  SearchPerson(PersonName);
end;
```

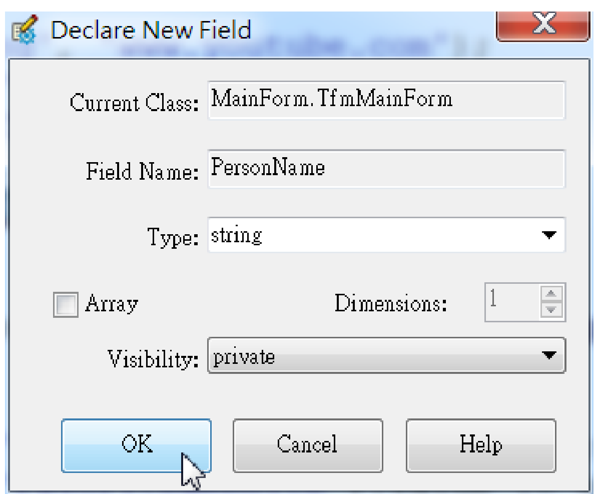
由於呼叫程式時傳入的變數 PersonName 沒有宣告和定義，因此整合發展環境會在 PersonName 變數下方顯示一排紅色的波浪線，如果您把滑鼠移動到『PersonName』變數上方，IDE 就會顯示如上來『Undeclared identifier 'PersonName』資訊。而『PersonName』變數下方的紅線稱為『錯誤洞察』，這是因為『PersonName』變數尚未宣告在程式碼，因此『錯誤洞察』以紅線提醒您，現在就讓我們宣告『PersonName』變數以修正這個錯誤。

4-9 宣告類別變數(Declare Field)

現在請把游標移動到『PersonName』上，接著點選滑鼠右鍵，從快顯功能表中選擇『Refactor』，再選擇『Declare Field...』選項，如下所示，或是直接同時按下『Shift+Ctrl+D』:



接著 IDE 會顯示『Declare New Field』對話盒如下，它將在類別中宣告 PersonName 變數，型態為 String，請點選『OK』按鈕接受:



點選 OK 按鈕之後，IDE 就會在類別的 **private** 部份加入如下的程式碼:

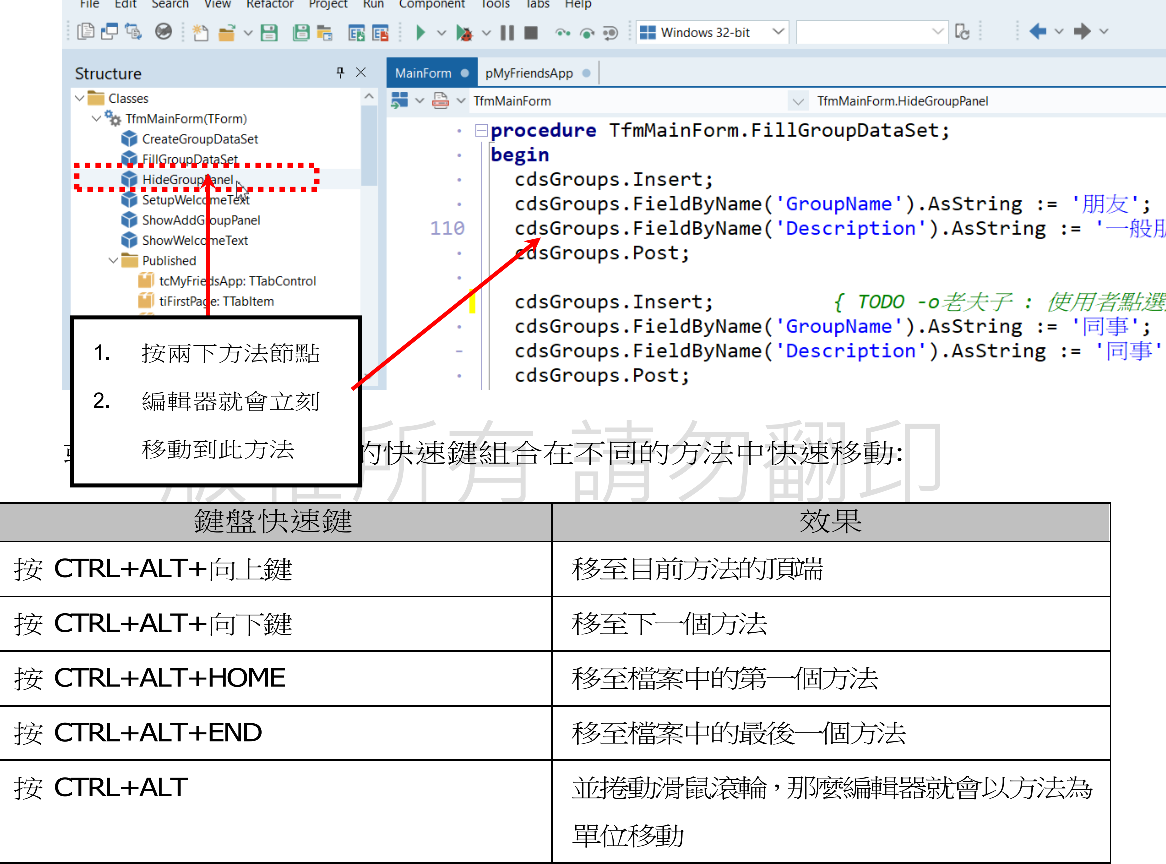
```
private
```

```
PersonName: string;
```

而此時 FillGroupDataSet 程式中原先的『錯誤洞察』紅線也會消失，因為現在 PersonName 變數已經正確的宣告了。

程式碼流覽

當程式碼中的事件處理函式和程式愈來愈多時，您可能需要快速在不同的方法程式碼中流覽或是移動，您可以按兩下 IDE 左上方的項目樹狀架構中的方法節點，那麼編輯器會立刻移動到此方法，如下所示:



的快速鍵組合在不同的方法中快速移動:

版權所有 請勿翻印

鍵盤快速鍵	效果
按 CTRL+ALT+向上鍵	移至目前方法的頂端
按 CTRL+ALT+向下鍵	移至下一個方法
按 CTRL+ALT+HOME	移至檔案中的第一個方法
按 CTRL+ALT+END	移至檔案中的最後一個方法
按 CTRL+ALT	並捲動滑鼠滾輪，那麼編輯器就會以方法為單位移動

現在您就可以在開啟範例專案的程式碼編輯器中試著使用這些快速鍵。

4- 10 設定和使用書籤

當您在撰寫大量的程式碼時，您可能需要先暫停目前的程式碼而移動到另外的程式碼處撰寫其他的程式碼，之後再回到目前的程式碼繼續撰寫。在這種使用需求下您可以利用 IDE 的書籤功能。

IDE 提供了最多 10 個書籤可讓您在程式碼中標注，一旦您在程式碼中設定了書籤，就可以藉由快速鍵立刻在不同的書籤程式碼標注處移動。要在 IDE 中

設定書簽，您可以使用 Ctrl+Shift+0，Ctrl+Shift+1 一直到 Ctrl+Shift+9 最多設定 10 個書簽。

> 在繁體中文 OS 的環境下，您可能無法使用 Ctrl+Shift+0，Ctrl+Shift+1 來設定書簽。

> 您也可以使用 Ctrl+k+0 到 Ctrl+k+9 來設定書簽，使用 Ctrl+k+0 和 Ctrl+k+1 就可以避免無法使用 Ctrl+Shift+0，Ctrl+Shift+1 設定的問題。

現在請您移動游標到 FillGroupDataSet 程式的第 1 行程式碼處，接著同時按下 Ctrl+Shift+2，就可以如下圖看到 IDE 在編輯器最左邊出現了一個『2』的標誌，這就代表您在這行程式碼設定了一個書簽:

```
procedure TfmMainForm.FillGroupDataSet;
begin
  cdsGroups.Insert;
  cdsGroups.FieldByName('GroupName').AsString := '朋友';
  cdsGroups.FieldByName('Description').AsString := '一般朋友';
  cdsGroups.Post;

  cdsGroups.Insert;        { TODO -o老夫子 : 使用者點選此鍵後開啟瀏覽頁面 }
```

接著再請您移動游標到 InsertData 程式的第 1 行程式碼處，接著同時按下 Ctrl+Shift+3，就可以如下圖看到 IDE 在編輯器最左邊出現了一個『3』的標誌，這就代表您在這行程式碼設定了一個書簽:

```
end;

procedure TfmMainForm.ShowAddGroupPanel;
begin
  pnlAddGroup.Visible := True;
  pnlAddGroup.AnimateFloat('Height', 111, 1.5);
  edtAddGroup.SetFocus;
end;

procedure TfmMainForm.ShowWelcomeText;
```

現在您可以藉由同時按下 Ctrl+2 立刻把游標移動回 FillGroupDataSet 程式的第 1 行程式碼處，如果您同時按下 Ctrl+3，那麼又可以立刻把游標移動回 InsertData 程式的第 1 行程式碼處。

藉由使用 IDE 的書簽功能，可以方便的讓您在不同的工作程式碼處快速的移動和撰寫。

4-11 歷程管理員(History)

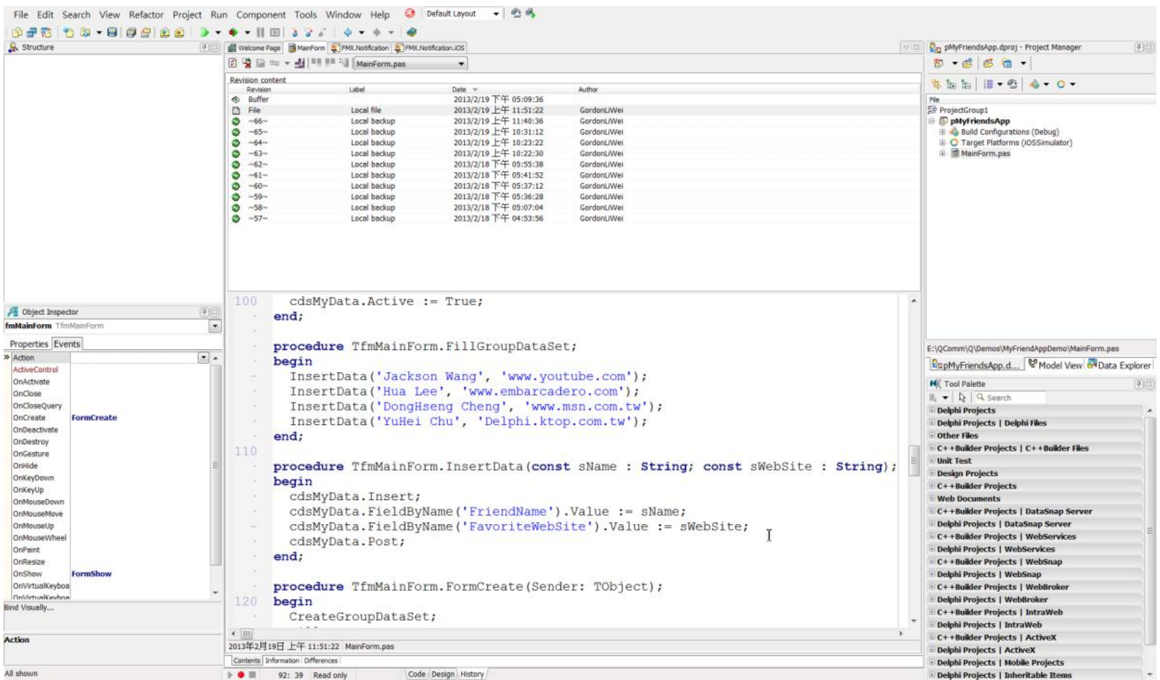
請您仔細看看上圖專案目錄中所有的檔案，您會看到其中有一個 History 子目錄，其中的檔案名比較特別，都是以『XXX.XXX.~數位~』樣例為檔案，如果您再仔細的觀察會發現其中的 XXX.XXX 都是您專案中的檔案。

事實上當您在 IDE 中對專案中的檔案進行異動時並且儲存之後，IDE 便會在這個 History 子目錄中儲存一份版本，因此這個 History 子目錄中所有的檔案便是您對專案修改的歷程，也就是說 IDE 會在這個 History 子目錄提供一些類似版本控制軟體提供的功能。

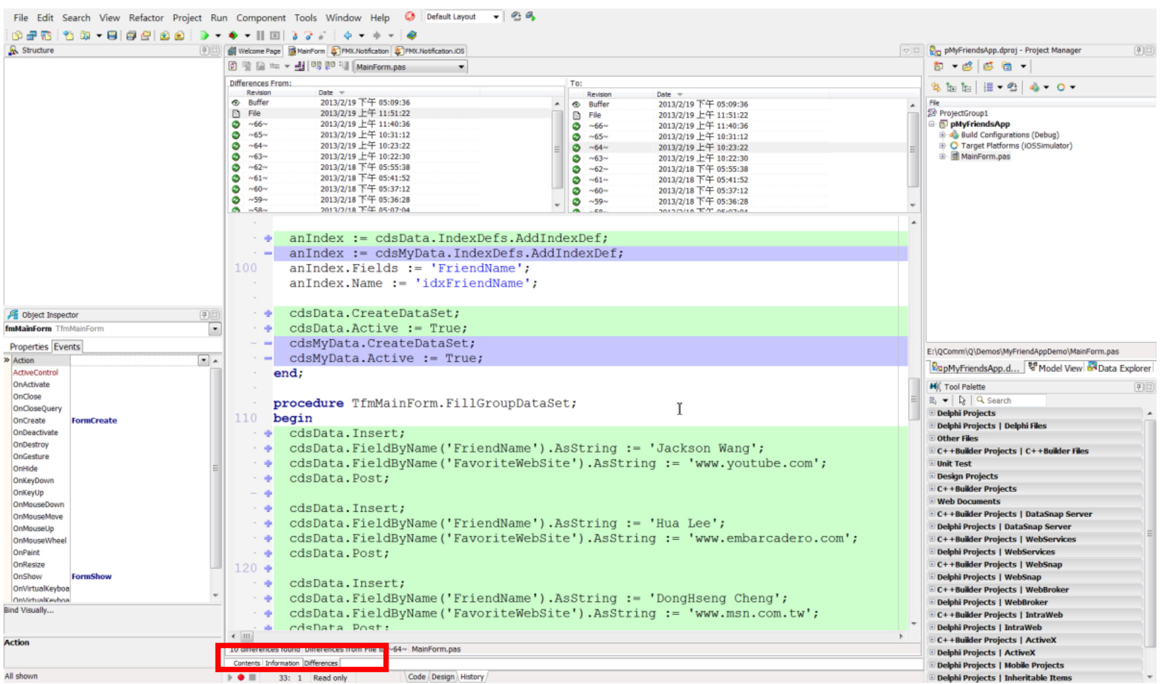
> 如果您使用 Delphi 開發真正的專案時，強烈的建議您應該使用真正的版本控制軟體，例如 git，來管理您的專案和原始程式檔案。

IDE 的這個功能稱為『歷程管理員』，您可以在 IDE 中啟動『歷程管理員』的管理和檢視介面，現在請您回到專案主表單的程式碼頁次，在編輯器下方您會看到一個『History』頁次，請按兩下這個頁次就會看到類似如下的畫面，每一個畫面上方的檔案就是您儲存(或是 IDE 自動儲存)的版本檔案，下方的視窗則是您點選上方版本的檔案內容:

版權所有 請勿翻印



您也可以比較不同版本之間的差異，請點選『History』頁次左下方的『Differences』子頁次，就可以看到類似如下的畫面:



在上圖中您可以先選擇左上方視窗中的版本檔案，再點選右上方中要比較差異的版本，例如上圖就是比較目前編輯器中的版本(Buffer)以及上一個已經儲存版本(~12~)的差異，那麼在下方的視窗中就可以看到 IDE 顯示兩個版本的差異，其中程式碼最左方如果出現『+』符號就代表是新增的程式碼，而『-』符號則代表是被刪除的程式碼。

版權所有 請勿翻印

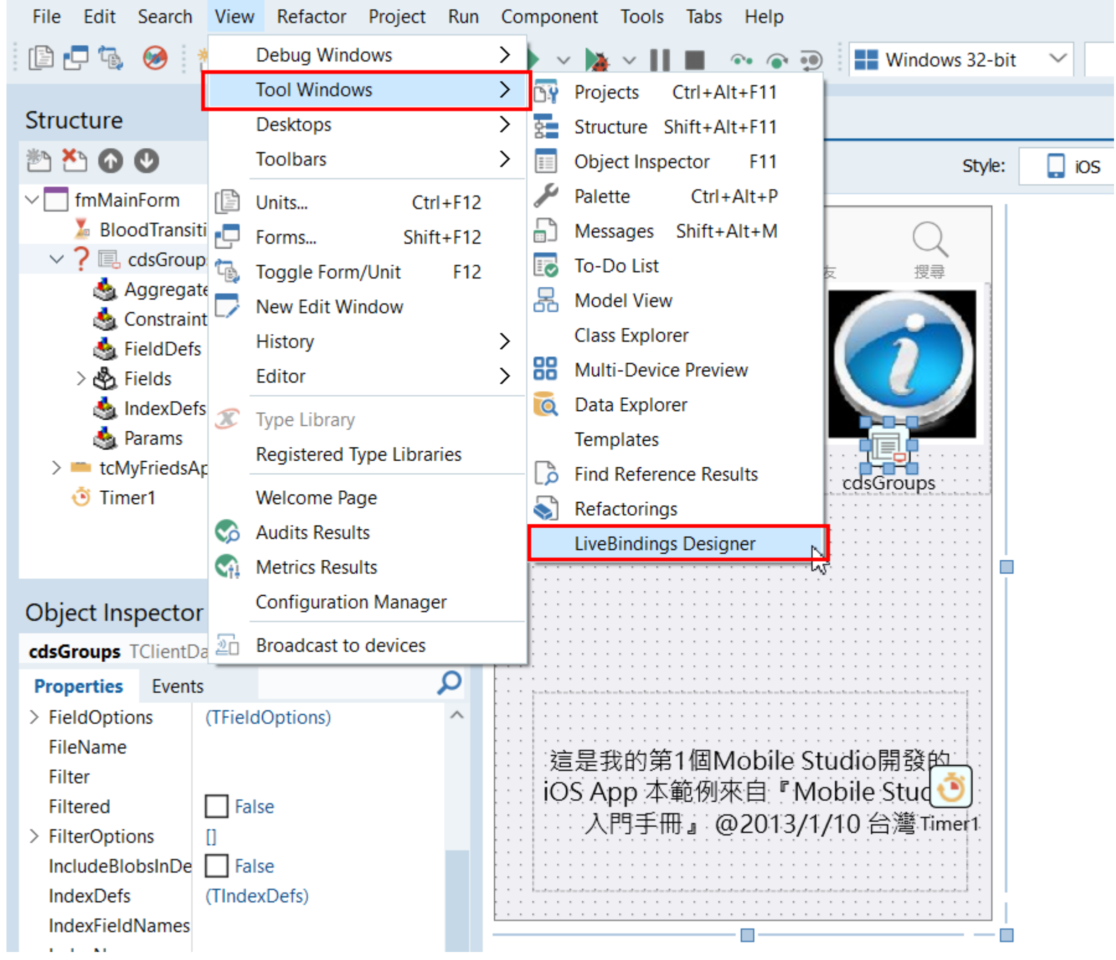
此外在上圖左上方，右上方視窗中的檔案前都有一個特定的圖像，不同的圖像代表不同的意義，下面的表格說明了不同圖像的意義:

圖像	說明
	最近一次儲存的版本
	備份的檔案版本
	目前存在於緩衝記憶體的版本，包含了尚未儲存的異動程式碼
	儲存在版本控制系統中的版本
	從控制系統中簽出(Check Out)的版本

現在我們就可以繼續開發這個範例 App 了，首先讓我們使用 LiveBinding 功能系結資料，最後再實作 SearchEditButton1Click 事件處理函式。

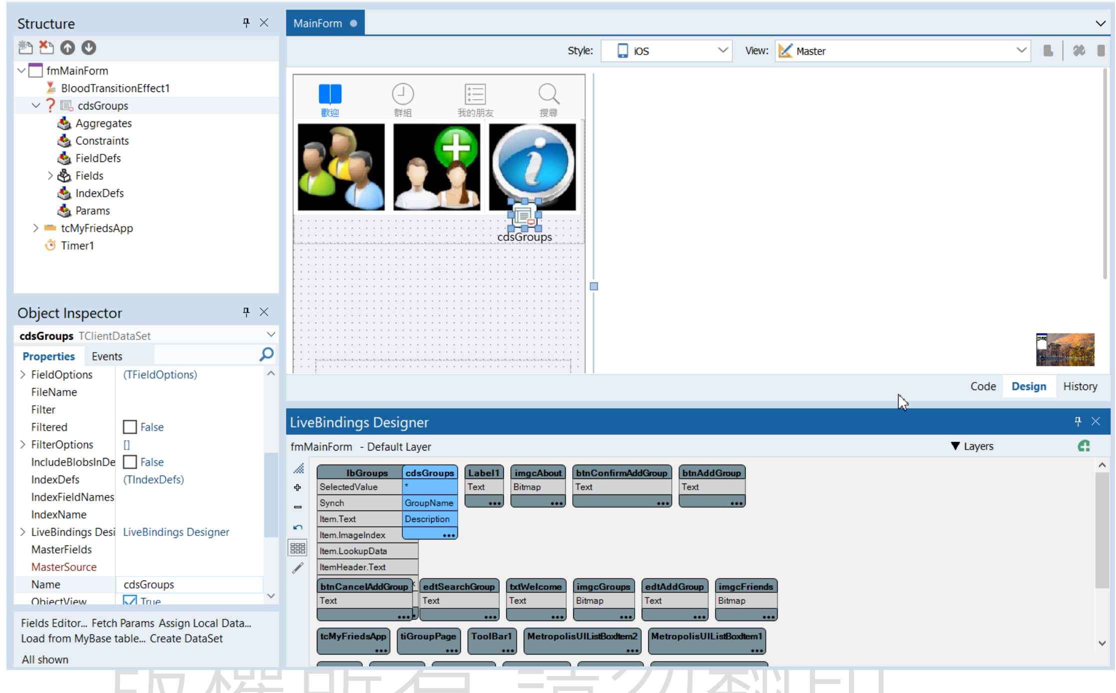
Delphi for iOS 提供了非常視覺化，簡單又強大的 LiveBinding 功能讓 iOS 開發人員能夠輕易的存取資料並且把資料顯示(系結)在視覺化組件中。

在前面的 FillGroupDataSet 程式中我們已經建立了一個 TClientDataSet 並且在其中新增了幾筆資料，現在讓我們在主表單的 TTabControl 的第 3 個頁次顯示這些資料。點選 TTabControl 的第 3 個頁次，再點選整合發展環境主功能表的 View | Tool Windows | LiveBindings Designer 功能表以開啟視覺化即時資料系結設計家，如下所示:



視覺化即時資料系結設計家中您可以使用拖曳拉線的方式系結資料和元件。例如現在我們想把 TClientDataSet 中的 FriendName 欄位顯示在 TTabControl 的第 3 個頁次中的第一個 TEdit 元件，把 FavoriteWebSite 欄位顯示在第二個 TEdit 元件中。因此請在視覺化即時資料系結設計家中先點選 BindSourceDB1 中的 FriendName 欄位再持續按著滑鼠左鍵拖曳到 Edit1 的 Text 特性上再放開滑鼠左鍵，此時在這 2 者之間就會出現一個雙向箭頭的線條，這就代表現在我們已經系結了 FriendName 欄位的數值和 Edit1.Text，也就是

說 FriendName 欄位的數值會自動顯示在 Edit1.Text 中。同樣的，先點選 BindSourceDB1 中的 FavoriteWebSite 欄位再持續按著滑鼠左鍵拖曳到 Edit2 的 Text 特性上再放開滑鼠左鍵，此時 2 者之間也會出現一個雙向箭頭的線條，如下所示:



版權所有 請勿翻印

現在如果我們編譯並且分發此時的範例 iOS App 到 iOS 的模擬器中的話，點選第 3 個頁次就可以看到類似如下的執行結果，資料果然自動顯示在 2 個 TEdit 元件中，如果點選上方的 Navigator 元件就可以在不同的資料中流覽和移動了，開發能夠存取資料的 iOS App 就是這麼簡單，太酷了。

注意，由於現在的範例 iOS App 使用了 Delphi for iOS 的 DataSnap 功能，因此您無法只編譯它就執行，您需要分發 DataSnap 相關的檔案和分享函式庫。在稍後的章節中會說明如何使用整合發展環境分發需要額外功能和檔案的 iOS App。

4-13 實作 SearchEditButton1Click 事件處理函式

現在我們可以完成這個範例 iOS App 的最後一個功能了，那就是當使用者點選了第 3 個頁次中位於 FavoriteWebSite 旁的搜尋按鈕時就開啟第 4 個頁次並且使用流覽器帶領使用者到 FavoriteWebSite 欄位數值指定的網站。

實作這個功能非常的簡單，Delphi for iOS 提供了 TWebBrowser 元件，只要我們使用 TWebBrowser 元件並且設定它的 URL 特性值，再呼叫它的 Navigate 方法即可。因此請在 SearchEditButton1 的事件處理函式中撰寫如下的程式碼:

```
procedure TfmMainForm.SearchEditButton1Click(Sender: TObject);
begin
  WebBrowser1.URL := Edit2.Text;
  tcMyFriedsApp.ActiveTab := tiSearch;
```

```
  WebBrowser1.Navigate;
end;
```

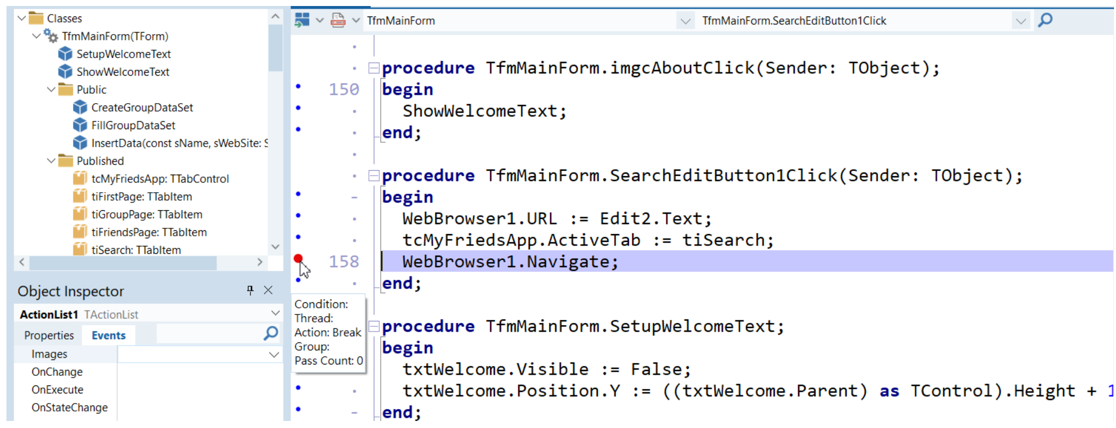
再次編譯並且執行範例 iOS App 就可以在 iOS 模擬器中點選第 3 個頁次中位於 FavoriteWebSite 旁的搜尋按鈕時看到類似如下的結果，範例 iOS App 果然可以流覽到 FavoriteWebSite 欄位指定的網站，Cool。

5 除錯您的 iOS App

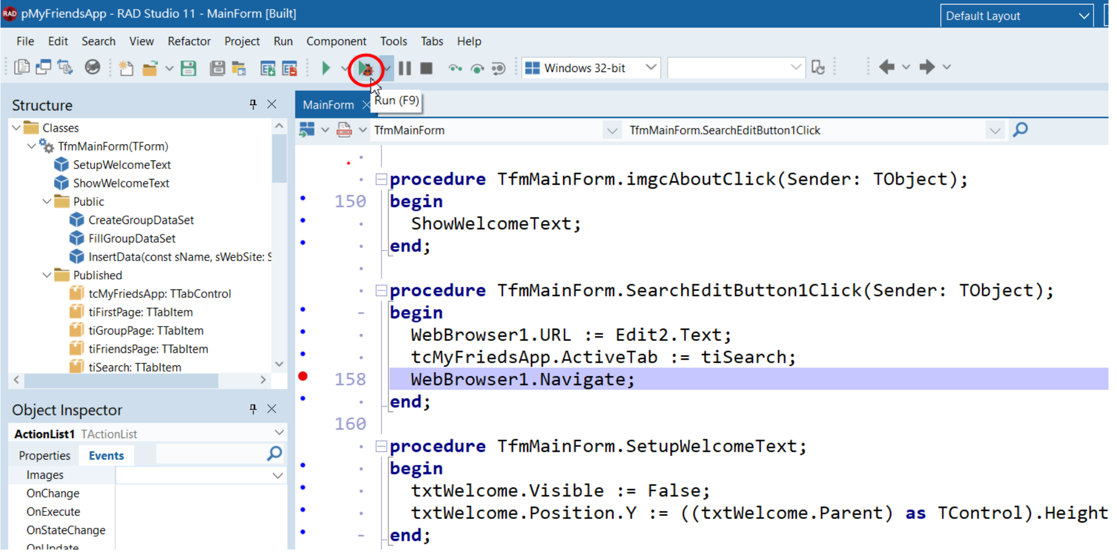
當您使用 Delphi for iOS 開發 iOS App 時一定會需要除錯您的應用程式，Delphi for iOS 提供了非常方便又強大的除錯功能能幫助您，例如本書的範例應用程式在實作了上一小節的程式碼後可能發生了一些錯誤，因此讓我們學習如何除錯應用程式以便修正範例應用程式中可能的錯誤。

Delphi for iOS 能夠讓您在 iOS 模擬器中除錯或是在 iOS 設備中除錯，甚至提供了一個 Windows 模擬介面提供您除錯。在本書中讓我們展示如何在 iOS 模擬器中除錯。首先讓我們在範例應用程式中設定『中斷點』，『中斷點』是指當 iOS App 在 iOS 模擬器中或是在 iOS 設備中執行到此地時便會中斷，以便讓開發人員可以檢查相關的變數，物件或是記憶體，CPU 等重要的資訊，來決定程式碼是否發生了錯誤。

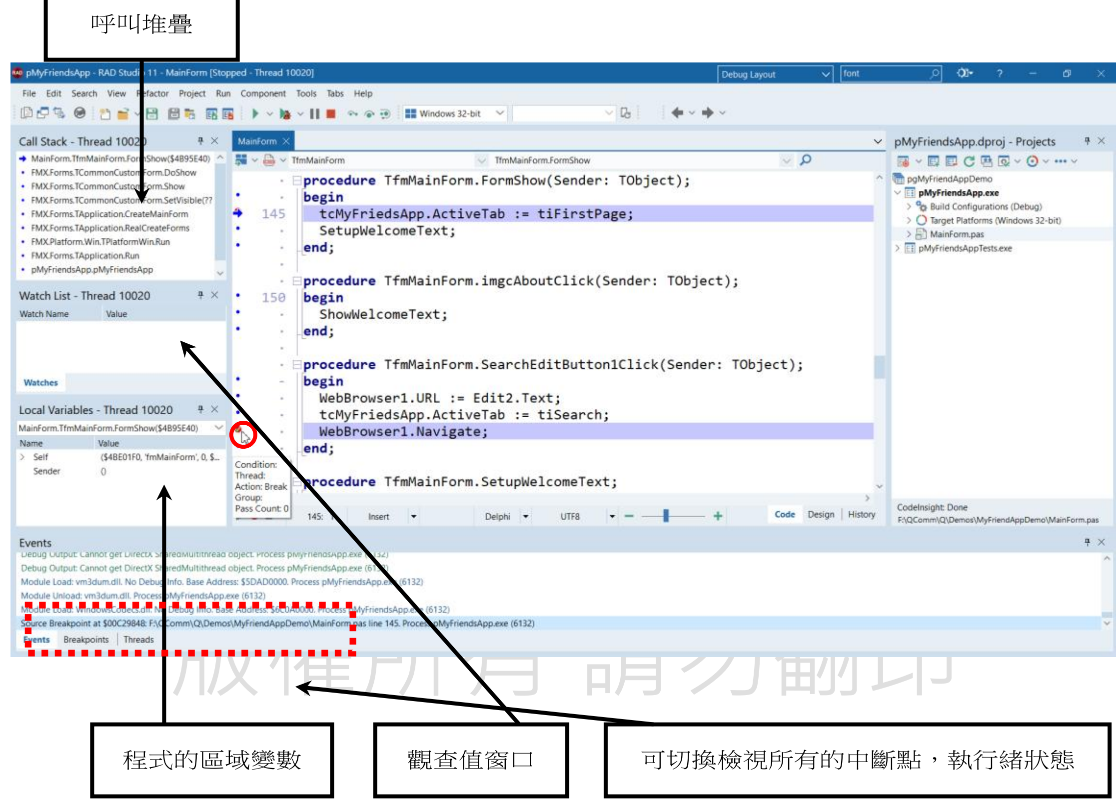
現在讓我們在前面剛實作的 SearchEditButton1Click 程式中設定一個中斷點，請切換到程式碼頁次，使用滑鼠在 SearchEditButton1Click 程式的最後一行程式碼處的最左邊按一下滑鼠左鍵，此時在 WebBrowser1.Navigate 最左邊就會出現一個紅色的圓點『🔴』，這就代表在此設定了一個『中斷點』，稍後當範例 iOS App 在 iOS 模擬器中執行到此地時就會中斷執行並且把執行權從應用程式切換回 Delphi for iOS 的 IDE，如下所示:



設定好此『中斷點』之後，您就可以點選 IDE 上方的『Run』按鈕，或是按下『F9』，IDE 就會啟動除錯器開始執行您的應用程式，如下所示:



當範例 iOS App 執行後，請先點選主表單 TTabControl 第 3 個頁次，再點選 3 個頁次中位於 FavoriteWebSite 旁的搜尋按鈕，就可以看到範例 iOS App 被暫停執行，並且切換回 Delphi for iOS 的 IDE，此時您會看到 IDE 暫停在剛才設定的『中斷點』上，而且原先『中斷點』的符號現在變成『』符號，如下所示。



而且請您注意 IDE 右上方，此時 IDE 也被設定成在除錯的桌面組態設定:

除錯的組態設定會自動顯示呼叫堆疊視窗，程式的區域變數視窗和觀查值視窗。下面的表格說明了這些視窗的意義:

窗口	說明
堆疊視窗	應用程式的呼叫堆疊次序，您可以在這個視窗中看到中斷點被呼叫的執行次序
程式的區域變數視窗	此視窗自動顯示目前程式中所有的區域變數的數值
觀查值窗口	您可以在此視窗中加入檢視任何的全域變數，資料結構或是物件的數值

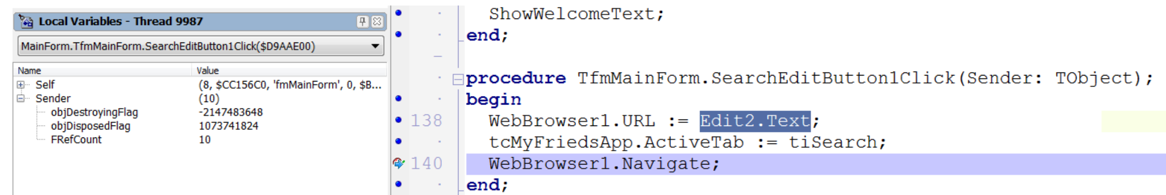
現在請您先觀察 IDE 左上方的『堆疊視窗』，您可以看到類似如下的內容:

```
Call Stack - Thread 9987
MainForm.TfmMainForm.SearchEditButton1Click($D9AAE00)
FMX.Controls.TControl.Click
FMX.StdCtrls.TCustomButton.Click
FMX.Controls.TControl.MouseUp(mbLeft,[ssLeft,ssTouch],10,11)
FMX.StdCtrls.TCustomButton.MouseUp(mbLeft,[ssLeft,ssTouch],10,11)
FMX.Forms.TCommonCustomForm.MouseUp(mbLeft,[ssLeft,ssTouch],280,149)
FMX.Platform.iOS.TFMXViewBase.touchesEnded(Pointer($1211DB90) as NSSet,Pointer($1211DBC0) as UIEvent)
Macapi.ObjectiveC.DispatchToDelphi
:0016518e DispatchToDelphi + $26
:05941cef ; UIKit
:05941f02 ; UIKit
:0591fd4a ; UIKit
:05911698 ; UIKit
:07002df9 ; GraphicsServices
:07002ad0 ; GraphicsServices
:04dd6bf5 ; CoreFoundation
:04dd6962 ; CoreFoundation
:04e07bb6 ; CoreFoundation
:04e06f44 ; CoreFoundation
:04e06e1b ; CoreFoundation
:070017e3 ; GraphicsServices
:07001668 ; GraphicsServices
:0590effc ; UIKit
FMX.Platform.iOS.TPlatformCocoaTouch.Run
FMX.Forms.TApplication.Run
pMyFriendsApp.pMyFriendsApp
```

『堆疊視窗』顯示了您的 iOS App 的執行路徑，例如在上圖中可以看到這個範例 iOS App 的進入點是 FMX.Platform.iOS.TPlatformCocoaTouch.Run 程式，接著範例 iOS App 回應剛才在主表單中滑鼠的點選事件，從 TControl.Click 方法呼叫主表單中的 TfmMainForm.SearchEditButton1Click 程式。

版權所有 請勿翻印

接著再觀察『區域變數視窗』，如果您仔細比較『區域變數視窗』的內容和 SearchEditButton1Click 程式，如下所示:



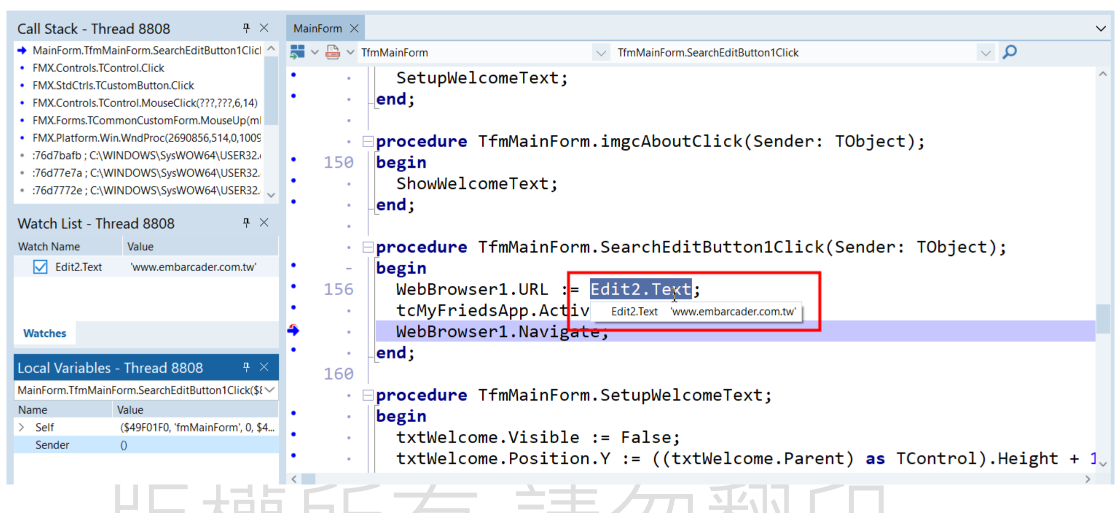
您可以發現『區域變數視窗』中顯示的內容正是 SearchEditButton1Click 程式中所有的區域變數和參數，如此一來當您除錯 SearchEditButton1Click 程式時就可以對所有的區域變數值以及參數值一目了然。

當應用程式執行權暫停在中斷點時，您也可以使用滑鼠來檢視程式碼中的變數，資料結構或是物件的數值。例如現在 SearchEditButton1Click 程式中使用了:

```
WebBrowser1.URL := Edit2.Text;
```

上面的 Edit2.Text 是 3 個頁次中位於 FavoriteWebSite 欄位的數值，此時您可能想知道它的數值是什麼。

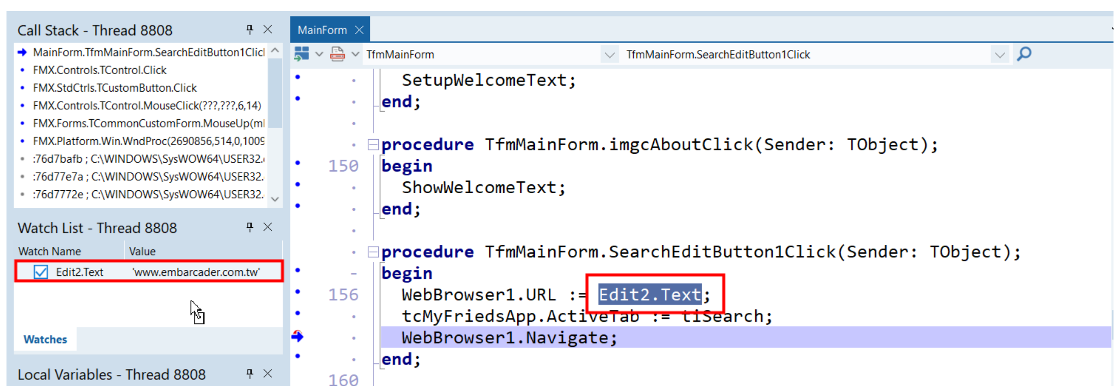
有數種方法可以讓您觀察 iOS App 程式碼中資料結構中的資料，第 1 種方法是使用滑鼠選擇您想觀察的資料結構數值，然後暫停滑鼠在此程式碼之上一下子，IDE 就會直接顯示這個資料結構中包含的數值。例如下圖就是使用滑鼠選擇了程式碼中的 Edit2.Text 之後，您就可以看到 IDE 在游標下方顯示了目前 Edit2.Text 中的數值:



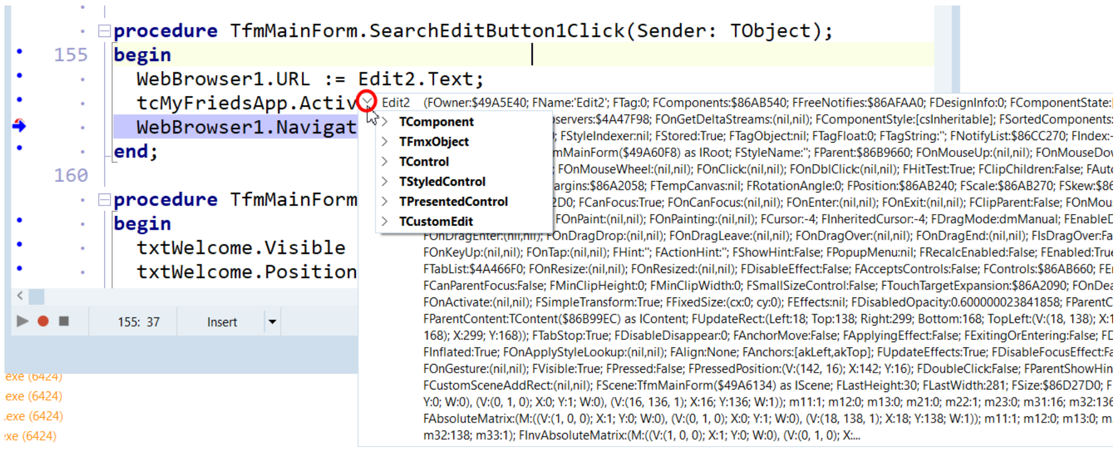
版權所有 請勿翻印

第 2 種方法是把這個程式碼拖曳到『觀查值視窗』中，例如下圖就是使用滑鼠選擇了 Edit2.Text 之後，把它拖曳到『觀查值視窗』中。

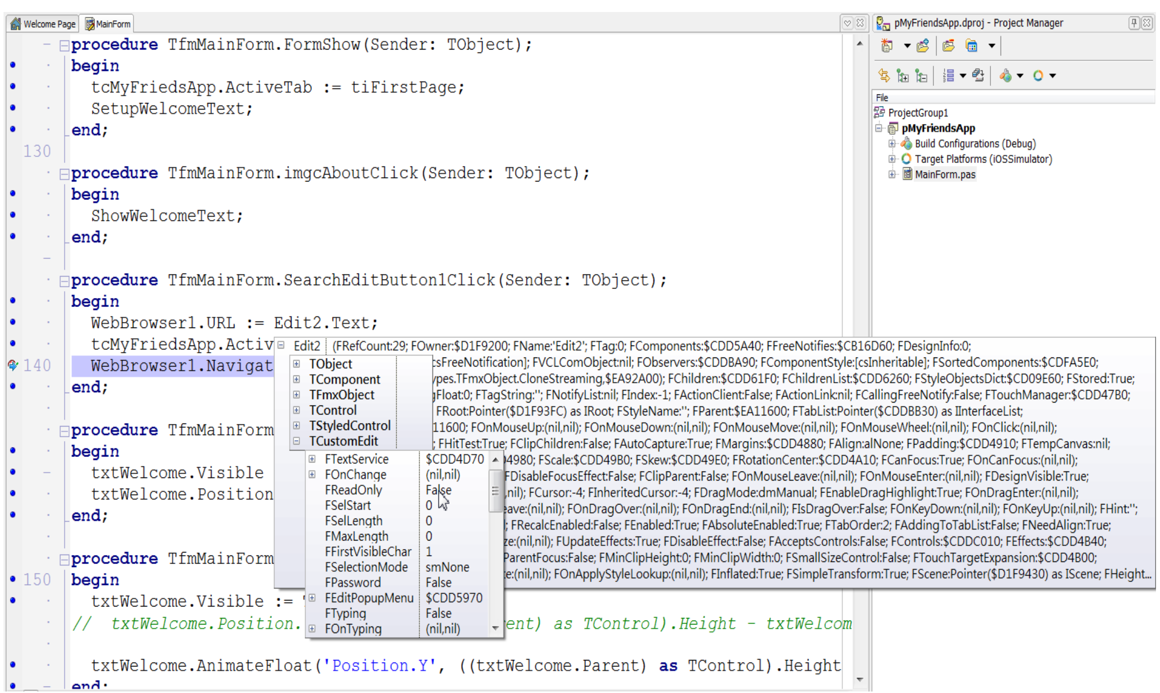
一旦資料結構被拖曳到『觀查值視窗』之後，它就會停駐在『觀查值視窗』中，而且當資料結構中的數值改變時，『觀查值視窗』也會立刻顯示最新的數值。



如果您想觀察整個資料結構或是物件中所有的資料，那麼您可以使用滑鼠把游標放在此資料結構或是物件之上，那麼除錯器就會顯示其中所有的數值。例如下圖就是把滑鼠游標放在程式碼中的 Edit2 之上，除錯器就立刻顯示 Edit2 之中所有的資料:



請注意上圖中 Edit2 左方有一個『>』符號，這代表您可以使用滑鼠展開其中的內容。例如在下圖移動滑鼠到 Edit2 左方的『>』符號，就可以看到除錯器展開 Edit2 中的內容，除錯器詳細的列出了 Edit2 中每一個元素的數值，這個功能對於觀察複雜的資料結構或是物件的內容是非常有用的。

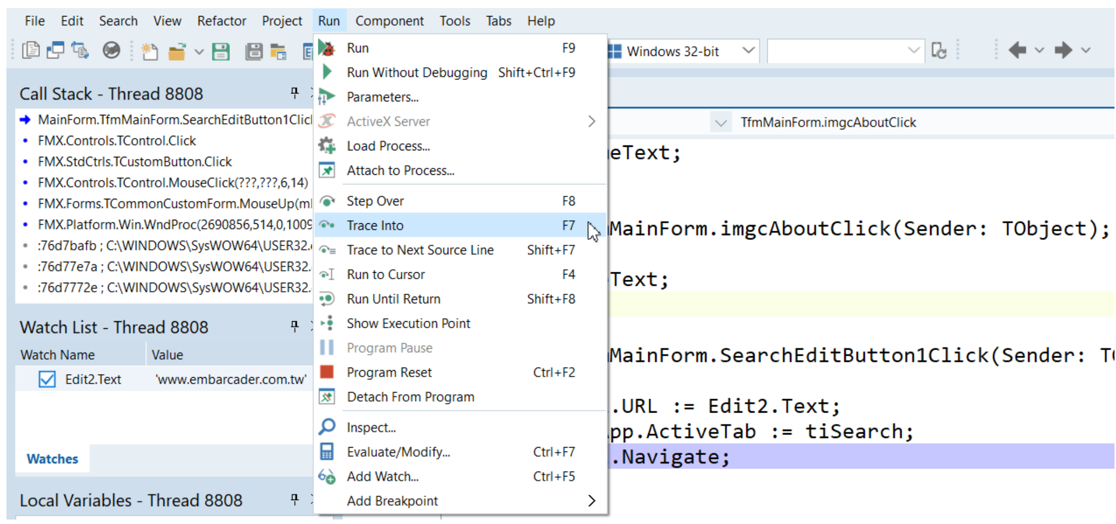


現在請您再次按一下程式碼中的中斷點以取消原本的中斷點，然後按下 F9 執行範例應用程式，您就可以看到範例 iOS App 會繼續執行下去了：

版權所有 請勿翻印

現在範例 iOS App 果然根據不同的資料中 FavoriteWebSite 欄位的數值帶領使用者到不同的網站了。

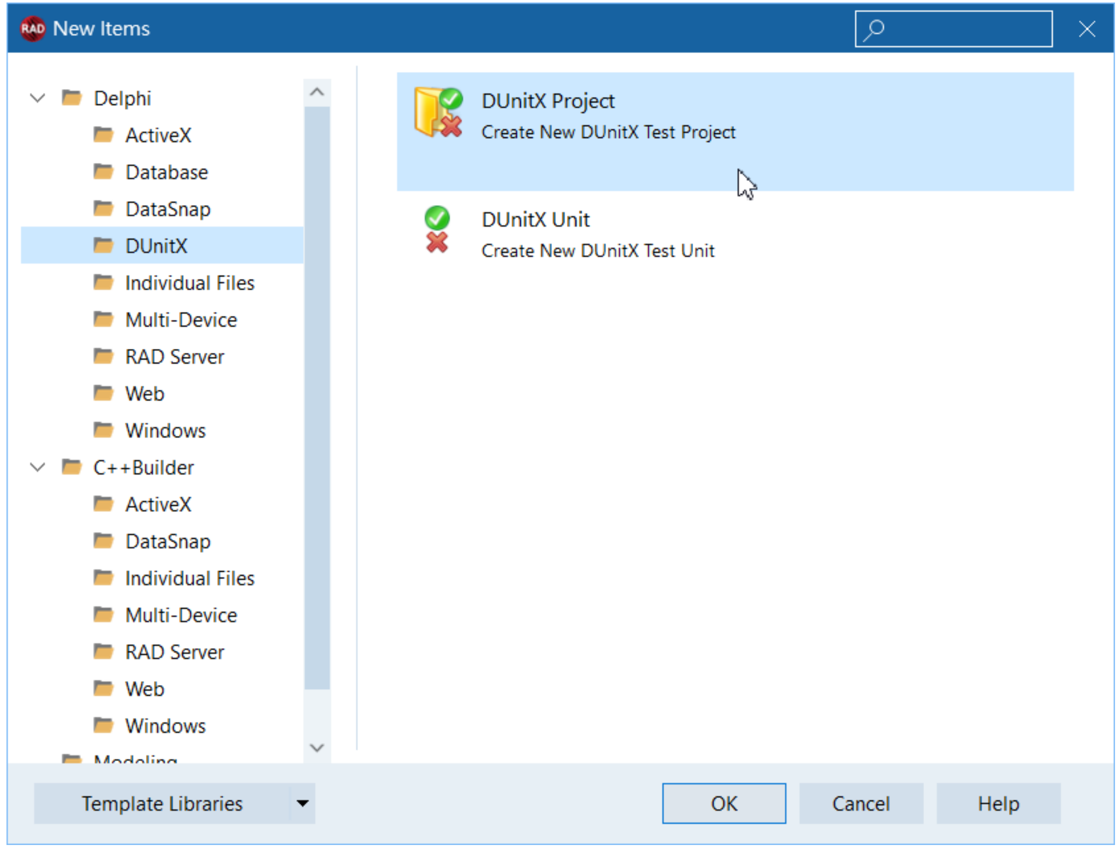
當您在除錯應用程式時，您也可以按下 F7 鍵一行一行的除錯程式碼，或是按下 F8 鍵一次執行一個方法。如果您的程式碼中擁有迴圈而您不想一直在迴圈中除錯，您可以在迴圈之後的程式碼處設定中斷點，再按下 F4 鍵一次執行到迴圈之後的中斷點，下圖就是您在 IDE 中除錯時可以使用的功能鍵:

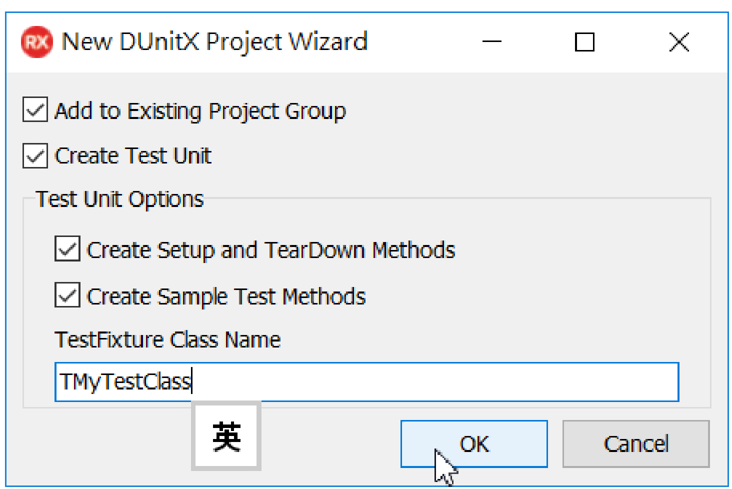


6. 為您的 iOS App 進行單元測試

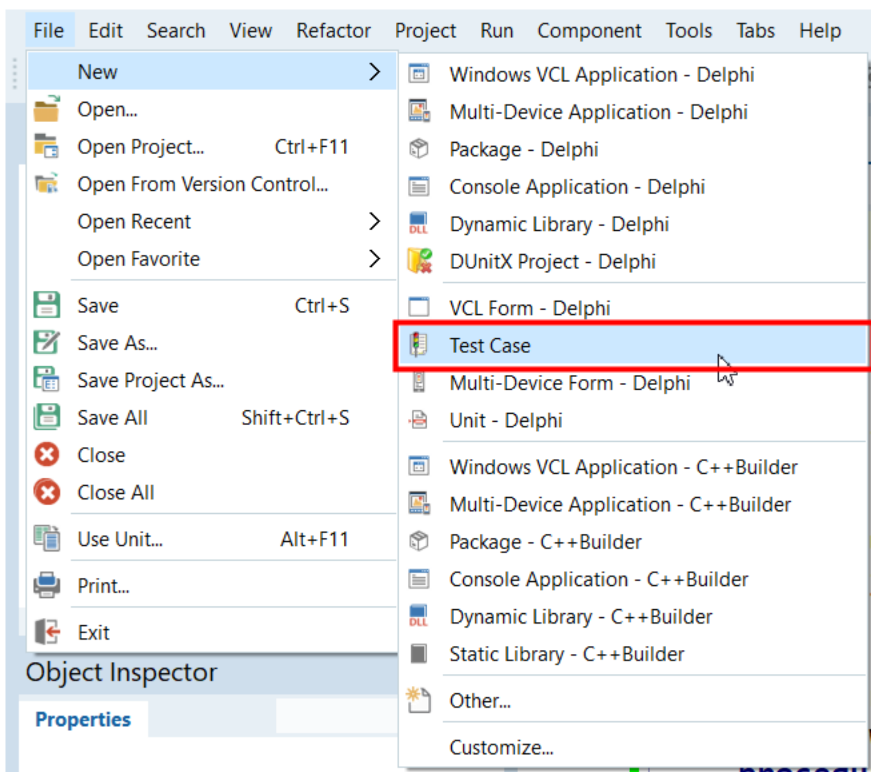
在使用 Delphi for iOS 開發 App 時您需要對於 App 的品質進行嚴格的把關，因為相對於 PC 的軟體來說 iOS App 執行的硬體環境更為嚴格。Delphi for iOS 雖然已經在程式語言，執行時期函式館和框架中盡可能的您，讓您開發的 iOS App 儘量的安全，但您自己撰寫的程式碼也需要盡可能的安全，以避免您的 App 在執行時發生異常的狀況。因此 Delphi for iOS 提供了單元測試的功能來測試您撰寫的程式碼。本小節就簡單的說明如何使用單元測試功能來測試您的程式碼。

請在整合發展環境中開啟 pMyFriendsApp 範例 App 專案，點選專案經理中的 ProjectGroup1 節點，點選 New items 快捷鈕，再點選 Unit Test 群組中的 Test Project 圖像以便在專案群組中建立一個測試案例項目，如下所示:

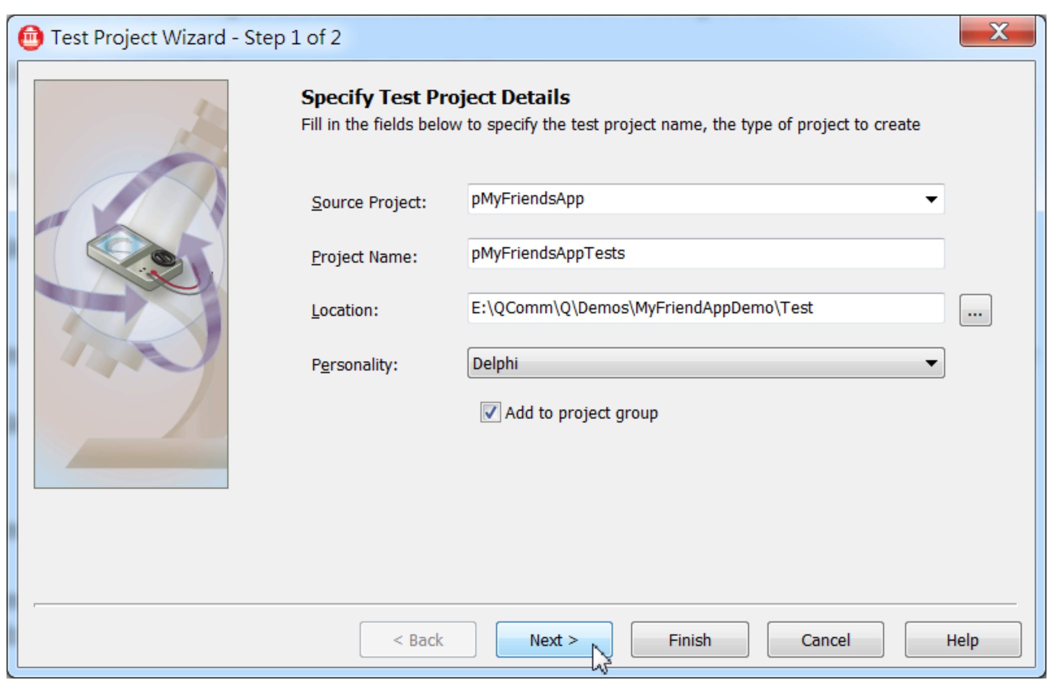




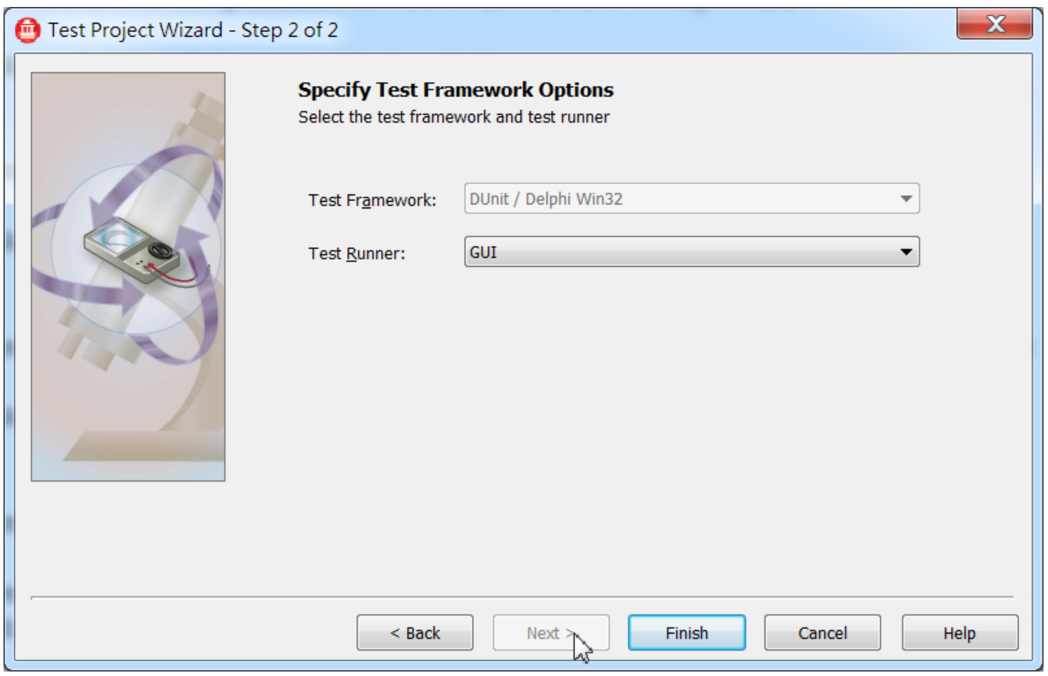
點選上圖中的 OK 之後，再建立一個測試案例(如果讀者沒看到此選項，可先點選下圖中的 Customize 選項,再選擇把 Test Case 加入選單中):



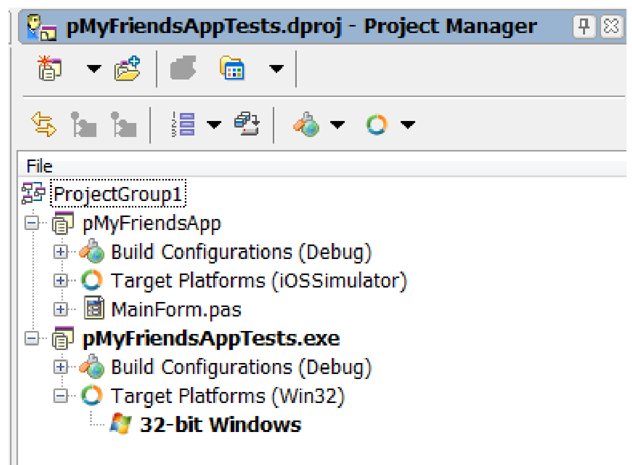
在隨後的對話盒中輸入如下的資訊:



點選 Next，再如下圖的對話盒選擇 GUI 測試介面，最後點選 Finish 以完成建立單元測試專案。



整合發展環境就會以 pMyFriendsAppTests.dproj 名稱儲存此測試單元專案，如下所示:

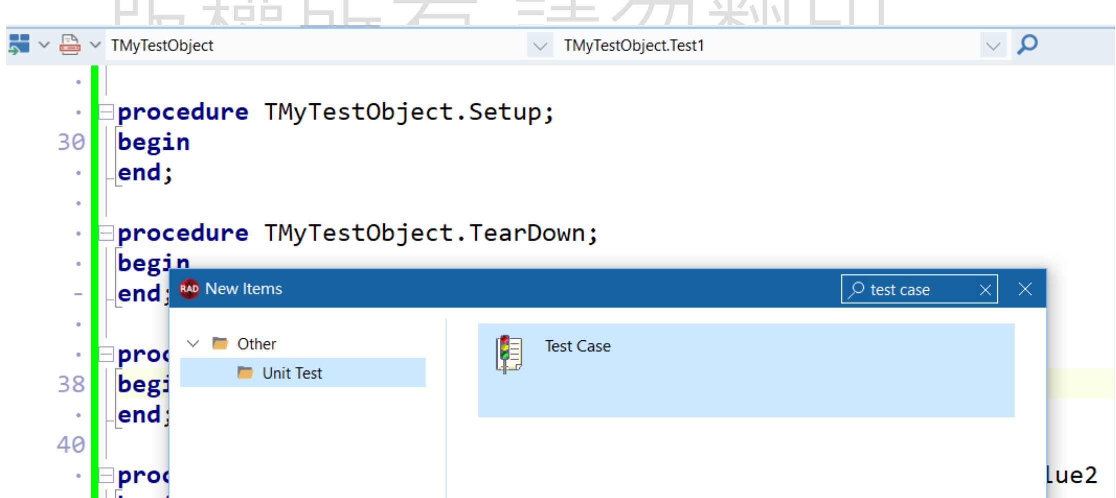


進行單元測試 - 建立測試案例

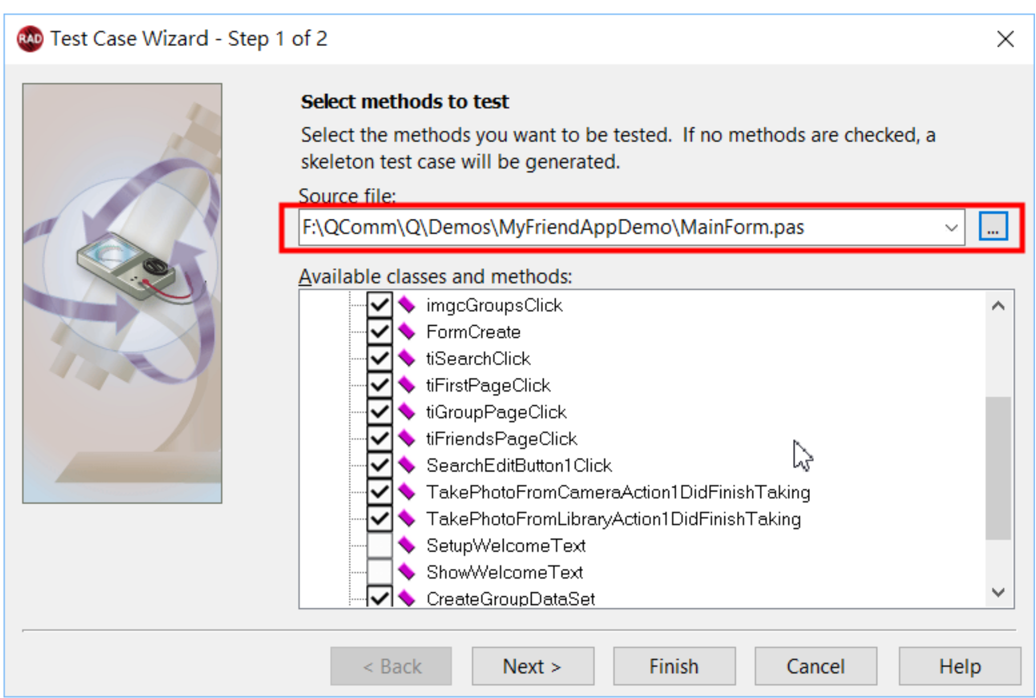
建立了單元測試專案之後請在請在專案經理中按兩下『pMyFriendsAppTests』專案讓它成為目前的專案，然後點選工具列中的 New items 快速鍵在專案中建立一個測試案例以測試前面開發的 pMyFriendsApp 專案中的類別程式碼。

請在 New Items 對話盒中選擇 Unit Test 中的 Test Case 圖像，如下所示:

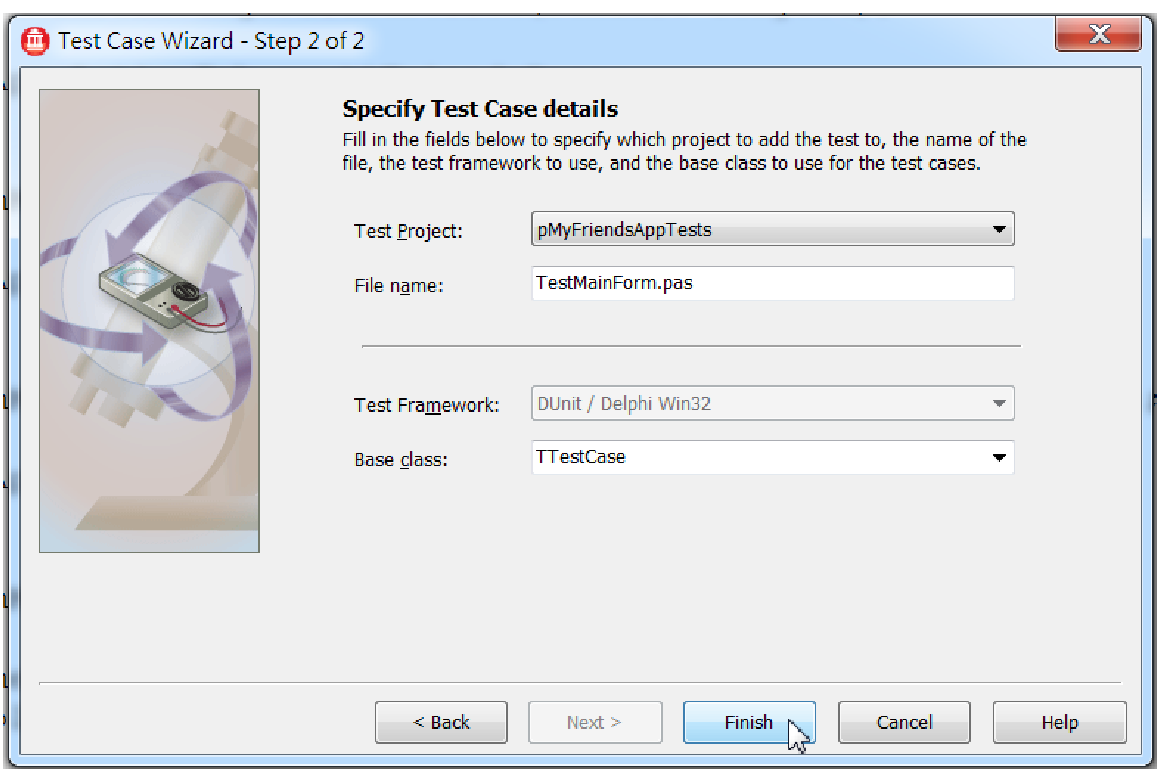
版權所有 請勿翻印



接著在 Test Case Wizard 對話盒中的 Source file:處選擇要測試『pMyFriendsApp』項目中的 MainForm.pas，此時測試案例精靈就會顯示 MainForm.pas 中所有的方法，您可以勾選您想測試的方法，在這裡讓我們勾選 CreateGroupDataSet，FillGroupDataSet 和 InsertData 這 3 個方法，如下所示:



最後點選 Finish 按鈕以完成在測試專案中建立測試案例的工作:



點選 Finish 按鈕之後測試案例精靈會產生如下的程式碼，其中 051 行的 TestCreateGroupDataSet 是使用來測試 CreateGroupDataSet 方法，057 行的 TestFillGroupDataSet 是測試 FillGroupDataSet 方法，而 063 行的 TestInsertData 是測試 InsertData 方法，您可以看到

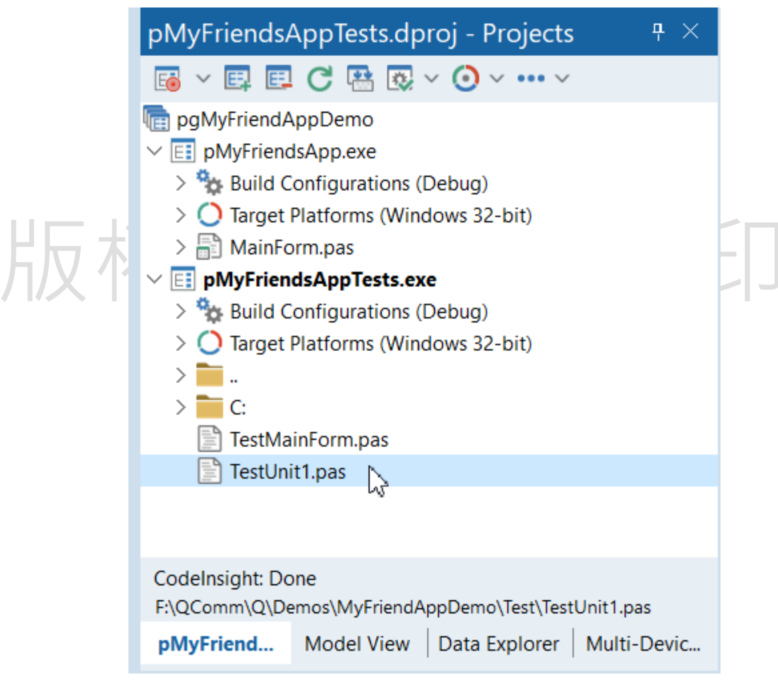
TestCreateGroupDataSet 和 TestFillGroupDataSet 中都有 ToDo 標籤，這代表開發人員需要在其中撰寫測試程式碼，我們很快就會進行這個工作。

```
001   unit TestMainForm;
002   {
003
004     DelphiDUnit Test Case
005     ----------------------
006     This unit contains a skeleton test case class generated by the
Test Case Wizard.
007     Modify the generated code to correctly setup and call the methods
from the unit
008     being tested.
009
010   }
011
012   interface
013
014   uses
015     TestFramework, Datasnap.Provider, FMX.Layouts, System.Types,
Fmx.Bind.Editors,
016     System.SysUtils, FMX.Filter.Effects, FMX.Forms, System.Classes,
Fmx.Bind.DBEngExt,
017     MainForm, Fmx.Bind.Navigator, Datasnap.DBClient, FMX.Edit,
FMX.TabControl,
018     System.UITypes, FMX.WebBrowser, Data.Bind.DBScope, FMX.Types,
Data.DB,
019     Data.Bind.Components, FMX.ListBox, System.Bindings.Outputs,
System.Rtti,
020     FMX.Dialogs, Data.FMTBcd, FMX.Effects, System.Variants,
FMX.Objects, FMX.Controls,
021     Data.SqlExpr, FMX.StdCtrls, Data.Bind.EngExt;
022
023   type
024     // Test methods for class TfmMainForm
025
026     TestTfmMainForm = class(TTestCase)
027     strict private
```

```
028        FfmMainForm: TfmMainForm;
029      public
030        procedure SetUp; override;
031        procedure TearDown; override;
032      published
033        procedure TestCreateGroupDataSet;
034        procedure TestFillGroupDataSet;
035        procedure TestInsertData;
036      end;
037
038    implementation
039
040    procedure TestTfmMainForm.SetUp;
041    begin
042      FfmMainForm := TfmMainForm.Create;
043    end;
044
045    procedure TestTfmMainForm.TearDown;
046    begin
047      FfmMainForm.Free;
048      FfmMainForm := nil;
049    end;
050
051    procedure TestTfmMainForm.TestCreateGroupDataSet;
052    begin
053      FfmMainForm.CreateGroupDataSet;
054      // TODO: Validate method results
055    end;
056
057    procedure TestTfmMainForm.TestFillGroupDataSet;
058    begin
059      FfmMainForm.FillGroupDataSet;
060      // TODO: Validate method results
061    end;
062
063    procedure TestTfmMainForm.TestInsertData;
064    var
065      sWebSite: string;
```

```
066       sName: string;
067     begin
068       // TODO: Setup method call parameters
069       FfmMainForm.InsertData(sName, sWebSite);
070       // TODO: Validate method results
071     end;
072
073     initialization
074       // Register any test cases with the test runner
075       RegisterTest(TestTfmMainForm.Suite);
076     end.
```

現在如果回到專案經理中便可以看到如下的專案群組，以及其中的 2 個專案:



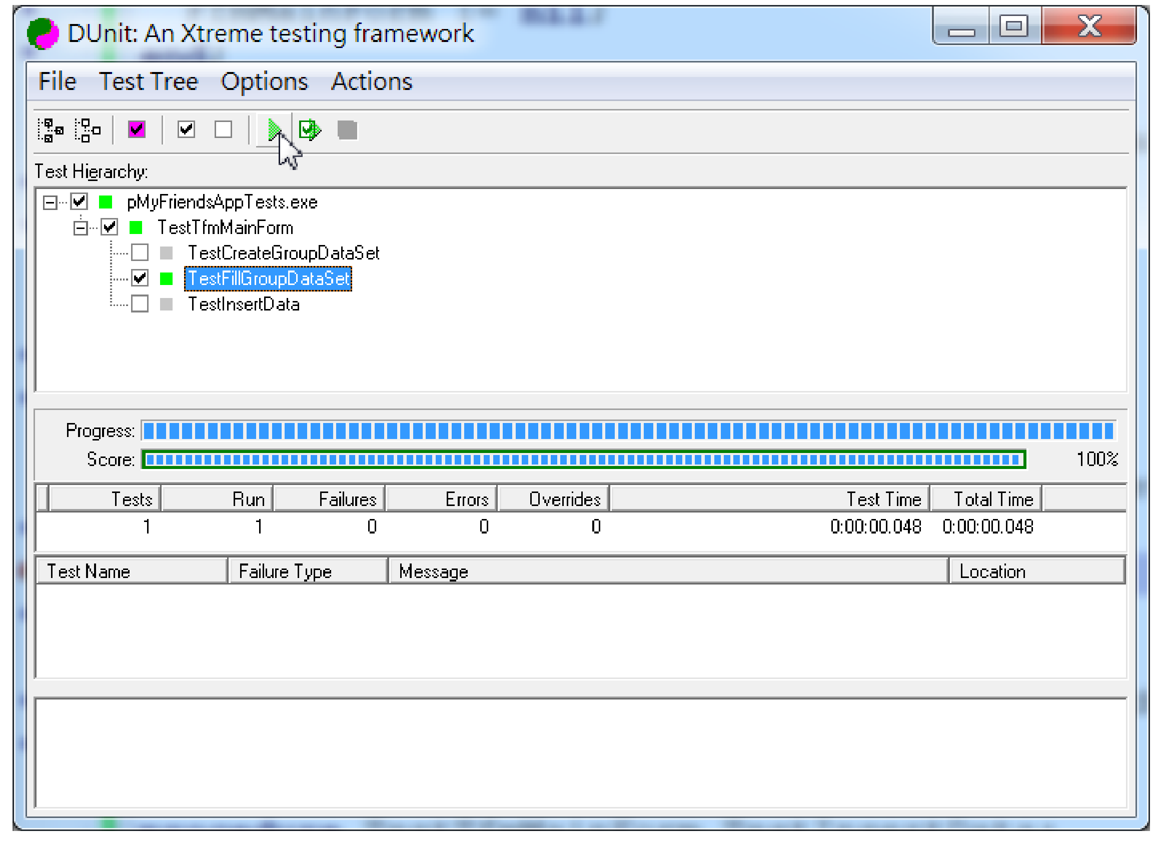
現在讓我們在 TestCreateGroupDataSet 中撰寫一些測試程式碼看看 CreateGroupDataSet 方法是否能正確的工作，在 003 行呼叫 TfmMainForm 物件的 CreateGroupDataSet 方法以建立 TClientDataSet 物件，接著 006 行再測試 TClientDataSet 物件是否成功建立並且是開啟的狀態，如果是的話就代表 TestCreateGroupDataSet 方法測試成功。

TestFillGroupDataSet 方法先在 011 行建立 TClientDataSet 物件，再於 012 行呼叫 FillGroupDataSet 方法，最後 014 行呼叫 CheckEquals 方法測試 FillGroupDataSet 方法是否成功的在 TClientDataSet 物件中建立了 4 筆資料，是的話 TestFillGroupDataSet 方法就代表測試成功。

最後的 TestInsertData 方法呼叫 CreateGroupDataSet 方法建立 TClientDataSet 物件，再呼叫 InsertData 方法新增一筆資料到 TClientDataSet 中，最後再呼叫 CheckEquals 來檢查剛才新增的資料是否真正的加入到 TClientDataSet 物件中。

```
001   procedure TestTfmMainForm.TestCreateGroupDataSet;
002   begin
003     FfmMainForm.CreateGroupDataSet;
004     // TODO: Validate method results
005
006     CheckTrue(FfmMainForm.cdsMyData.Active);
007   end;
008
009   procedure TestTfmMainForm.TestFillGroupDataSet;
010   begin
011     FfmMainForm.CreateGroupDataSet;
012     FfmMainForm.FillGroupDataSet;
013     // TODO: Validate method results
014     CheckEquals(4, FfmMainForm.cdsMyData.RecordCount);
015   end;
016
017   procedure TestTfmMainForm.TestInsertData;
018   var
019     sWebSite: string;
020     sName: string;
021   begin
022     FfmMainForm.CreateGroupDataSet;
023     // TODO: Setup method call parameters
024     FfmMainForm.InsertData('test', 'testsite');
025     // TODO: Validate method results
026     CheckEquals('test',
FfmMainForm.cdsMyData.FieldByName('FriendName').Value);
027   end;
```

現在請執行此測試項目，點選工具列中的綠色執行測試按鈕您就可以看到測試專案執行了 TestFillGroupDataSet 並且以綠色顯示 TestFillGroupDataSet 成功的的測試了 FillGroupDataSet 方法而且 FillGroupDataSet 方法是正確的執行。



版權所有 請勿翻印

使用單元測試功能非常的實用，每當您撰寫一個新的方法時就應該去測試案例專案撰寫測試此新方法的程式碼，再執行測試案例專案來測試新方法的正確性，並且再次執行所有已存在的測試案例來保證新方法沒有造成所有已存在方法測試失敗。當您撰寫的程式碼愈來愈時，測試案例也會隨著累積成為您測試程式碼的軟體資產。

使用 DelphiFor iOS 開發 iOS App 真的簡單又方便，但不管您開發的 App 是多簡單或是多複雜，您都會希望最後能把開發的 App 分發到實際的 iOS 設備中執行，這才是使用 DelphiFor iOS 最後的目的。因此現在讓我們說明如何實際的開發一個 iPad Mini App 並且分發到真正的 iPad Mini 機器中執行，在您瞭解了這整個流程之後您也可以使用這個流程來開發和部署您的 App。

7 開發和分發 iOS App 到 iOS 設備中

在 DelphiFor iOS 中要分發 App 到實際的 iOS 設備中，您需要執行下面的流程:

- 先確定安裝了 XCode 的命令列工具
- 在 DelphiFor iOS 中建立 iOS 設備的遠端組態

➢ 在 DelphiFor iOS 開發和除錯您的 iOS App

➢ 取得 Apple 開發/分發認證

➢ 使用 DelphiFor iOS 的部署管理員分發您的 iOS App

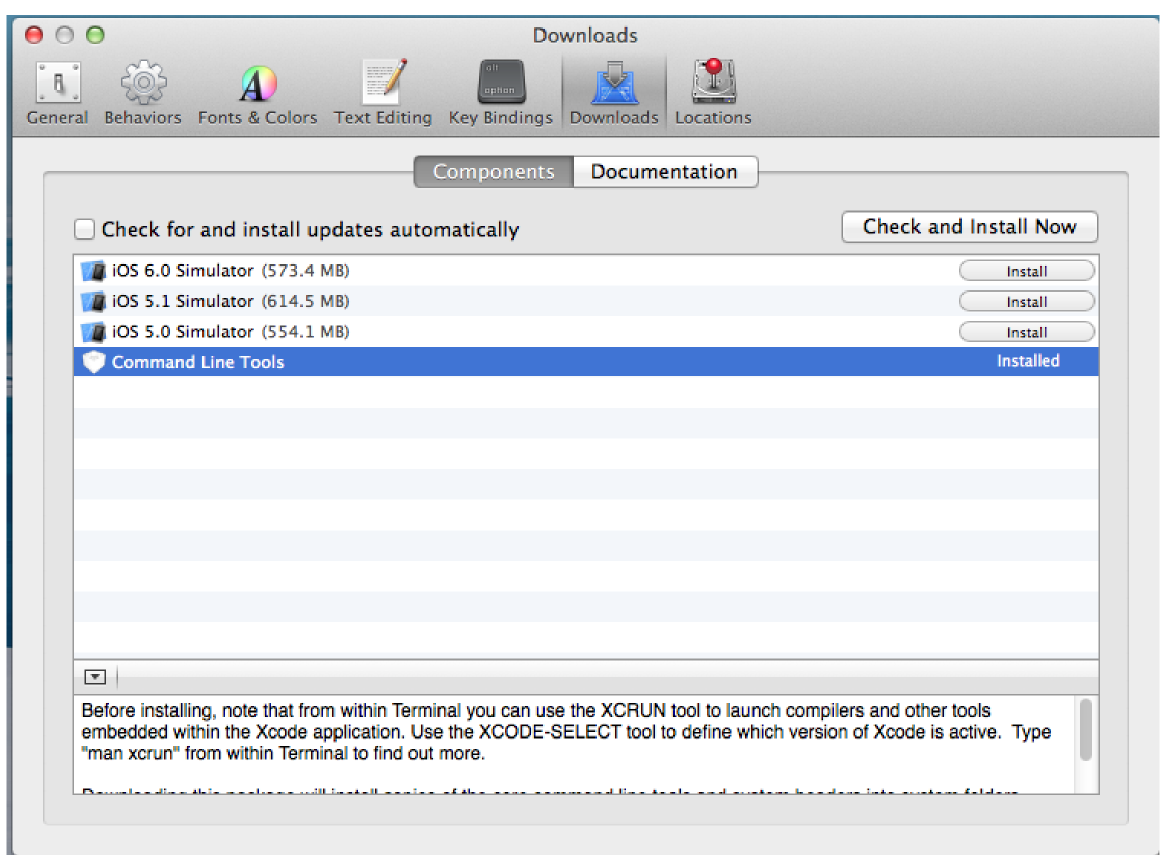
在下面的小節將使用一個實際的範例來說明上述的流程。

確定安裝了 XCode 的命令列工具

在實際能夠分發 iOS App 之前，請確定您的 XCode 已經安裝命令列工具，因為 DelphiFor iOS 需要這些命令列工具來數位簽章分發的 App 到 iOS 設備中。要安裝 XCode 命令列工具，請執行 Mac 中的 XCode，然後點選:

```
[Xcode]|[Preferences...]|[Donwloads]|[Components]|[Command Line
Tools]|[Install]
```

安裝命令列工具，在正確安裝完之後，您應該可以看到類似如下的結果:



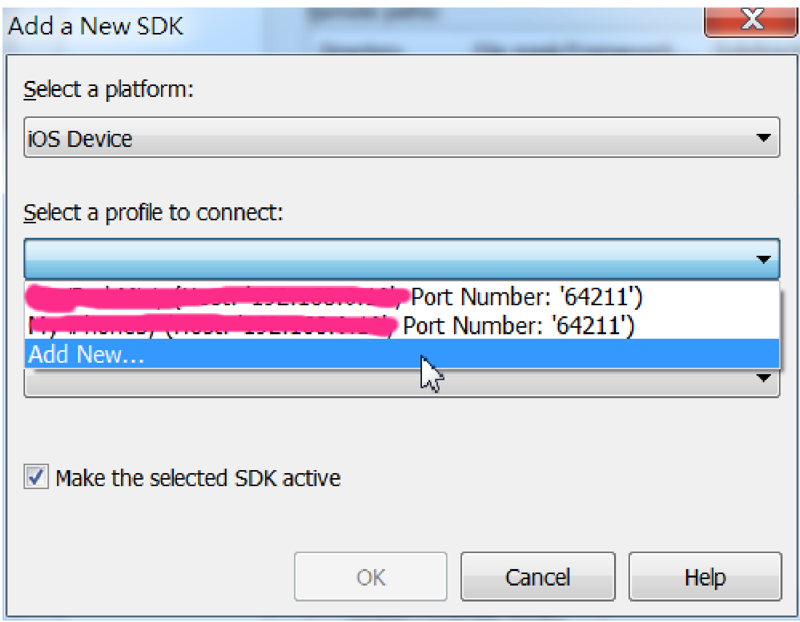
現在我們就可以進行下一步了。

建立 iOS 設備的遠端組態

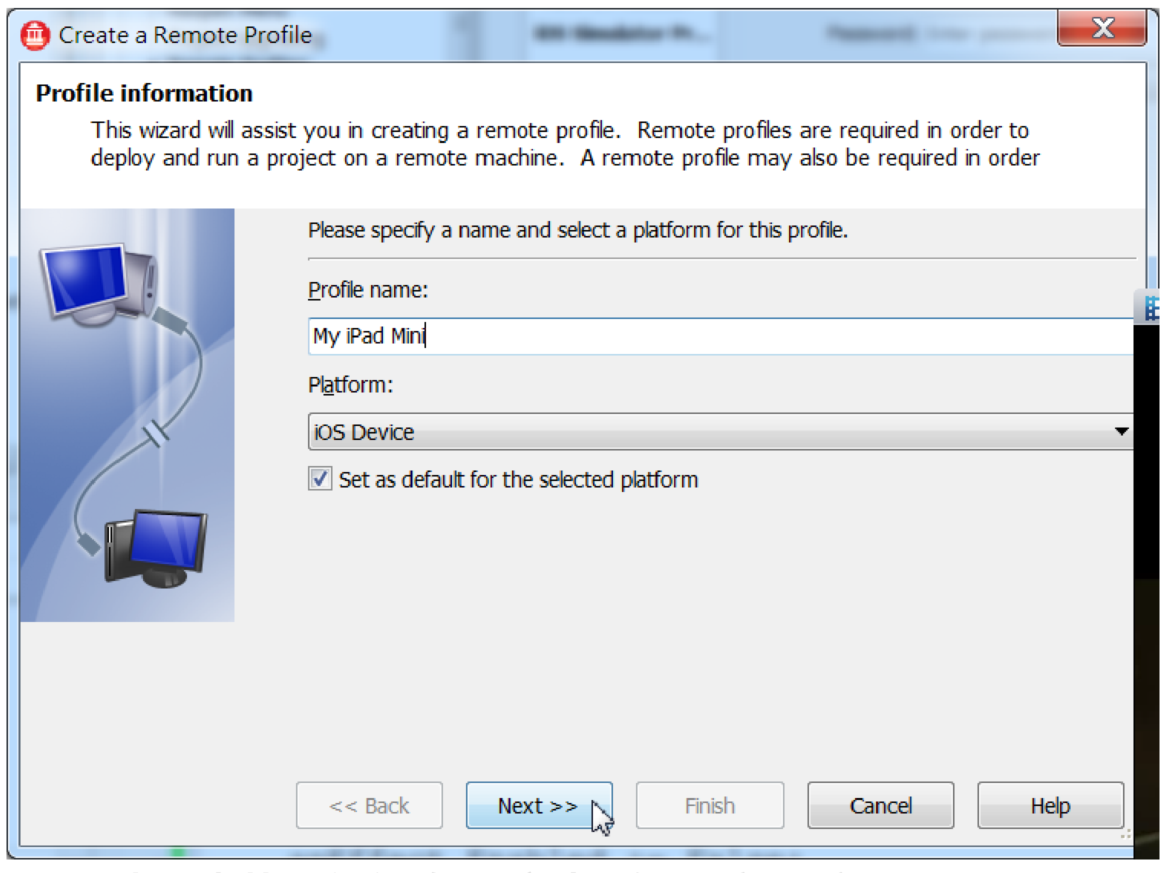
在前面我們說明了如何在 DelphiFor iOS 整合發展環境中建立 iOS 模擬器的組態以便我們開發和測試 iOS App。如果我們需要在 DelphiFor iOS 整合發展環境中實際分發 iOS App，我們也需要建立 iOS 設備的遠端組態，才能藉由 Delphi for iOS 和 XCode 的命令列工具把 App 自動分發到 iOS 設備中。

> 在建立 iOS 設備遠端組態之後請先確定 Mac 平臺中的 PSServer 已經在執行狀態中。

請在 Delphi for iOS 整合發展環境中點選 Tools|Options 功能表，在 SDK Manager 中點選『Add』按鈕建立遠端組態，此時會出現一個『Add a New SDK』對話盒，請在 Select a platform 欄位中選擇 iOS Device，再於 Select a profile to connect 欄位中選擇『Add New...』選項，如下所示:

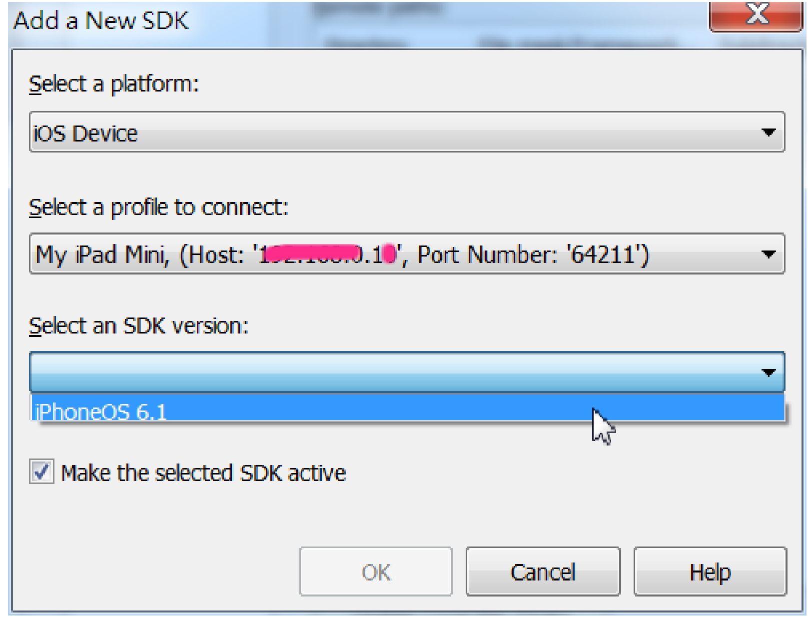


然後在 Create a Remote Profile 對話盒中為您的 iOS 設備取一個名稱並且在 Platform 欄位中選擇建立『iOS Device』平臺組態。例如筆者使用的 iOS Device 是 iPad Mini，因此在下面的對話盒中取名為 My iPad Mini 設備名稱:



版權所有 請勿翻印

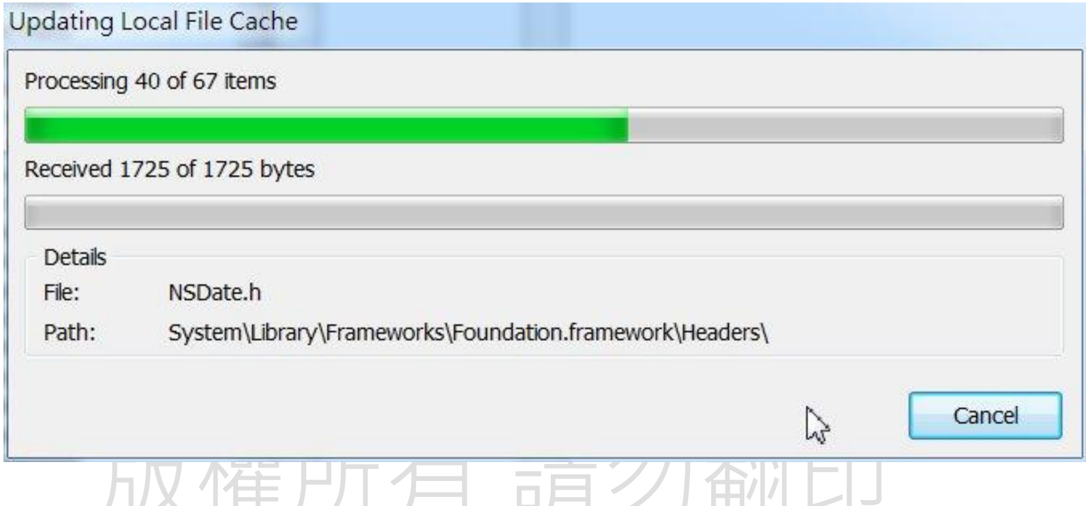
接著點選 Next 按鈕到下一個頁面輸入 Mac 主機的名稱或 IP 位置之後就會回到『Add a New SDK』對話盒，再點選 Select an SDK version 後『Add a New SDK』對話盒就會立刻和 Mac 通訊並未找出此 iOS 設備使用的 SDK 版本，例如下圖就顯示『Add a New SDK』對話盒找到筆者使用的 iPad Mini 是使用 iOS 6.1 的版本，請選擇顯示的版本:



再點選 OK 按鈕之後

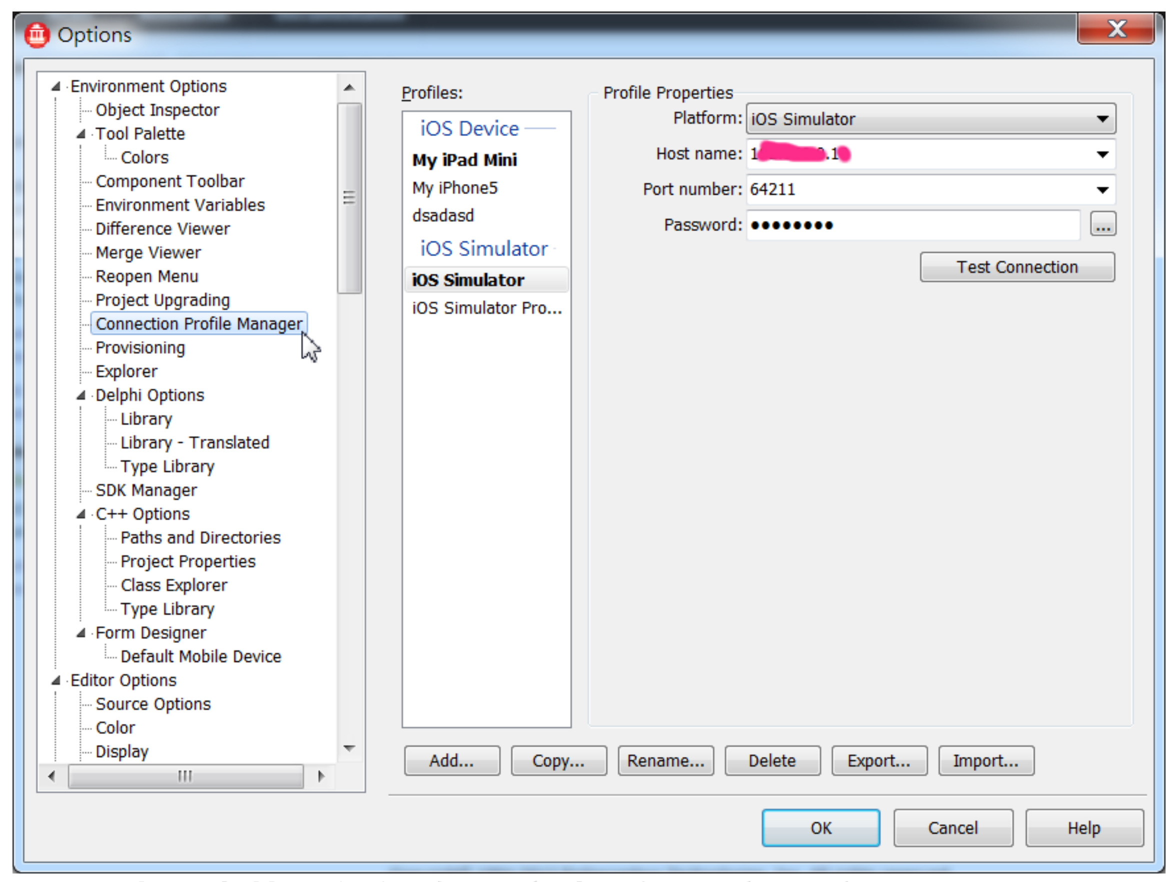
(筆者建議讀者在建立完 iOS 設備的遠端組態之後可以先關閉整個 Options 對話盒，再使用 Tools|Options 功能表開啟 Options 對話盒，再到『SDK Manager』選項建立 SDK 組態)

此時 Delphi for iOS 整合發展環境就會自動根據您的設定拷貝和更新相關的檔案以說明您分發 iOS App，如下所示:



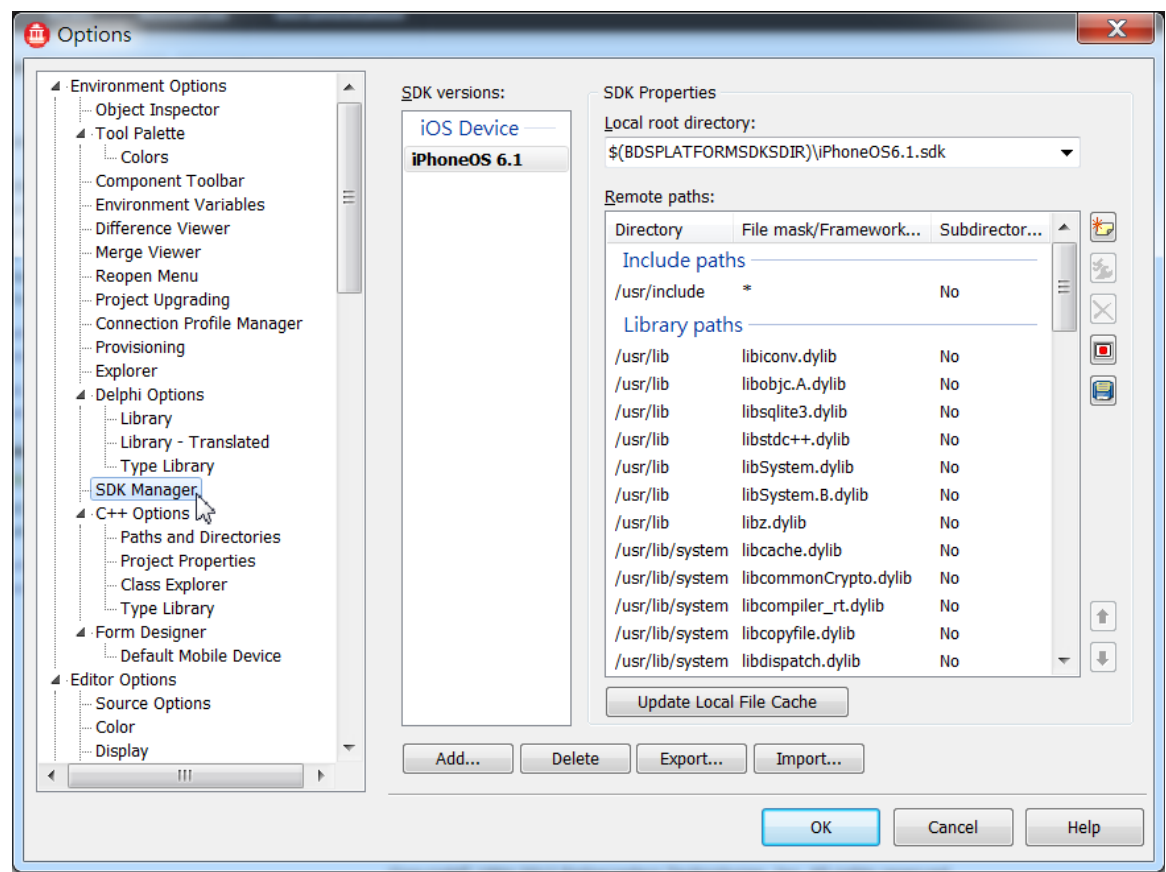
版權所有 請勿翻印

在完成上面的步驟後設定 iOS 設備遠端組態的工作就完成了，此時在您的 Connection Profile Manager 中就應該有類似如下的組態，分別是分發到 iOS 模擬器中的組態和分發到 iOS 實際設備中的組態:



版權所有 請勿翻印

而 SDK Manager 中也會顯示您的 iOS 設備使用的 SDK 版本資訊和相關的檔案:

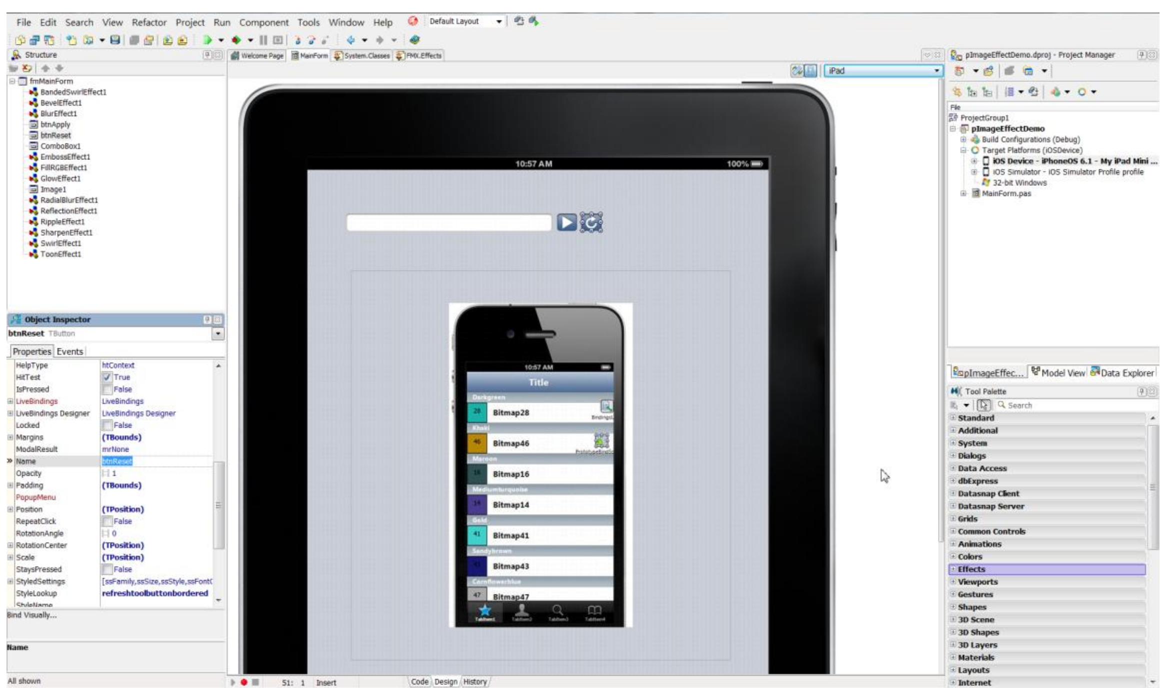


版權所有 請勿翻印

開發 iPad Mini 範例 App

現在請在 Delphi for iOS 整合發展環境中建立一個 FireMonky Mobile Application 專案，在主表單右上方選擇建立 iPad 表單，如下所示:

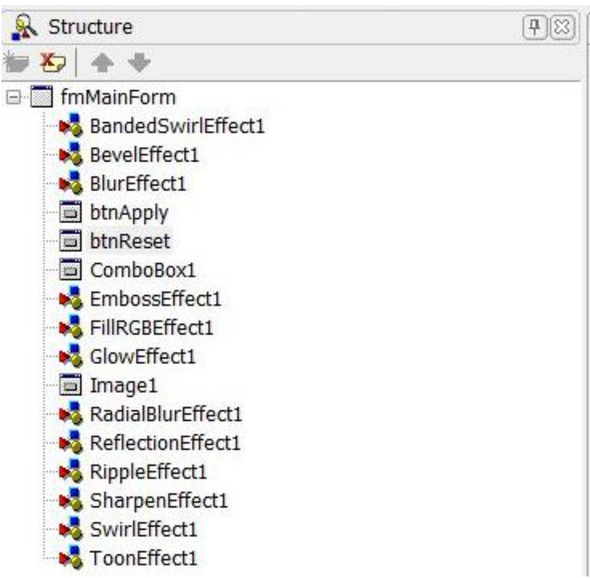
接著在主表單中放入 TImage，TEdit，TComboBox 和 2 個按鈕元件，再從工具盤中隨便選擇數個 Effect 元件，例如 TRippleEffect TShowEffect 等。最後再使用 TImage 元件的 Bitmap 特性的特性值編輯器載入一個影像，如下所示:



這個 iPad 範例 App 的功能是讓用戶可以從 TComboBox 中選擇要對主表單 TImage 元件中的影像執行各種影像效果，並且在 TImage 元件中顯示各種效果執行的結果。

版權所有 請勿翻印

因此現在如果您檢視整合發展環境左上方的樹狀架構視窗，應該可以看到類似如下的結果，我們在主表單中放入的各種效果元件都是屬於主表單的子元件。



現在讓我們撰寫一些程式碼讓這個 iPad App 能夠工作。首先建立主表單的 OnCreate 事件處理函式，它呼叫了 GetAllAvailabelEffects 方法取得主表單中所有效果元件:

```
procedure TfmMainForm.FormCreate(Sender: TObject);
begin
  //取得 FireMonkey 的 Effect
  GetAllAvailabelEffects(ComboBox1.Items);
end;
```

GetAllAvailabelEffects 方法檢視主表單中所有屬於 TEffect 的衍生元件，並且把它的類別名稱和元件參考加入到 TComboxBox 中:

```
procedure TfmMainForm.GetAllAvailabelEffects(sl: TStrings);
var
  anEffect : TEffect;
  iCount : Integer;
begin
  for iCount := 0 to Self.ComponentCount - 1 do
  begin
    if (Self.Components[iCount] is TEffect) then
    begin
      anEffect := Self.Components[iCount] as TEffect;
      sl.AddObject(anEffect.ClassName, anEffect);
    end;
  end;
end;
```

接著為主表單中第一個按鈕實作如下的 OnClick 事件處理函式。當使用者點選此按鈕之後 007 行就從 TComboBox 中取得前面儲存在 TComboBox 中的效果元件參考，010 行先呼叫 ResetEffect 方法以清除以前使用過的效果元件。要讓效果元件影響主表單中的 TImage 元件中的影像，我們只需要在 011 行設定效果組件的 Parent 特性值為 TImage 組件，012 行再設定效果元件的 Enabled 特性值為 True 即可。最後 013 行把目前作用中的效果元件儲存在 oldEffect 物件變數中，以便稍後使用者使用其他效果元件時先移除目前效果元件的作用。

```
001    procedure TfmMainForm.btnApplyClick(Sender: TObject);
002    var
003      anEffect : TEffect;
004    begin
005      if (ComboBox1.ItemIndex = -1) then
006        Exit;
007      anEffect := ComboBox1.Items.Objects[ComboBox1.ItemIndex] as
TEffect;
```

```
008     if (Assigned(anEffect)) then
009     begin
010       ResetEffect(oldEffect);
011       anEffect.Parent := Image1;
012       anEffect.Enabled := True;
013       oldEffect := anEffect;
014     end;
015   end;
```

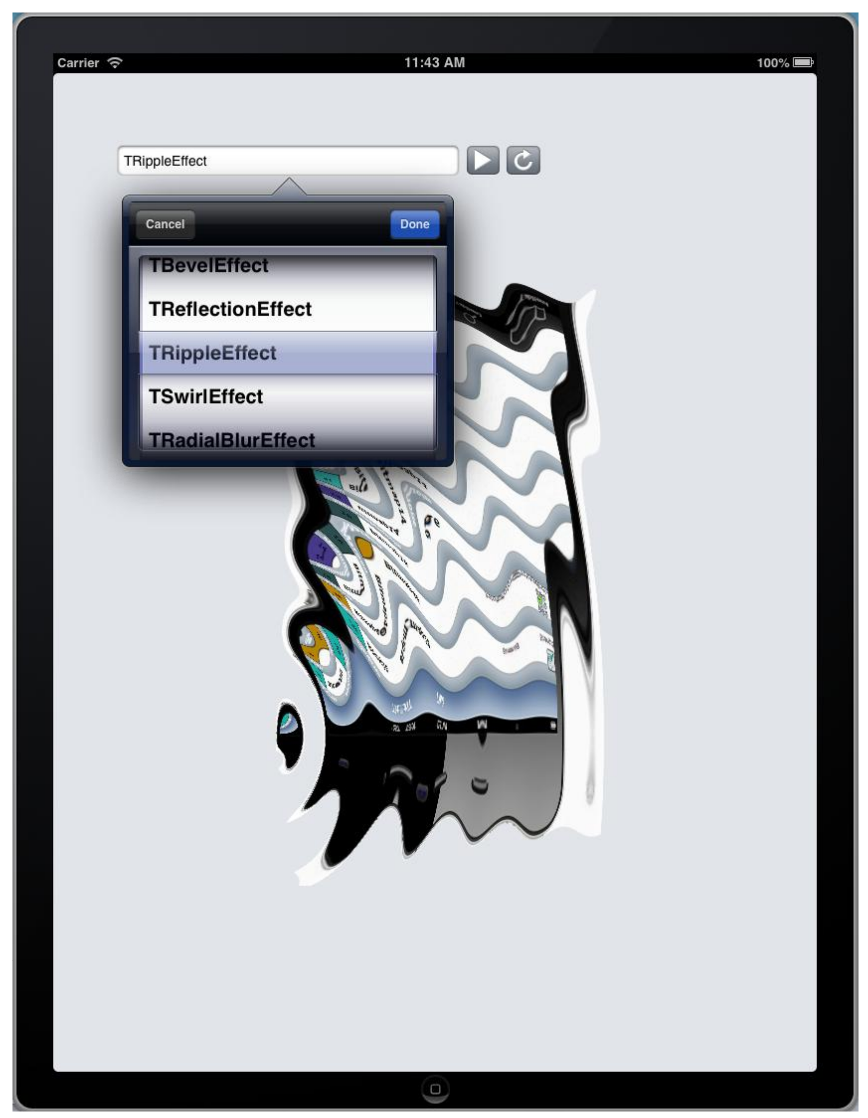
第二個按鈕的 OnClick 事件處理函式是在移除效果組件的作用:

```
procedure TfmMainForm.btnResetClick(Sender: TObject);
begin
  ResetEffect(oldEffect);
end;
```

ResetEffect 方法判斷目前是否有任何效果元件在作用中，如果是的話就先把作用中的效果元件關閉，再設定它的 Parent 特性值為 nil 以便讓 TImage 元件中的影像回復原始的狀態，最後再把 oldEffect 物件變數設定為 nil。

```
procedure TfmMainForm.ResetEffect(anEffect: TEffect);
begin
  if (Assigned(anEffect)) then
  begin
    anEffect.Enabled := False;
    anEffect.Parent := nil;
    oldEffect := nil;
  end;
end;
```

我們可以準備執行這個範例 iPad App 了，如果一切順利讀者就可以在 iOS 模擬器中看到它啟動 iPad 模擬器，請選擇 TComboBox 中的效果元件，接著點選第一個按鈕就可以看到類似下面的執行結果了:



現在我們已經成功的開發了一個 iPad App，但我們如何才能夠把這個 iPad App 分發到真正的 iPad Mini 中執行呢?

取得 Apple 認證

在實際能夠分發您使用 Delphi for iOS 開發的 App 到 iOS 設備之前，您需要具備 iOS 開發人員計畫資格。您可以使用 2 種方式取得 iOS 開發人員計畫資格:

- 一是自行加入，您需要支付每年 99 美元的費用以加入這個計畫
- 或是加入您公司的 iOS 企業開發計畫，例如筆者就是藉由這個方法取得認證，Embarcadero 允許筆者參加 Embarcadero 的 iOS 企業開發計畫

雖然有 2 種不同的方法可以取得您的 iOS 開發資格，但一旦您取得了資格之後接下來取得認證的步驟就差不多，因此本書將說明筆者如何取得企業開發計畫並且據以取得分發認證以部署筆者使用 Delphi for iOS 開發的 App 到 iPad Mini 設備之中。

這整個流程如下:

➢ 先申請 Apple ID

➢ 向 iOS 企業計畫管理者發 EMail 要求加入 iOS 企業計畫

➢ iOS 企業計畫管理者會把您加入 iOS 企業計畫，Apple 會自動寄一封 EMail 告訴您已經加入了 Apple 的 iOS 企業開發計畫:

Welcome to the iOS Developer Enterprise Program

Dear Gordon Li,

Thank you for joining the iOS Developer Enterprise Program. You now have access to a comprehensive set of development tools and resources to assist you in developing and distributing innovative apps for iPhone, iPad, and iPod touch.

How to get started
Learn how to take advantage of your program resources and benefits.
iOS Enterprise Developer Program Getting Started Guide

Access your account
Log in to Member Center to manage your account and access technical resources.

Best regards,

Apple Developer Program Support

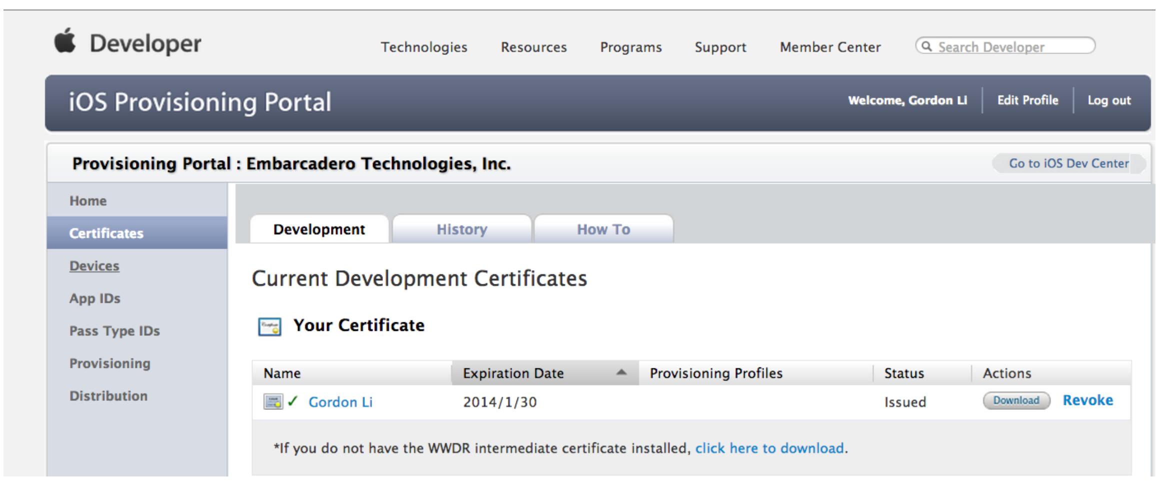
➢ 請使用流覽器前往:

```
https://developer.apple.com/devcenter/ios/index.action
```

並且使用您的 Apple ID 登錄:

在成功登錄之後請點選流覽器右上方的『iOS Provisioning Portal』準備向 Embarcadero 的 iOS 企業計畫管理者申請筆者 iPad Mini 的認證。

首先點選流覽器左方的 Certificates 專案，然後點選下載 AppleWWWDRCA.cer 檔案:

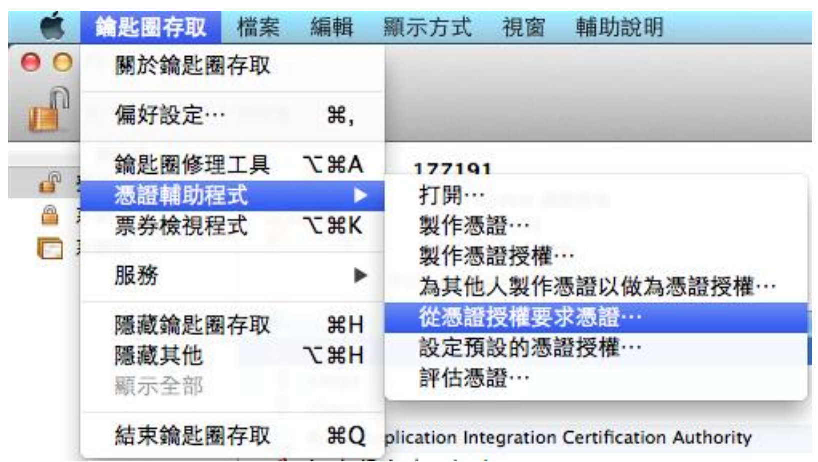


昨天
2013/1/30

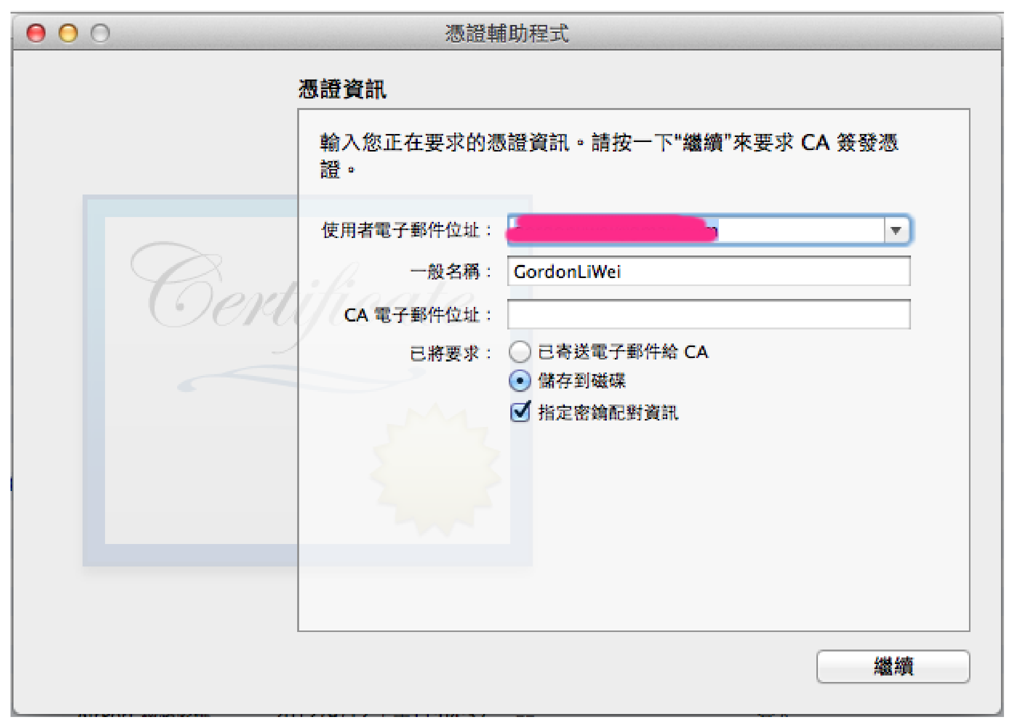
AppleWWDRCA.cer
https://developer.apple.com/certificationauthority/AppleWW…
在 Finder 中顯示　從清單中移除

下載之後按兩下 AppleWWWDRCA.cer 檔案就會自動啟動鑰匙圈存取程式，在其中讀者可以看到 Apple WorldWide 開發資訊已經註冊:

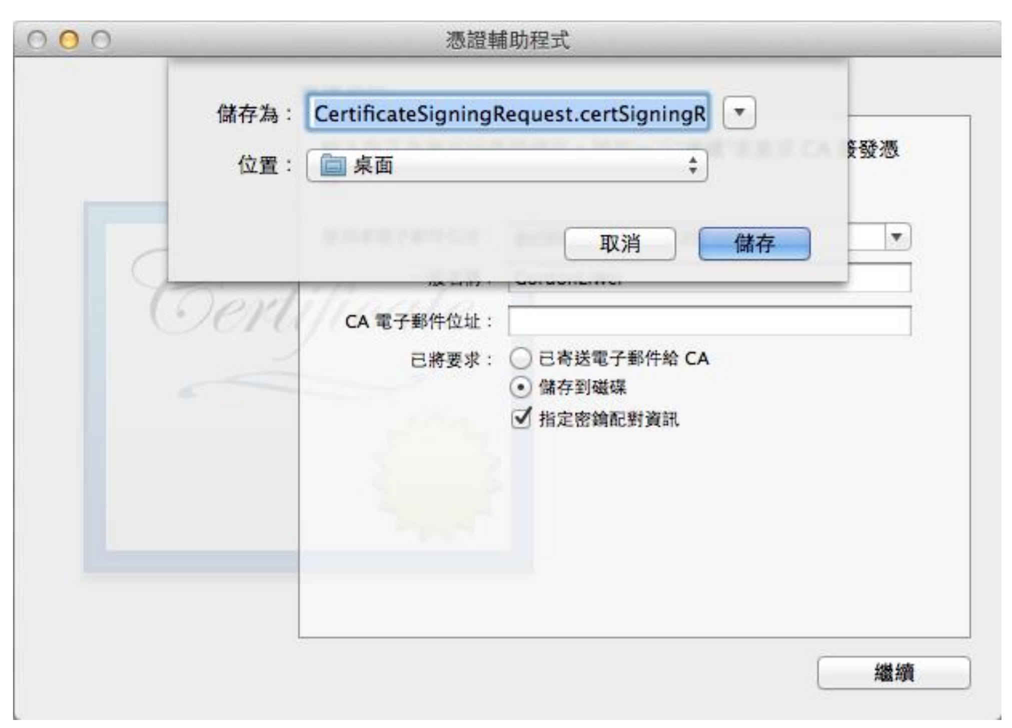
接著請點選鑰匙圈存取程式功能表，選擇從憑證授權機構(對筆者來說就是 Embarcadero)要求憑證，如下所示:



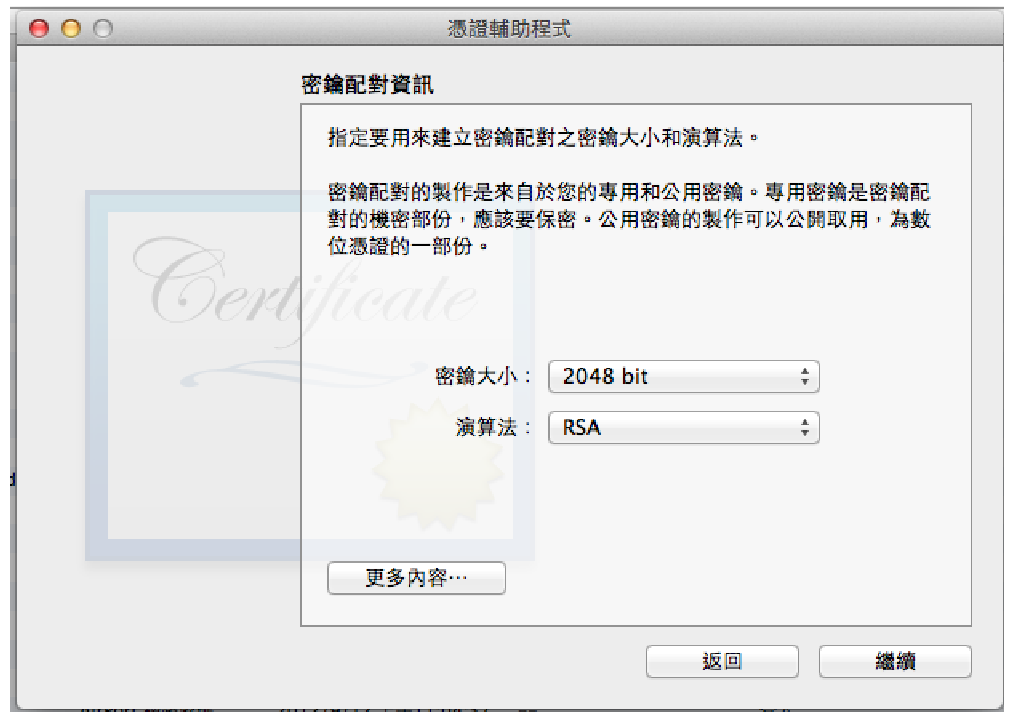
在憑證輔助程式中輸入您的資訊:



點選繼續憑證輔助程式會在桌面儲存一個檔案:CertificateSigningRequest.certSigningRequest:



請選擇 RSA 演算法，點選繼續

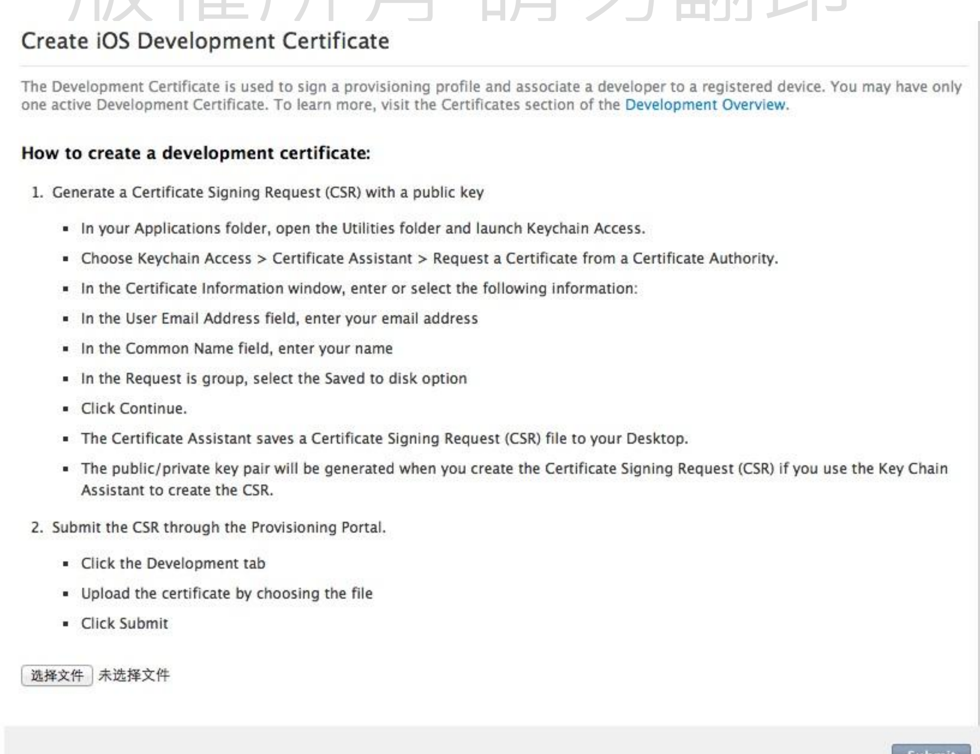


點選繼續之後就會在 Mac 的桌面看到產生的 CertificateSigningRequest.certSigningRequest:

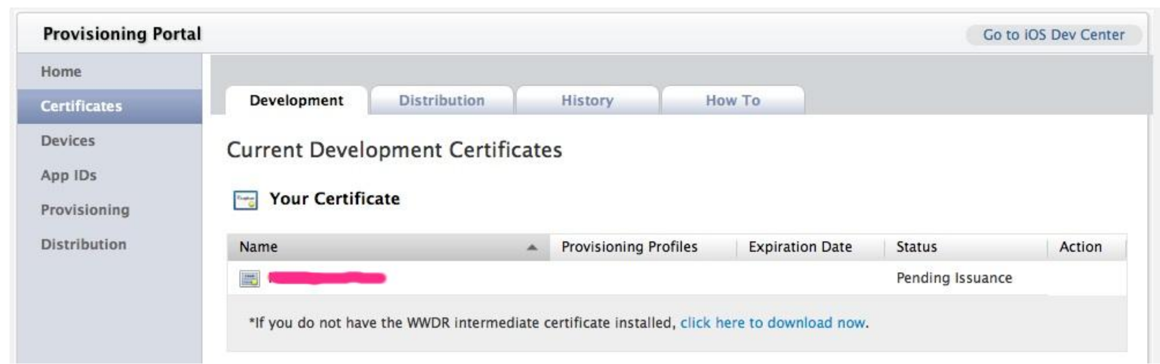
回到流覽器的 Certificates 項目，點選其中的『Request Certificate』按鈕:

在下面的畫面中選擇前面儲存的 CertificateSigningRequest.certSigningRequest 檔案:

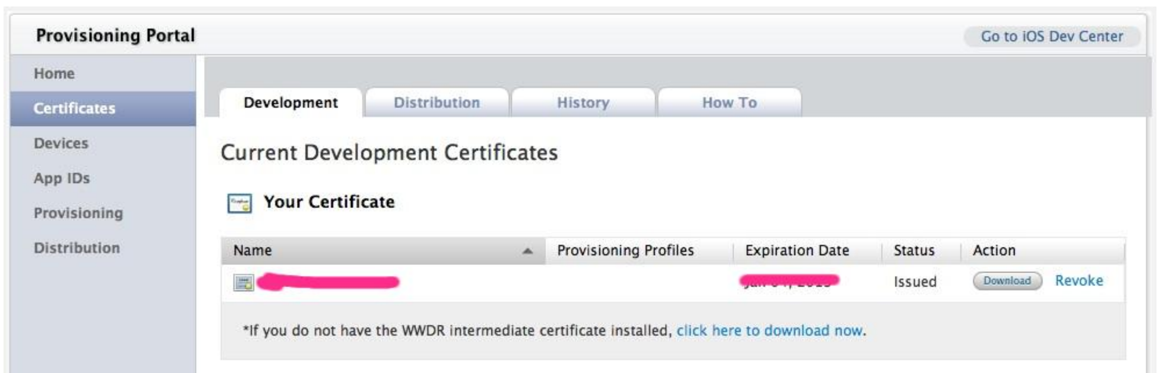
版權所有 請勿翻印



然後點選『Submit』按鈕提出授權要求現在您就需要等待了，等待您的管理員核准您的授權請求。此時在 Certificates 項目中您會看到您的授權請求在 Pending 狀態，等待管理員批准:

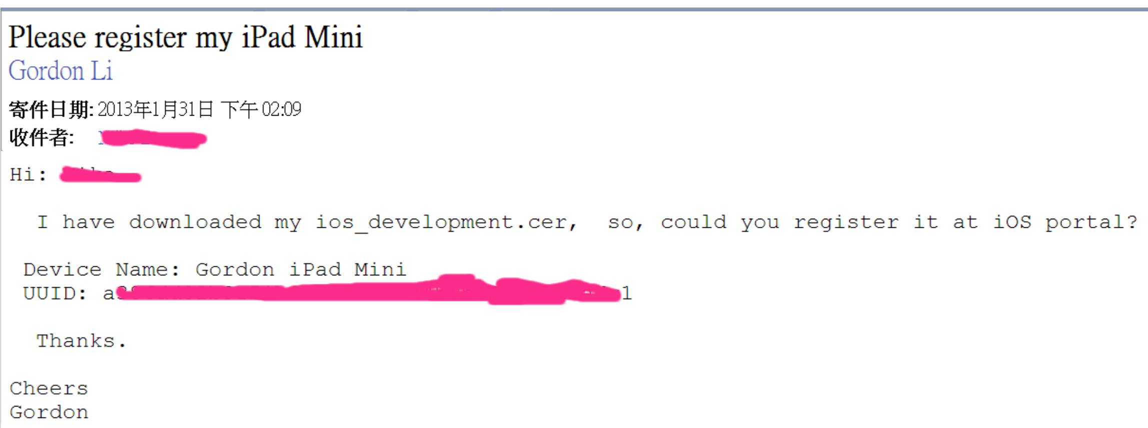


一旦您的管理員批准您的授權請求之後您就可以在 Certificates 項目中看到授權被核發了(Issued):



現在您就可以點選其中的『Download』按鈕正式下載您的 iOS 開發授權認證，筆者的授權檔案是 ios_development.cer:

最後按兩下此授權認證檔案之後認證資訊就會成功寫入您的 Mac 機器中，最後一個步驟就是連結您的授權認證檔案和您的 iOS 設備，在筆者的範例中就是筆者使用的 iPad Mini。



Please register my iPad Mini
Gordon Li
寄件日期: 2013年1月31日 下午 02:09
收件者:

Hi:

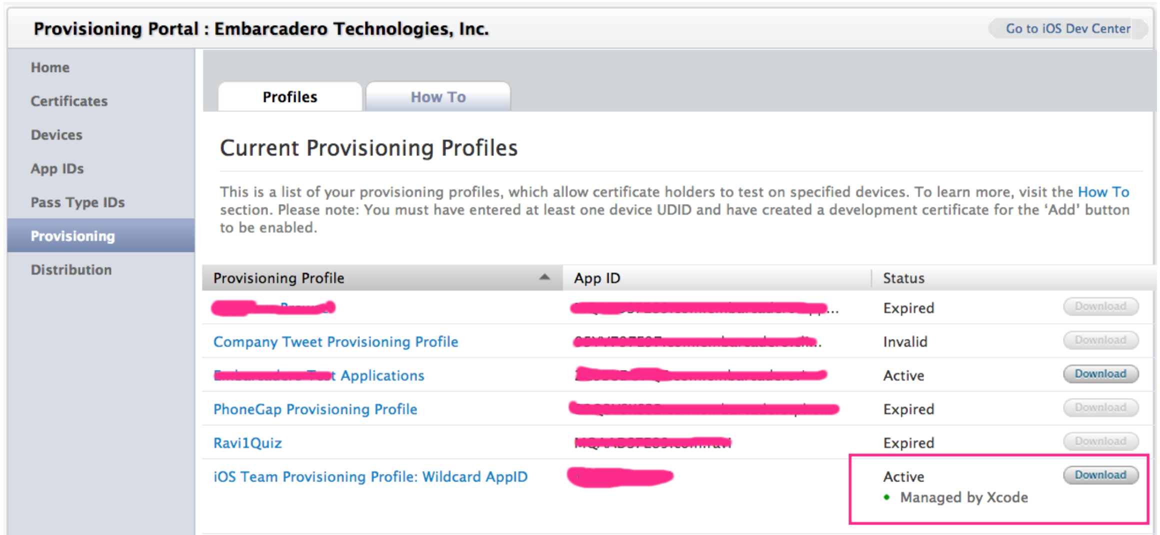
I have downloaded my ios_development.cer, so, could you register it at iOS portal?

Device Name: Gordon iPad Mini
UUID: a

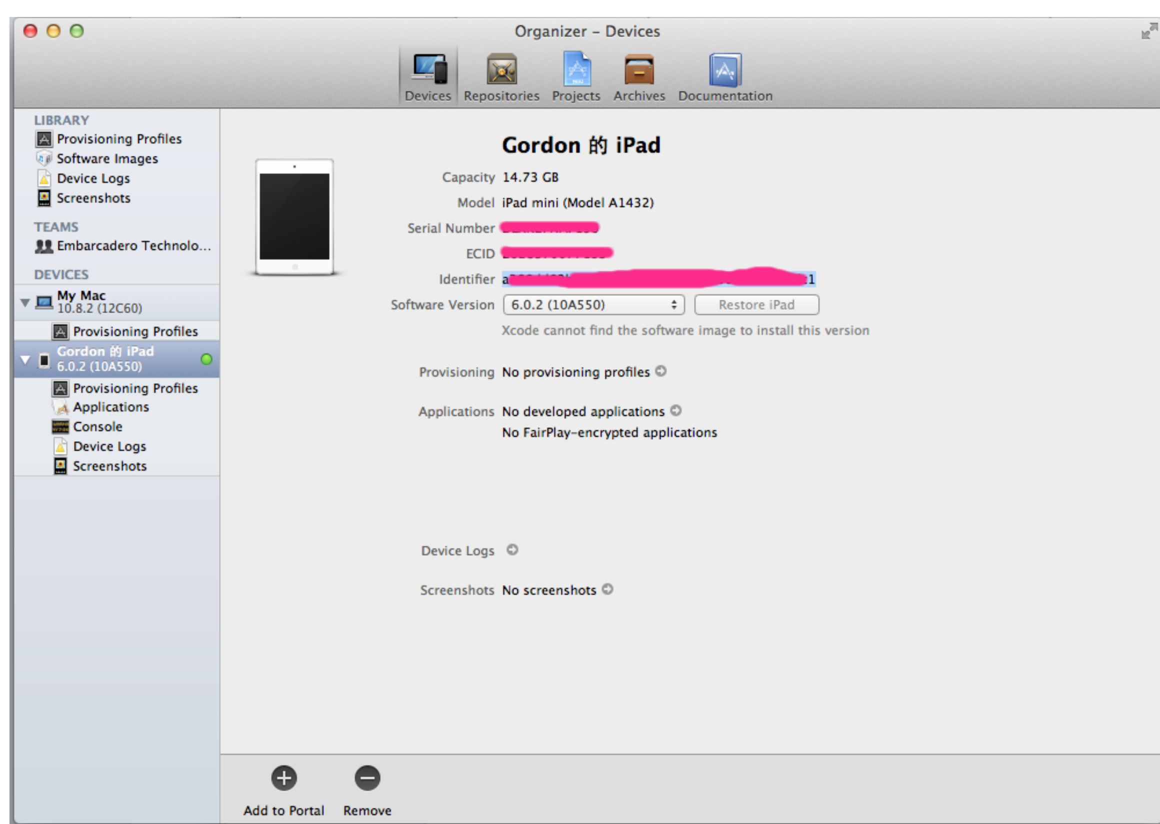
Thanks.

Cheers
Gordon

最後到 Provisioning 頁次下載可使用的認證檔案，再按兩下它以便把認證資訊匯入到 XCode 中:

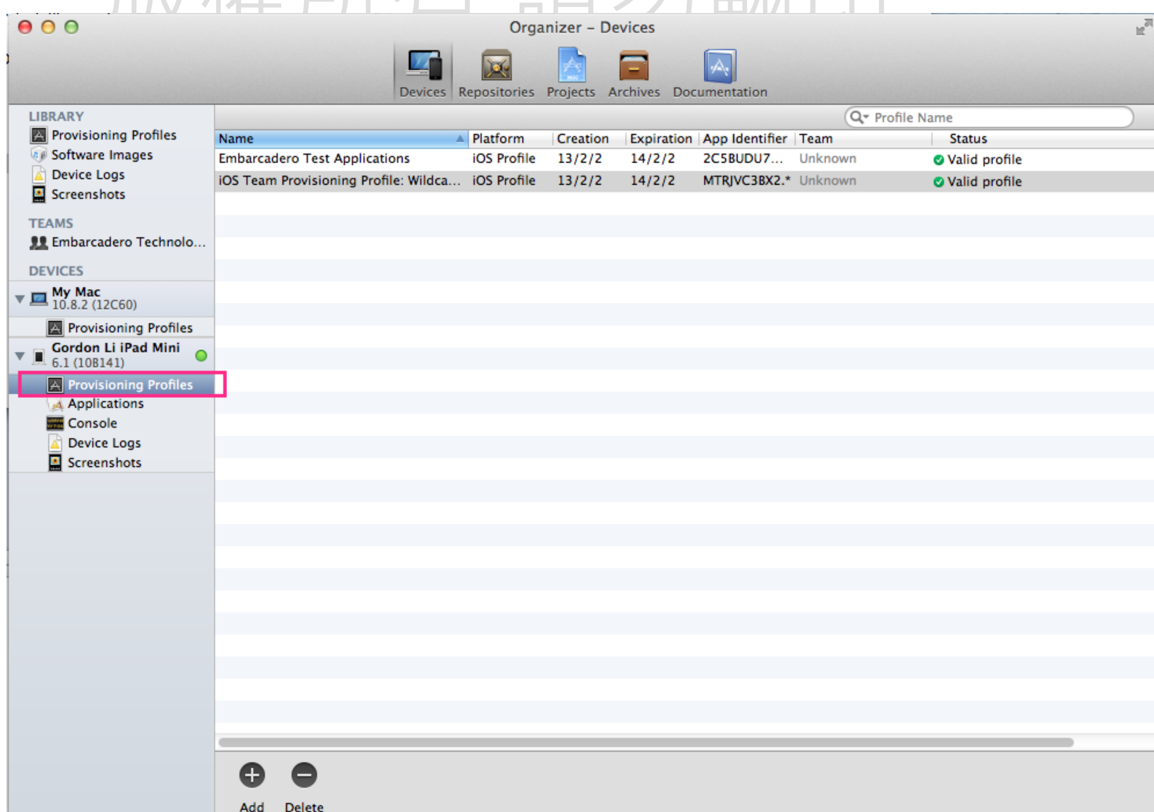


一旦完成這些步驟之後請讀者執行 XCode 並且連結您的 iOS 設備到 Mac 機器，點選 XCode 的 Windows | Organizer 功能表，就可以看到類似如下的畫面，XCode 成功的連結了筆者的 iPad Mini:



點選它的『Provisioning Profiles』頁次就可以看到授權認證資訊:

版權所有 請勿翻印



筆者使用 iPad Mini 執行設定程式，於一般 | 描述檔專案就可以看到授權認證資訊被分發到筆者的 iPad Mini 中:

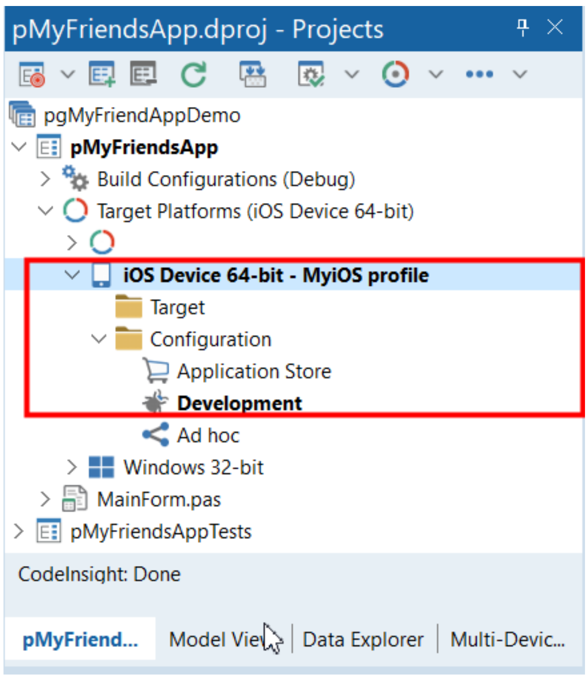
現在筆者的 iPad Mini 已經處於『已驗證』狀態，也代表筆者可以正式使用 Delphi for iOS 部署和分發 iOS App 了。

使用 Delphi for iOS 部署和分發 iOS App 非常的簡單，下一節就會說明。

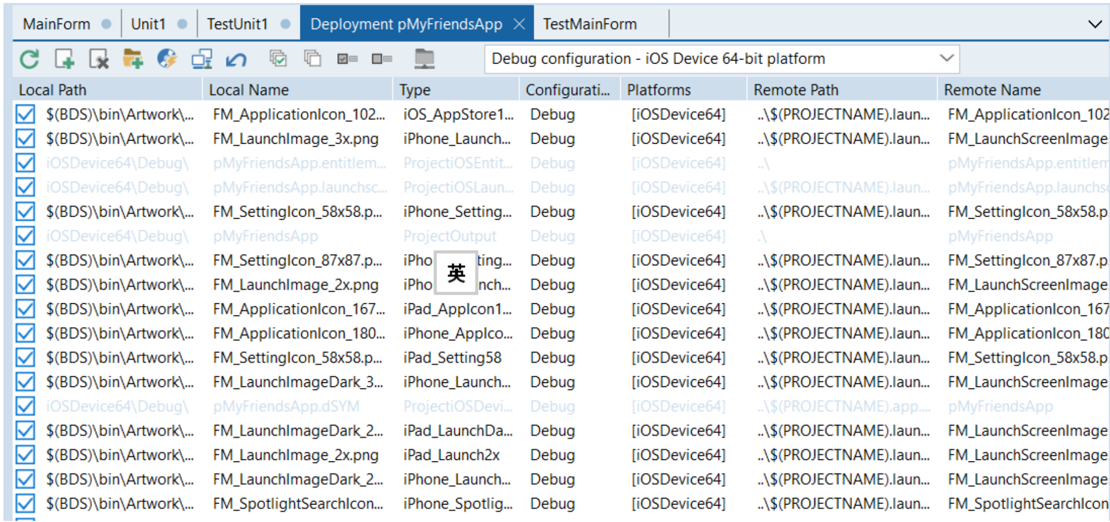
使用部署管理員分發您的 iOS App

版權所有 請勿翻印

對於簡單的iOS App專案，讀者只需要在，項目管理員的Target Platforms節點中選擇使用 iOS 遠端組態，再編譯執行 iOS App 專案即可：

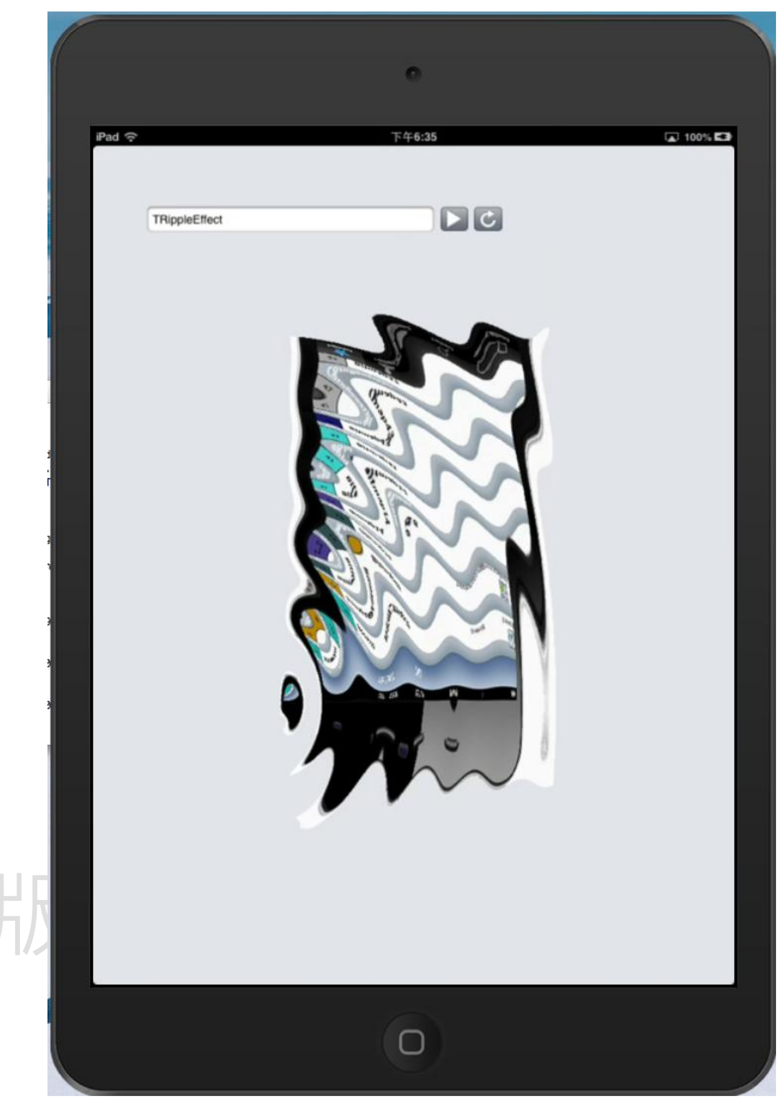


當然對於需要隨著 iOS App 專案分發的功能和額外的檔案，讀者需要使用整合發展環境中的分發精靈來幫助分發複雜的 iOS App 專案，讀者可以點選 Project | Deployment 功能表來啟動分發精靈:



例如對於本節的範例 iPad App 由於只使用 FireMonkey 框架並沒有使用資料庫功能或是其他額外的檔案，因此我們只需要在項目經理中選擇 iOS 遠端組態，再執行此範例 iPad App 之後，Delphi for iOS 就會自動編譯和分發此範例 iPad App 到筆者使用的 iPad Mini 中執行。下面的畫面就是此範例 iPad App 執行在筆者的 iPad Mini 中的結果，讀者可以看到它的執行結果和前面執行在 iPad 模擬器中是一樣的。

版權所有 請勿翻印



筆者是使用 Mac 的 Reflector 軟體截取 iPad Mini 的執行畫面。如果讀者對於 Reflector 有興趣，可以參考 http://www.reflectorapp.com

好了，現在讀者應該瞭解了使用 Delphi for iOS 分發 iOS App 是非常簡單的工作了。

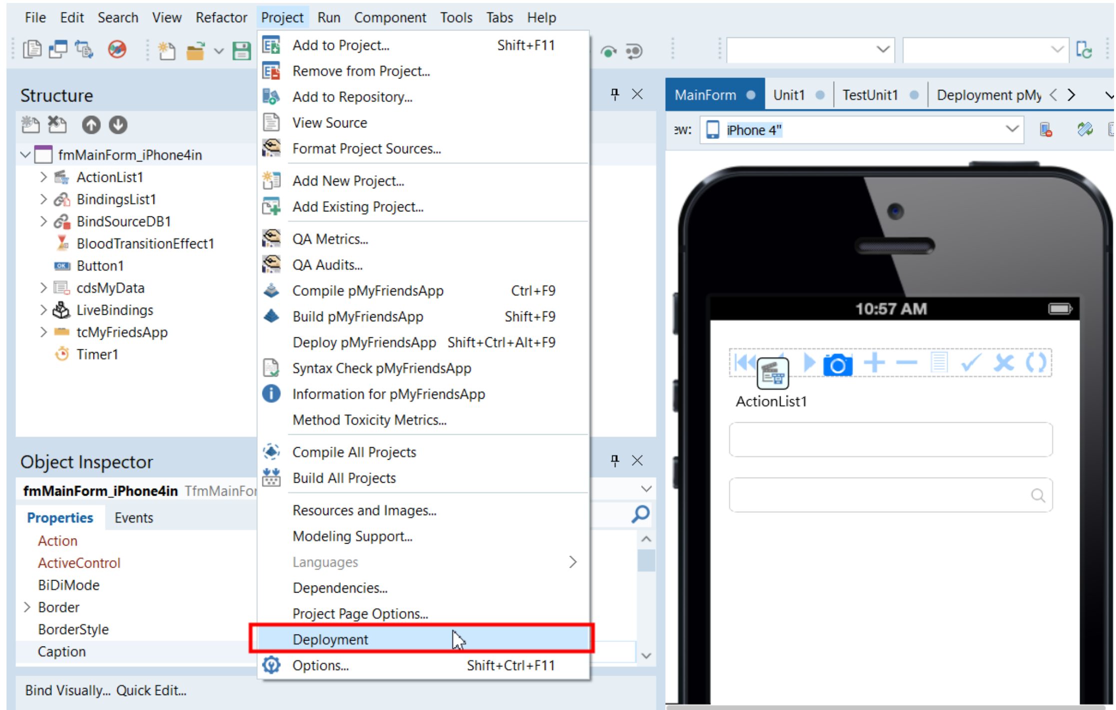
8 分發複雜的 iOS App

我們即將進入本書最後的小節，討論如何分發除了 App 本身之前還需要分發其他額外的檔案。

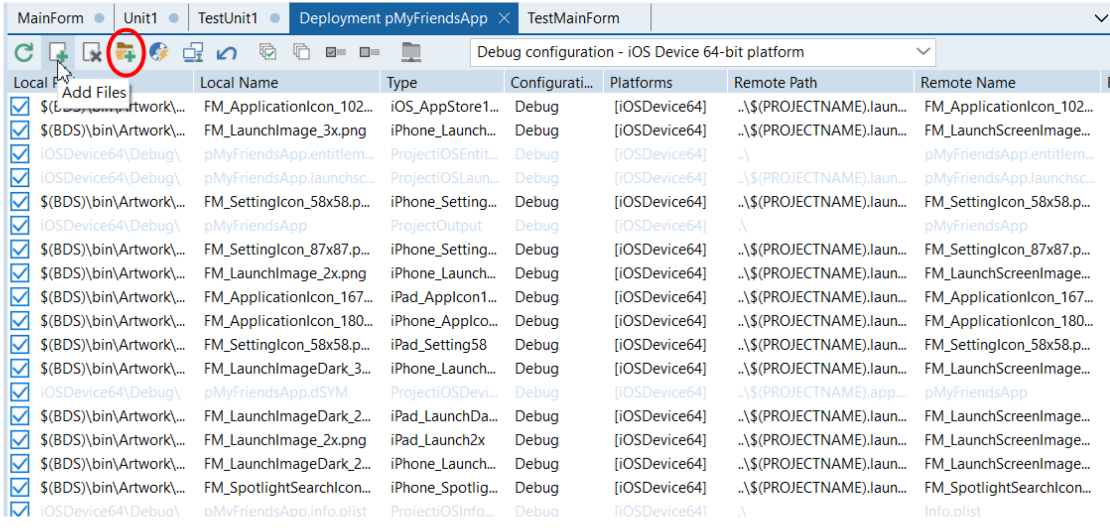
請回到前面討論的 iPhone 範例 App。由於這個範例使用了 DataSnap 技術來處理資料，因此當我們要分發此範例 App 到模擬器中或是到實際的 iOS 設

備中時，都需要分發 DataSnap 相關的函式庫和檔案，這就需要使用整合發展環境中分發管理員的幫忙了。

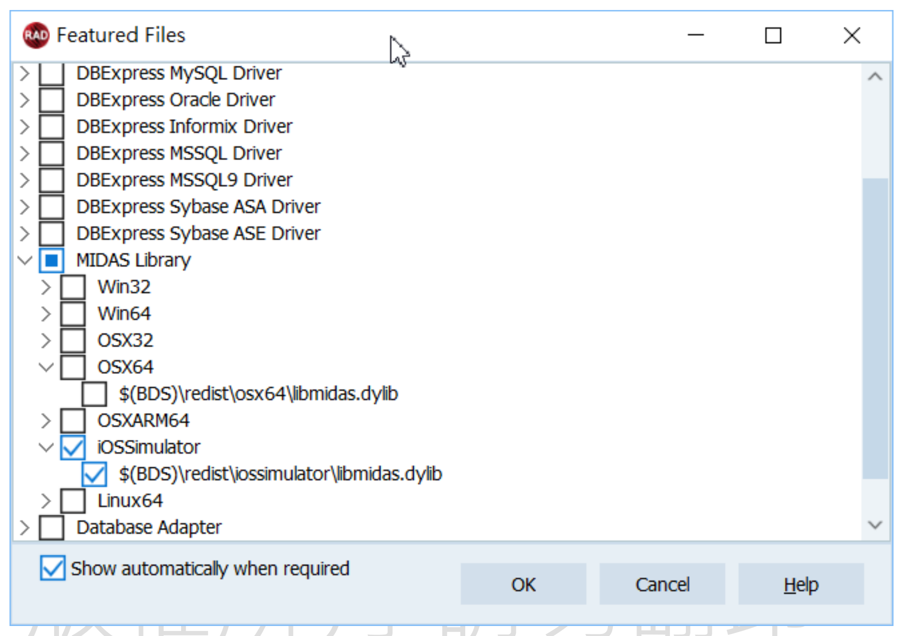
請在整合發展環境中重新開啟 pMyFriendsApp 範例 App，然後點選主功能表的 Project|Deployment 啟動分發管理員，如下所示:



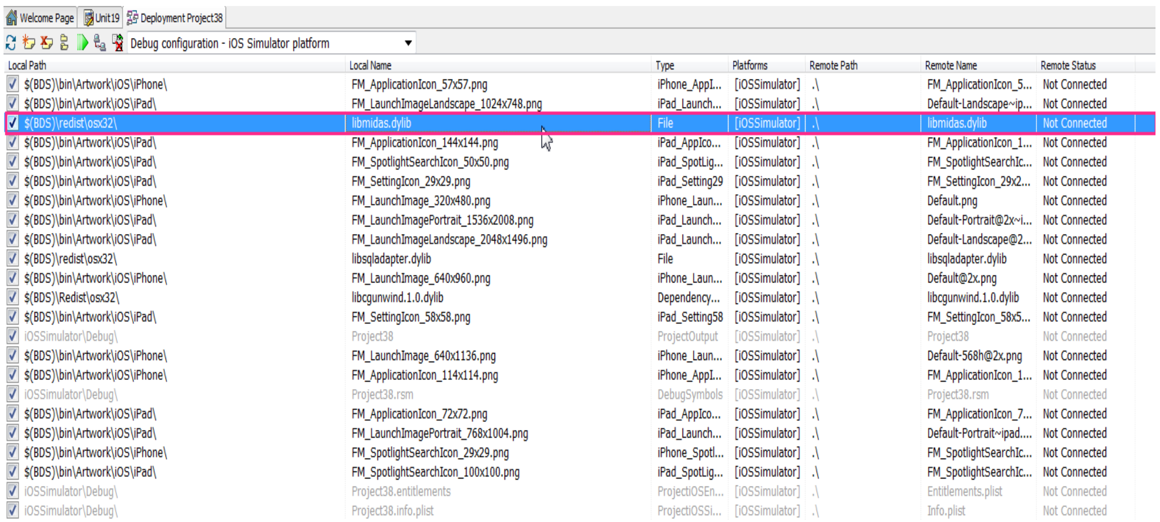
分發管理員接著會顯示如下的視窗，顯示目前這個範例 App 在分發時所有相關的檔案。請點選分發管理員視窗左上方的『Add Featured Files』圖像，如下所示:



點選『Add Featured Files』圖像之後整合發展環境會顯示 Featured Files 對話盒，您可以從其中點選您需要分發的功能檔案。例如現在我們需要分發 DataSnap 功能的相關檔案，因此請點選 Featured Files 對話盒中的 Midas Library 選項，從其下再勾選 iOSSimulator 項目，在 iOSSimulator 項目下您就可以看到分發管理員會分發『$(BDS)\redist\osx32\libmidas.dylib』檔案，如下所示:



接著點選 Featured Files 對話盒的 OK 按鈕後，在分發管理員中就會看到 libmidas.dylib 已經加入到分發檔案的清單中，如下所示:

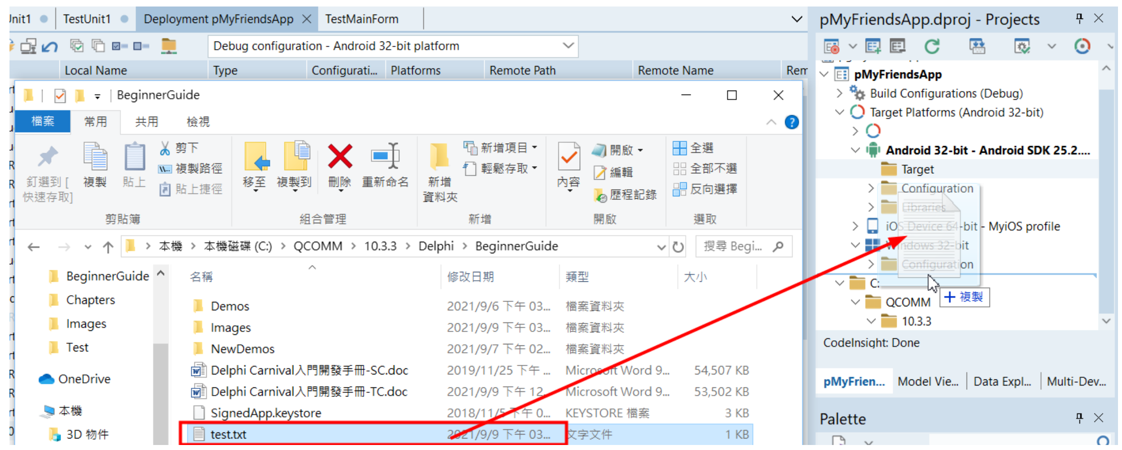


Local Path	Local Name	Type	Platforms	Remote Path	Remote Name	Remote Status
$(BDS)\bin\Artwork\iOS\iPhone\	FM_ApplicationIcon_57x57.png	iPhone_Appl...	[iOSSimulator]	.\	FM_ApplicationIcon_5...	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_LaunchImageLandscape_1024x748.png	iPad_Launch...	[iOSSimulator]	.\	Default-Landscape~ip...	Not Connected
$(BDS)\redist\osx32\	libmidas.dylib	File	[iOSSimulator]	.\	libmidas.dylib	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_ApplicationIcon_144x144.png	iPad_AppIco...	[iOSSimulator]	.\	FM_ApplicationIcon_1...	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_SpotlightSearchIcon_50x50.png	iPad_SpotLig...	[iOSSimulator]	.\	FM_SpotlightSearchIc...	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_SettingIcon_29x29.png	iPad_Setting29	[iOSSimulator]	.\	FM_SettingIcon_29x2...	Not Connected
$(BDS)\bin\Artwork\iOS\iPhone\	FM_LaunchImage_320x480.png	iPhone_Laun...	[iOSSimulator]	.\	Default.png	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_LaunchImagePortrait_1536x2008.png	iPad_Launch...	[iOSSimulator]	.\	Default-Portrait@2x~i...	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_LaunchImageLandscape_2048x1496.png	iPad_Launch...	[iOSSimulator]	.\	Default-Landscape@2...	Not Connected
$(BDS)\redist\osx32\	libsqladapter.dylib	File	[iOSSimulator]	.\	libsqladapter.dylib	Not Connected
$(BDS)\bin\Artwork\iOS\iPhone\	FM_LaunchImage_640x960.png	iPhone_Laun...	[iOSSimulator]	.\	Default@2x.png	Not Connected
$(BDS)\Redist\osx32\	libcgunwind.1.0.dylib	Dependency...	[iOSSimulator]	.\	libcgunwind.1.0.dylib	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_SettingIcon_58x58.png	iPad_Setting58	[iOSSimulator]	.\	FM_SettingIcon_58x5...	Not Connected
iOSSimulator\Debug\	Project38	ProjectOutput	[iOSSimulator]	.\	Project38	Not Connected
$(BDS)\bin\Artwork\iOS\iPhone\	FM_LaunchImage_640x1136.png	iPhone_Laun...	[iOSSimulator]	.\	Default-568h@2x.png	Not Connected
$(BDS)\bin\Artwork\iOS\iPhone\	FM_ApplicationIcon_114x114.png	iPhone_Appl...	[iOSSimulator]	.\	FM_ApplicationIcon_1...	Not Connected
iOSSimulator\Debug\	Project38.rsm	DebugSymbols	[iOSSimulator]	.\	Project38.rsm	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_ApplicationIcon_72x72.png	iPad_AppIco...	[iOSSimulator]	.\	FM_ApplicationIcon_7...	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_LaunchImagePortrait_768x1004.png	iPad_Launch...	[iOSSimulator]	.\	Default-Portrait~ipad....	Not Connected
$(BDS)\bin\Artwork\iOS\iPhone\	FM_SpotlightSearchIcon_29x29.png	iPhone_Spotl...	[iOSSimulator]	.\	FM_SpotlightSearchIc...	Not Connected
$(BDS)\bin\Artwork\iOS\iPad\	FM_SpotlightSearchIcon_100x100.png	iPad_SpotLig...	[iOSSimulator]	.\	FM_SpotlightSearchIc...	Not Connected
iOSSimulator\Debug\	Project38.entitlements	ProjectiOSEn...	[iOSSimulator]	.\	Entitlements.plist	Not Connected
iOSSimulator\Debug\	Project38.info.plist	ProjectiOSSi...	[iOSSimulator]	.\	Info.plist	Not Connected

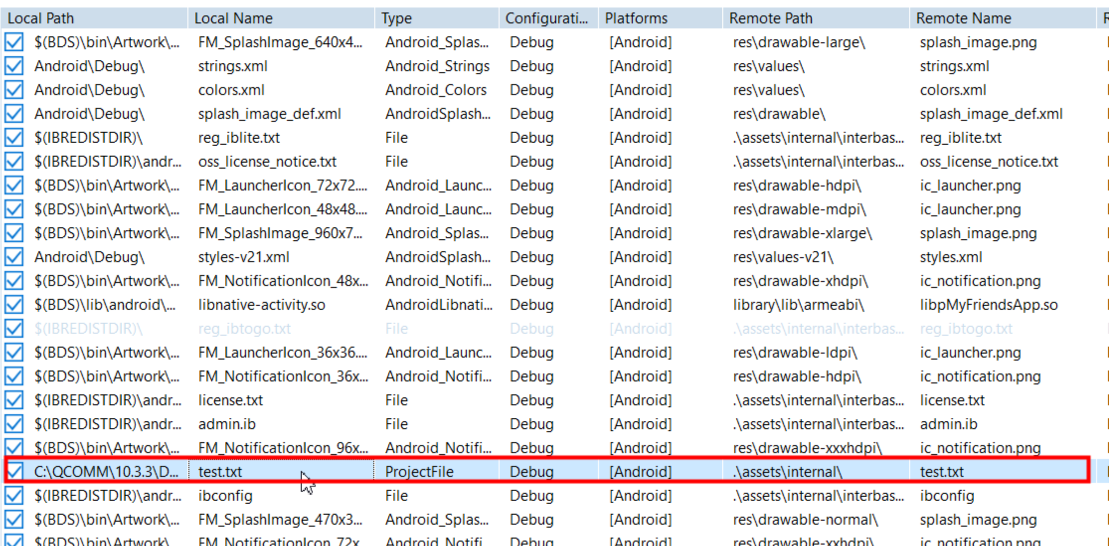
如此就完成了分發額外功能檔案的工作，現在請在專案管理員中點選 iOS 設備的組態，編譯並且執行此範例 App，那麼這個範例 App 應該就可以分發到您的 iOS 設備中執行了。例如下面的畫面就是筆者把這個範例 App 分發到筆者

的 iPad Mini 中執行的結果，這個範例 App 成功的在 iPad Mini 中執行，處理資料並且流覽資料指定的網站了。

另外一種更方便部署額外檔案的方式是直接把要部署的檔案拖曳到 IDE 的專案管理員，如下所示：



此時如果再到分發管理員視窗就可以看到剛才拖曳到專案管理員的檔案已經自動加入到部署檔案之中了。



Local Path	Local Name	Type	Configurati...	Platforms	Remote Path	Remote Name
$(BDS)\bin\Artwork\...	FM_SplashImage_640x4...	Android_Splas...	Debug	[Android]	res\drawable-large\	splash_image.png
Android\Debug\	strings.xml	Android_Strings	Debug	[Android]	res\values\	strings.xml
Android\Debug\	colors.xml	Android_Colors	Debug	[Android]	res\values\	colors.xml
Android\Debug\	splash_image_def.xml	AndroidSplash...	Debug	[Android]	res\drawable\	splash_image_def.xml
$(IBREDISTDIR)\	reg_iblite.txt	File	Debug	[Android]	.\assets\internal\interbas...	reg_iblite.txt
$(IBREDISTDIR)\andr...	oss_license_notice.txt	File	Debug	[Android]	.\assets\internal\interbas...	oss_license_notice.txt
$(BDS)\bin\Artwork\...	FM_LauncherIcon_72x72...	Android_Launc...	Debug	[Android]	res\drawable-hdpi\	ic_launcher.png
$(BDS)\bin\Artwork\...	FM_LauncherIcon_48x48...	Android_Launc...	Debug	[Android]	res\drawable-mdpi\	ic_launcher.png
$(BDS)\bin\Artwork\...	FM_SplashImage_960x7...	Android_Splas...	Debug	[Android]	res\drawable-xlarge\	splash_image.png
Android\Debug\	styles-v21.xml	AndroidSplash...	Debug	[Android]	res\values-v21\	styles.xml
$(BDS)\bin\Artwork\...	FM_NotificationIcon_48x...	Android_Notifi...	Debug	[Android]	res\drawable-xhdpi\	ic_notification.png
$(BDS)\lib\android\...	libnative-activity.so	AndroidLibnati...	Debug	[Android]	library\lib\armeabi\	libpMyFriendsApp.so
$(IBREDISTDIR)\	reg_ibtogo.txt	File	Debug	[Android]	.\assets\internal\interbas...	reg_ibtogo.txt
$(BDS)\bin\Artwork\...	FM_LauncherIcon_36x36...	Android_Launc...	Debug	[Android]	res\drawable-ldpi\	ic_launcher.png
$(BDS)\bin\Artwork\...	FM_NotificationIcon_36x...	Android_Notifi...	Debug	[Android]	res\drawable-hdpi\	ic_notification.png
$(IBREDISTDIR)\andr...	license.txt	File	Debug	[Android]	.\assets\internal\interbas...	license.txt
$(IBREDISTDIR)\andr...	admin.ib	File	Debug	[Android]	.\assets\internal\interbas...	admin.ib
$(BDS)\bin\Artwork\...	FM_NotificationIcon_96x...	Android_Notifi...	Debug	[Android]	res\drawable-xxhdpi\	ic_notification.png
C:\QCOMM\10.3.3\D...	test.txt	ProjectFile	Debug	[Android]	.\assets\internal\	test.txt
$(IBREDISTDIR)\andr...	ibconfig	File	Debug	[Android]	.\assets\internal\interbas...	ibconfig
$(BDS)\bin\Artwork\...	FM_SplashImage_470x3...	Android_Splas...	Debug	[Android]	res\drawable-normal\	splash_image.png
$(BDS)\bin\Artwork\...	FM_NotificationIcon_72x...	Android_Notifi...	Debug	[Android]	res\drawable-xxhdpi\	ic_notification.png

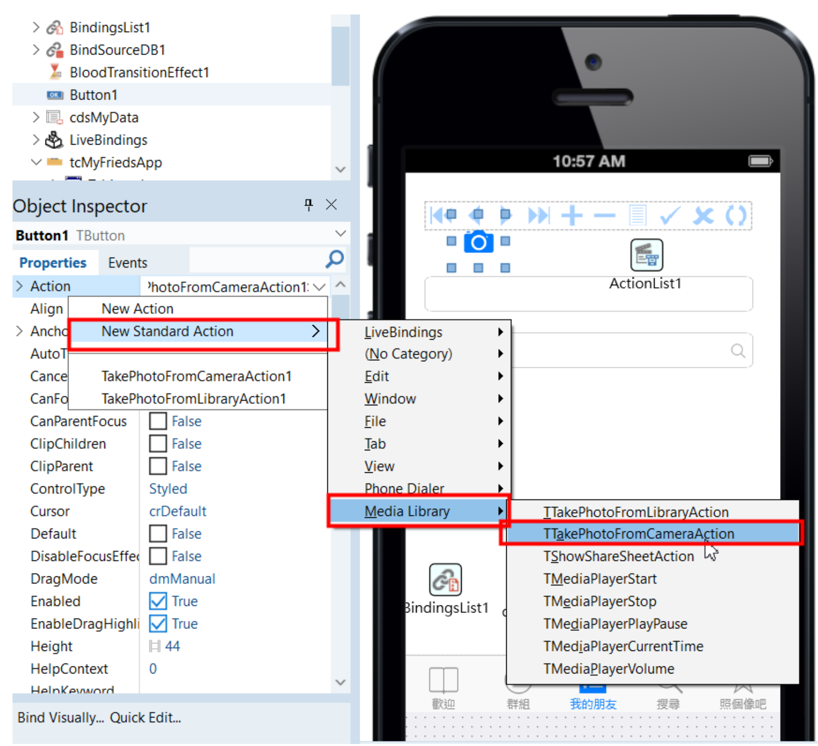
9 使用 iOS 的服務

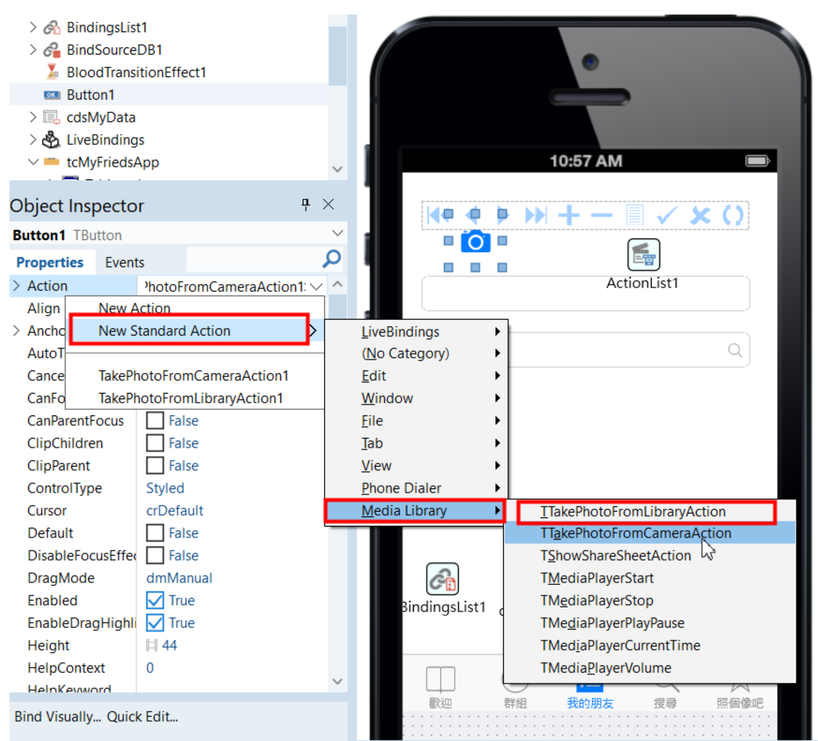
使用 Delphi for iOS 開發 iOS App 的好處就是它把許多 iOS 的服務封裝成了 FireMonkey 的元件，因此您在需要使用這些服務時只需要拖曳這些元件到表單中就可以使用 iOS 服務，而無需重複撰寫繁雜的服務程式碼。現在就讓我們為 pMyFriendsApp 這個範例 App 加上最後一個功能，我們將說明如何使用 iOS 的照相機服務以及如何從 iOS 的像片庫中選擇像片。

Delphi for iOS 把許多的 iOS 服務功能封裝在 TActionList 元件中，您只需要連結 FireMonkey 元件和 TActionList 元件中的服務即可使用 iOS 服務功能。現在就讓我們為主表單中的 TTabControl 加入第 5 個頁次物件 TTabItem，

再於其中放入兩個按鈕元件和一個 TActionList 元件。設定一個按鈕元件的 StyleLookup 特性值為 cameratoolbuttonbordered，另外一個按鈕元件的 StyleLookup 特性值為 organizetoolbuttonbordered，如下所示:

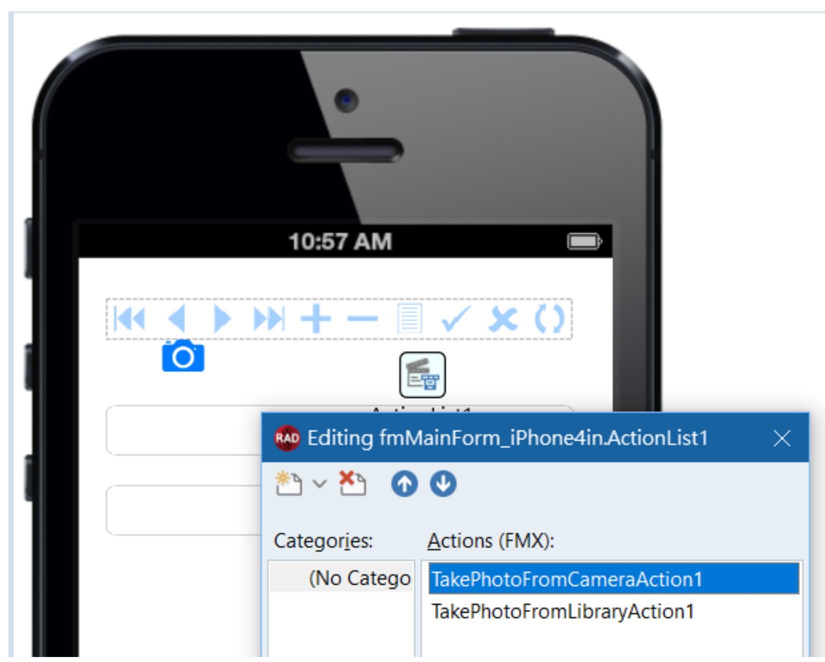
接著分別點選這兩個按鈕元件，在物件檢視器中點選它們的 Action 特性，選擇 New Standard Action|Media Library|TakePhotoFromCameraAction 和 New Standard Action|Media Library|TakePhotoFromLibraryAction 這兩個特性值，TakePhotoFromCameraAction 特性值就可以呼叫 iOS 的照片機服務，而 TakePhotoFromLibraryAction 特性值就可以啟動像片庫，如下面的 2 個圖形所示:

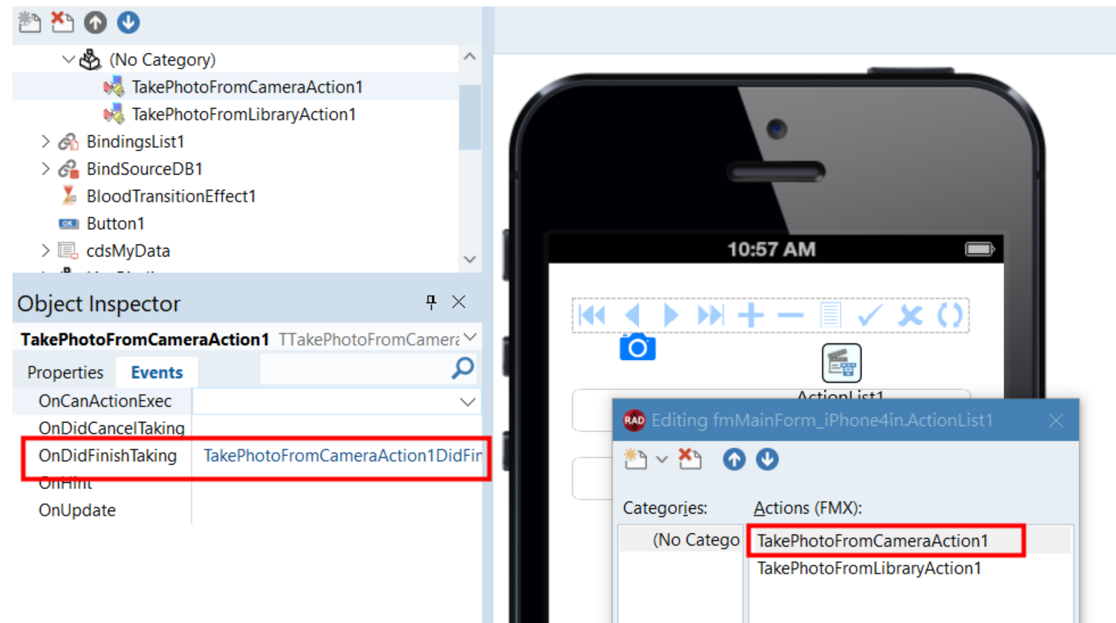




接著按兩下主表單中的 TActionList 元件啟動它的元件編輯器，在其中您就會看到剛才兩個按鈕連結的 TakePhotoFromCameraAction 和 TakePhotoFromLibraryAction 行動對象。接著請為這兩個行動物件在物件檢視器中建立 OnDidFinishTaking 事件處理函式。OnDidFinishTaking 事件處理函式會在這行動物件執行完畢時呼叫，例如 TakePhotoFromCameraAction

會在使用者使用完照相機服務之後呼叫它的 OnDidFinishTaking 事件處理函式，如下所示:





再於主表單 TTabControl 元件的第 1 個頁次和第 5 個頁次中分別放入 2 個 TImage 組件:Image1 和 Image2。

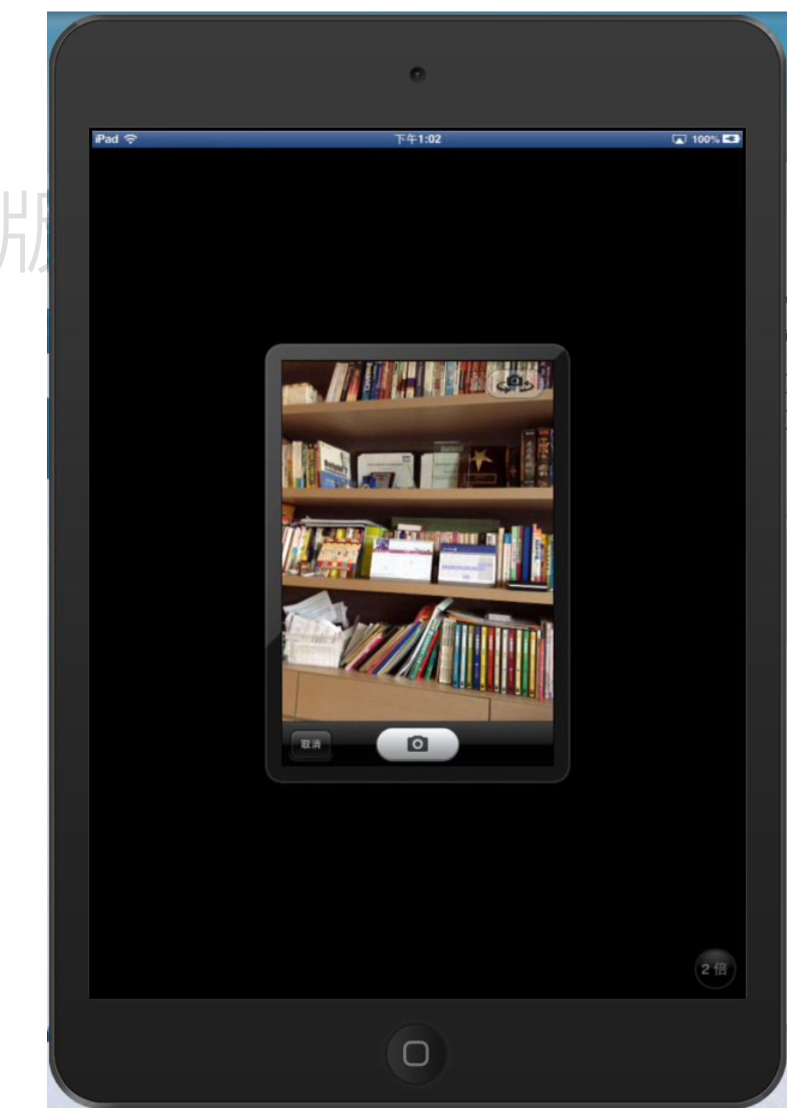
最後在這兩個 OnDidFinishTaking 事件處理函式中撰寫如下的程式碼:

```
procedure TfmMainForm.TakePhotoFromCameraAction1DidFinishTaking(Image:
TBitmap);
begin
```

```
  Image2.Bitmap.Assign(Image);
end;

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

OnDidFinishTaking 事件處理函式提供一個 TImage 參數，這個參數就是使用者使用照相機服務照的影像，或是從像片庫中選擇的影像。在這兩個事件處理函式中分別把影像參數指定給主表格中的 TImage 元件。

現在再請您編譯和執行此範例 App，在下面的畫面中您就可以看到我們在 iPad Mini 中使用 iOS 的照相服務以及從像片庫中選擇使用影像的服務了。



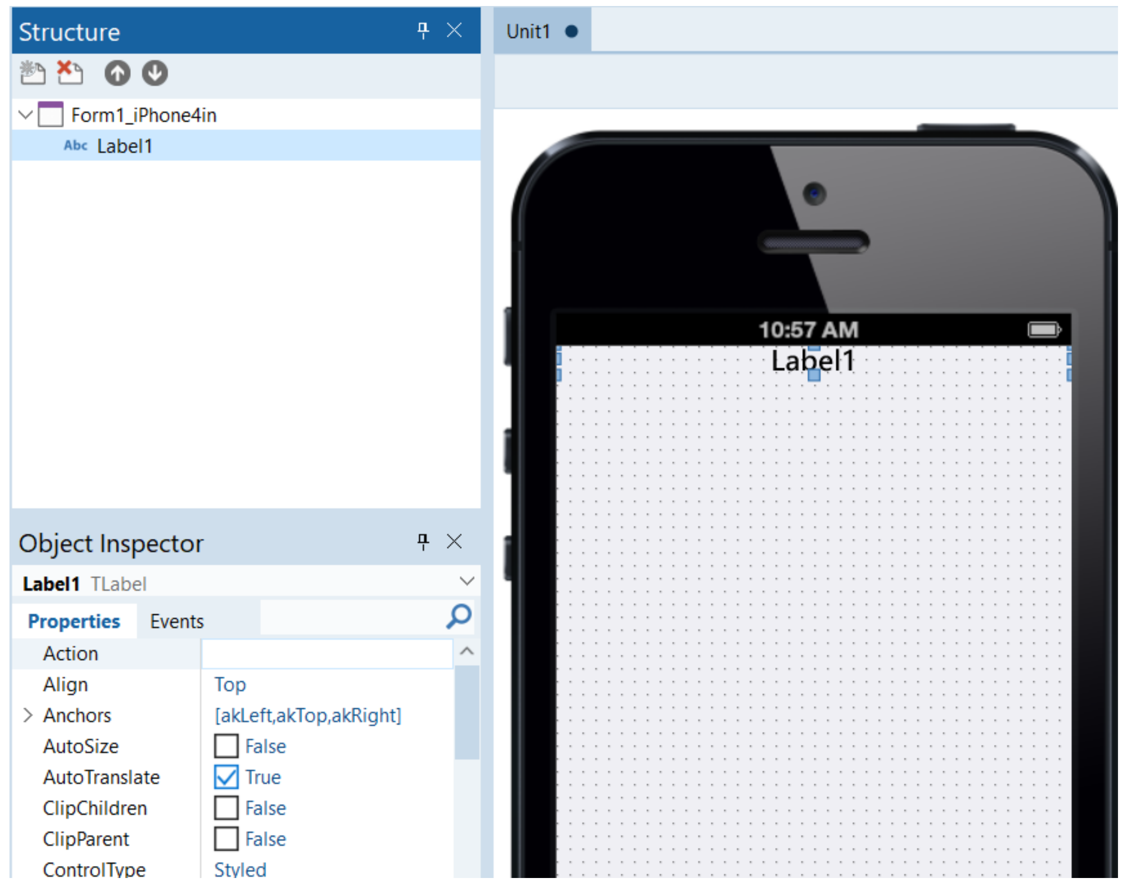
OnDidFinishTaking 事件處理函式提供一個 TImage 參數，這個參數就是使用者使用照相機服務照的影像，或是從像片函式庫中選擇的影像。在這兩個事件處理函式中分別把影像參數指定給主表格中的 TImage 元件。

10 分發額外檔案的 iOS App

最後讓我們討論如果您的 iOS App 需要分發您自己的檔案的話要怎麼處理。請回到 Delphi for iOS IDE，建立一個新的 iOS 項目，在這個 iOS 專案中

我們需要分發一個 INI 檔案，假設在這個 INI 檔案中存有一些初始化的資料需要在 iOS App 執行時先載入並且顯示出來。

在專案的主表單中加入一個 ToolBar 元件並且在其中再加入一個 TLabel 元件，最後再加入一個 TListBox 元件，如下所示:



此範例專案中包含一個 config.ini 檔，其中擁有如下的內容:

```
[General]
Name=Delphi for iOS 入門指引手冊
S1=安裝和設定 Delphi for iOS
S2=進入 Delphi for iOS 整合發展環境
S3=開發您的第 1 個 iOS App
S4=使用 Delphi for iOS 整合發展環境
S5=除錯您的 iOS App
S6=為您的 iOS App 進行單元測試
S7=開發和分發 iOS App 到 iOS 設備中
S8=分發複雜的 iOS App
S9=使用 iOS 的服務
```

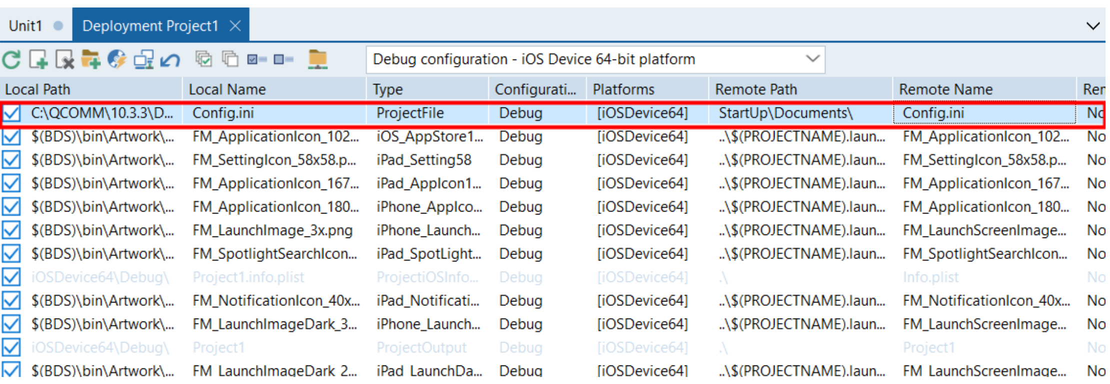
```
S10=分發額外檔案的 iOS App
```

接著在主表單的 OnCreate 事件處理函式中讀入 INI 檔案並且把讀入的資料顯示在 TListBox 中:

```
001    procedure TfmMain.AddToListBox(const sData: String);
002    var
003      lbi : TListBoxItem;
004    begin
005      lbi := TListBoxItem.Create(ListBox1);
006      lbi.Text := sData;
007      ListBox1.AddObject(lbi);
008    end;
009
010    procedure TfmMain.DisplayIniContent;
011    var
012      ConfigFile : TINIFile;
013      sData : String;
014      iIndex : Integer;
015      sName : String;
016    begin
017      if TFile.Exists(ConfigFilePath) then
018      begin
019        ConfigFile := TINIFile.Create(ConfigFilePath);
020        Label1.Text := Format('%s', [ConfigFile.ReadString('General',
'Name', '無法讀取')]);
021
022        for iIndex := 1 to 10 do
023        begin
024          sName := 'S' + IntToStr(iIndex);
025          sData := Format('%s', [ConfigFile.ReadString('General',
sName, '無法讀取')]);
026          AddToListBox(sData);
027        end;
028      end;
029    end;
030
031    procedure TfmMain.FormCreate(Sender: TObject);
```

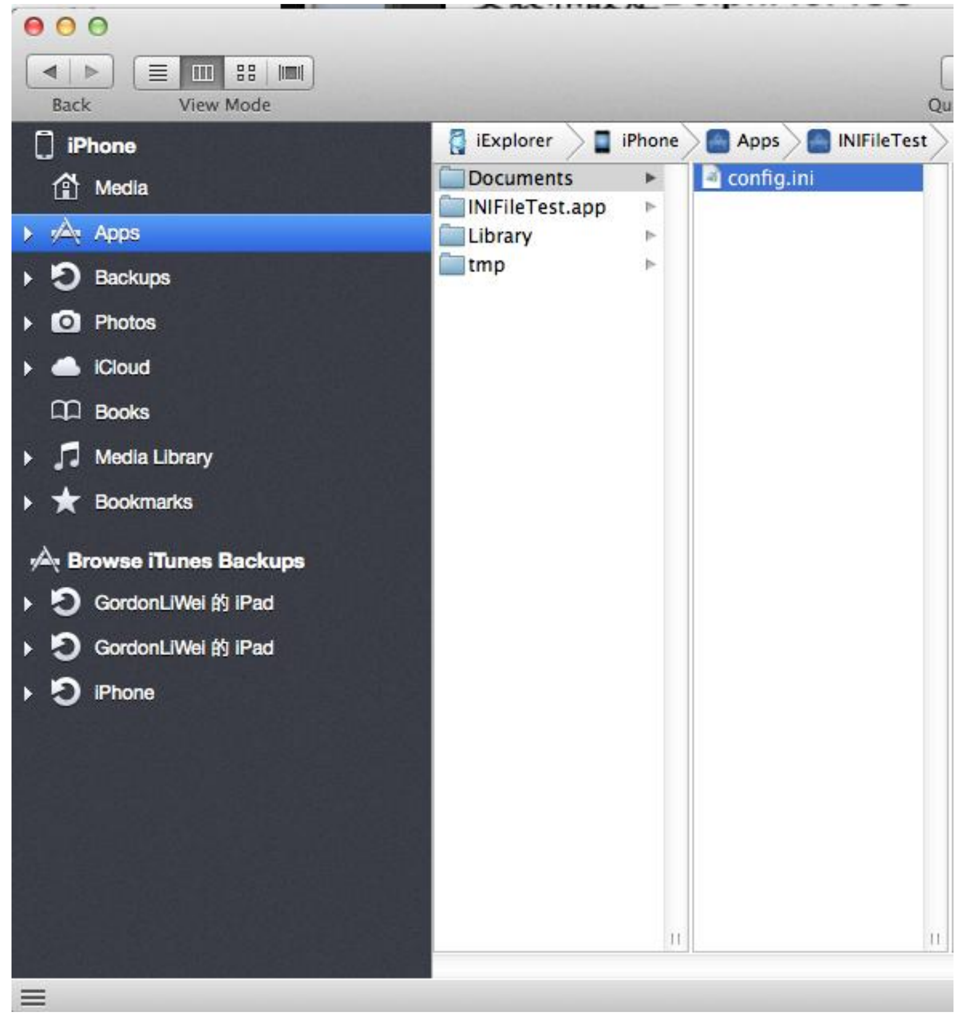
```
032    begin
033      ReadIniFile;
034      DisplayIniContent;
035    end;
036
037    procedure TfmMain.ReadIniFile;
038    begin
039      ConfigFilePath := TPath.GetDocumentsPath + PathDelim +
'config.ini';
040      if not TFile.Exists(ConfigFilePath) then
041        Label1.Text := Format('無法找到 : %s', [ConfigFilePath]);
042    end;
```

在執行此範例 iOS App 之前我們需要分發 config.ini 檔案，因此請點選 Project|Deployment 主選單啟動分發精靈，再點選左上方的『Add Files』按鈕加入專案中的 config.ini 檔，使用滑鼠按兩下它的『Remote Path』欄位，在其中輸入『StartUp\Documents』，如下所示即可:



現在請編譯並且執行此範例 iOS App，讀者可以在下圖看到此範例 iOS App 成功執行在 iOS Simulator 和真正的 iPhone 5 手機中了:

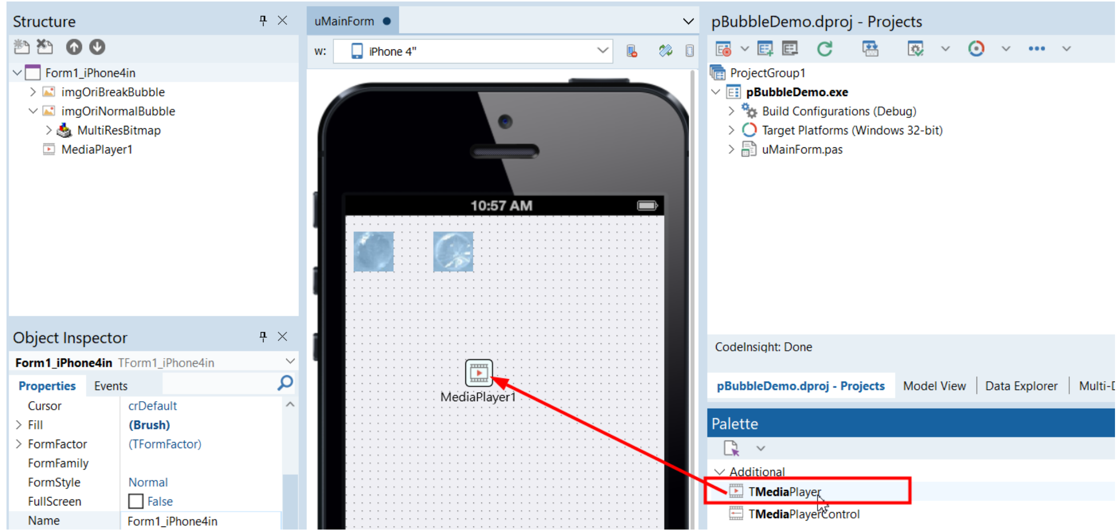
檢查此範例 iOS App 的 Docuemnts 目錄也可以看到下面的結果，config.ini 檔案果然成功的分發到 iPhone 5 手機的範例 iOS App 沙箱的 Documents 目錄中了，如下所示:



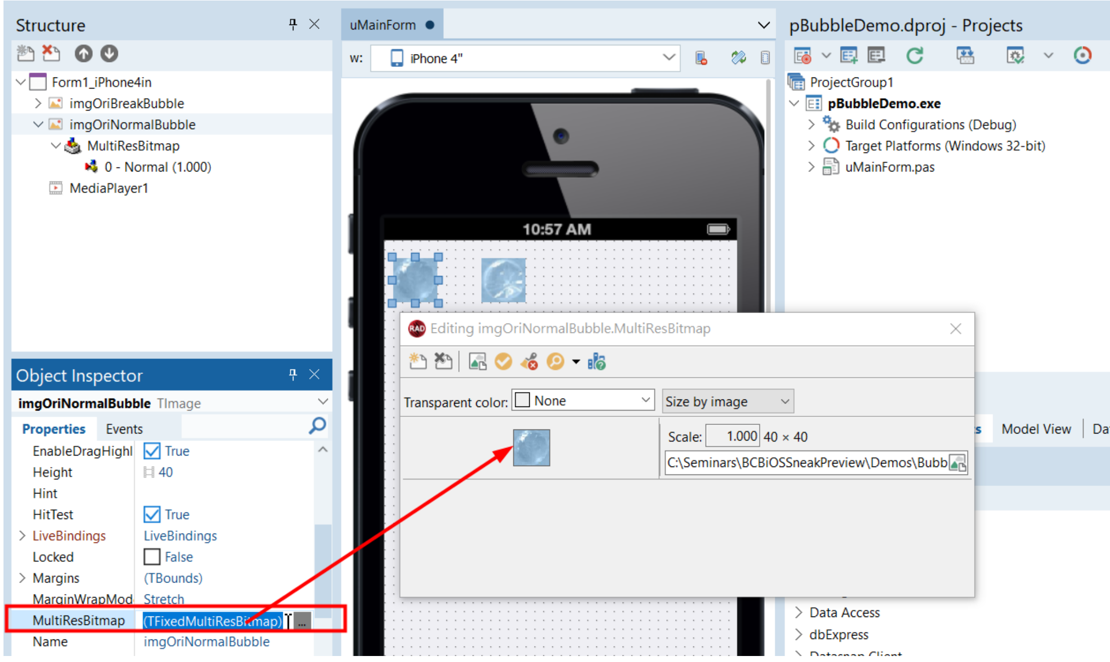
11 用 Delphi RIO 寫個遊戲吧

Delphi 在移動平臺上的原生 LLVM 編譯器的長處之一就是快速的執行速度，為了展示 Delphi RIO 開發 iOS App 的高生產力，讓我們來寫個 iOS App 的小遊戲吧，這個小遊戲就是小時候捏泡泡的遊戲，在這個過程中我們也會討論許多使用 Delphi RIO 開發 iOS App 的技巧。

首先在 Delphi IDE 中建立一個 FireMonkey Mobile Application 專案，先在主表單中放入一個 TLayout 元件並且設定它的 Align 特性值為 alClient，接著在主表格的 TLayout 元件中放入一個 TMediaPlayer 元件，二個 TImge 元件如下所示:



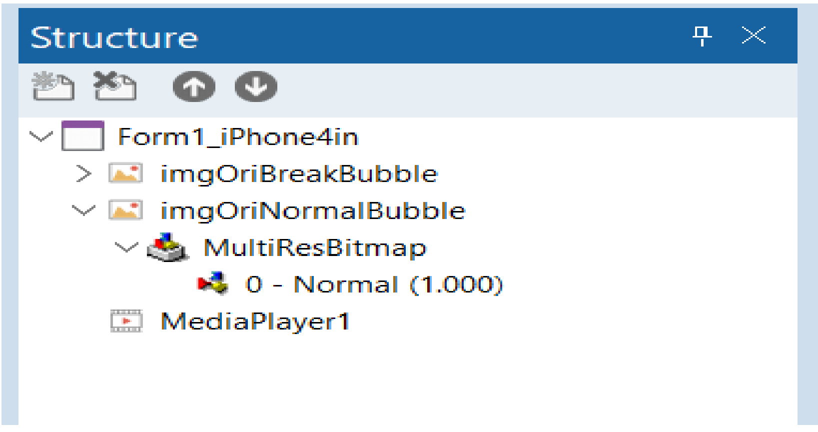
並且在物件檢視器中點選 TImage 元件的 MultiResBitmap 特性載入 2 個泡泡圖像，如下所示：



並且特定 2 個 TImage 組件的 Name 特性值如下:

Name特性值	說明
imgOriNormalBubble	顯示正常泡泡的圖像
imgOriBreakBubble	顯示捏破泡泡的圖像

現在主表單中所有先件之間的關係如下圖顯示在 IDE 左上方的架構視窗中:



11-1 讓泡泡充滿畫面吧

在主表單中的 imgOriNormalBubble 和 imgOriBreakBubble 只是做為顯示正常泡泡和捏破泡泡的母圖像，這個小遊戲首先將使用 imgOriNormalBubble 畫滿整個畫面，之後當玩者點選了畫面中任一正常泡泡之後就代表要捏破這個泡泡，那麼我們就需要把這個正常泡泡圖像改成顯示捏破泡泡的圖像並且播放一捏破泡泡的聲音檔。

第一個工作就是在主表單的 OnActivate 事件處理及式中呼叫 SetupBubbleSound() 方法設定捏破泡泡音效檔的位置，接著呼叫 SetupBubbles() 方法在主表單中繪滿正常泡泡的圖像，最後我們把 imgOriNormalBubble 和 imgOriBreakBubble 隱藏起來：

版權所有 請勿翻印

```
procedure TfmMainForm.FormActivate(Sender: TObject);
begin
  SetupBubbleSound();
  SetupBubbles();
  imgOriNormalBubble.Visible := false;
  imgOriBreakBubble.Visible := false;
end;
```

SetupBubbles()方法的工作就是在 011 行呼叫 imgOriNormalBubble 元件的 Clone()方法拷貝正常泡泡圖像的物件，013 行設定拷貝正常泡泡圖像物件的 Parent 是 Layout1，014/015 行設定這個拷貝正常泡泡圖像物件的正確位置，016 行設定它的 OnClick 事件處理及式以便在被點選時切換成被捏破的圖像，017 行把拷貝的正常泡泡圖像物件暫存在 pBubbles 陣列中，最後在代表這個拷貝正常泡泡圖像物件是否已被捏破的 BubblePoppedStatus 布林值陣列中設定為 false 以代表目前為正常的狀態。

```
001    procedure TfmMainForm.SetupBubbles();
002    var
```

```
003      iRow, iCol : Integer;
004      pCloneImage : TImage;
005    begin
006      if (not bSetup) then
007      begin
008        for iRow := 0 to IROWS - 1 do
009        begin
010          for iCol := 0 to ICOLS - 1 do
011          begin
012            pCloneImage := TImage(imgOriNormalBubble.Clone(Self));
013            pCloneImage.Parent := Layout1;
014            pCloneImage.Position.X := iCol * 40;
015            pCloneImage.Position.Y := iRow * 40;
016            pCloneImage.OnClick := OnBubbleClick;
017            pBubbles[iRow,iCol] := pCloneImage;
018            BubblePoppedStatus[iRow,iCol] := false;
019          end;
020        end;
021        bSetup := true;
022      end;
023    end;
```

而 BubblePoppedStatus 和 pBubbles 則是宣告在類別的 **private** 中：

```
BubblePoppedStatus : array[0..IROWS - 1, 0..ICOLS - 1] of Boolean;
pBubbles : array[0..IROWS - 1, 0..ICOLS - 1] of TImage;
```

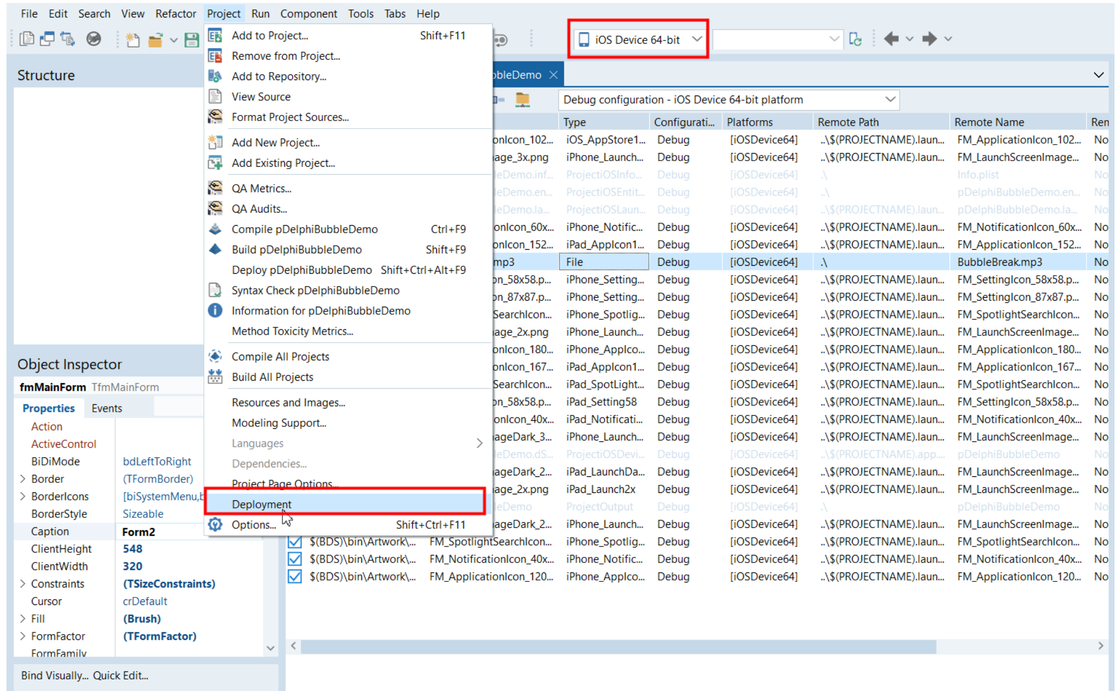
SetupBubbleSound()方法的工作則是設定捏破泡泡時要播放的音效檔位置，稍後我們會說明如何使用 IDE 部署聲音檔，因此 SetupBubbleSound()方法就是設定主表單中 TMediaPlayer 元件載入音效檔正確的路徑。由於部署 iOS App 時只能把 App 需要使用的其他檔案部署在 App 的沙箱中而不能隨意部署 App 要使用的額外檔案，因此對於像本範例使用的音效檔，我們應該把它部署在 App 的 Documents 目錄中。

因此在 SetupBubbleSound()方法的 003 行我們可呼叫 GetHomePath()方法取得 iOS App 本身的根目錄，接著再於它的 Documents 目錄下載入音效檔"BubbleBreak.mp3"，006 行判斷如果聲音檔"BubbleBreak.mp3"存在的話就設定 TMediaPlayer 組件的 FileName 特性值為此音效檔，否則就顯示一警告訊息：

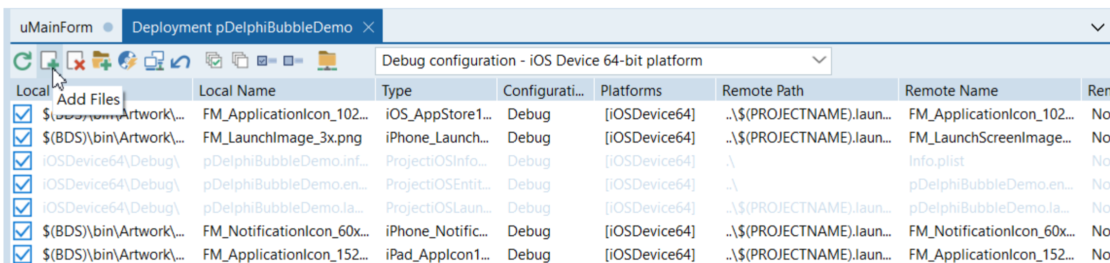
```
001    procedure TfmMainForm.SetupBubbleSound;
```

```
002    var
003      sFileName : String;
004    begin
005      sFileName := TPath.GetDocumentsPath() + PathDelim +
'BubbleBreak.mp3';
006      if (FileExists(sFileName)) then
007        MediaPlayer1.FileName := sFileName
008      else
009        ShowMessage(sFileName + '檔案不存在!');
010    end;
```

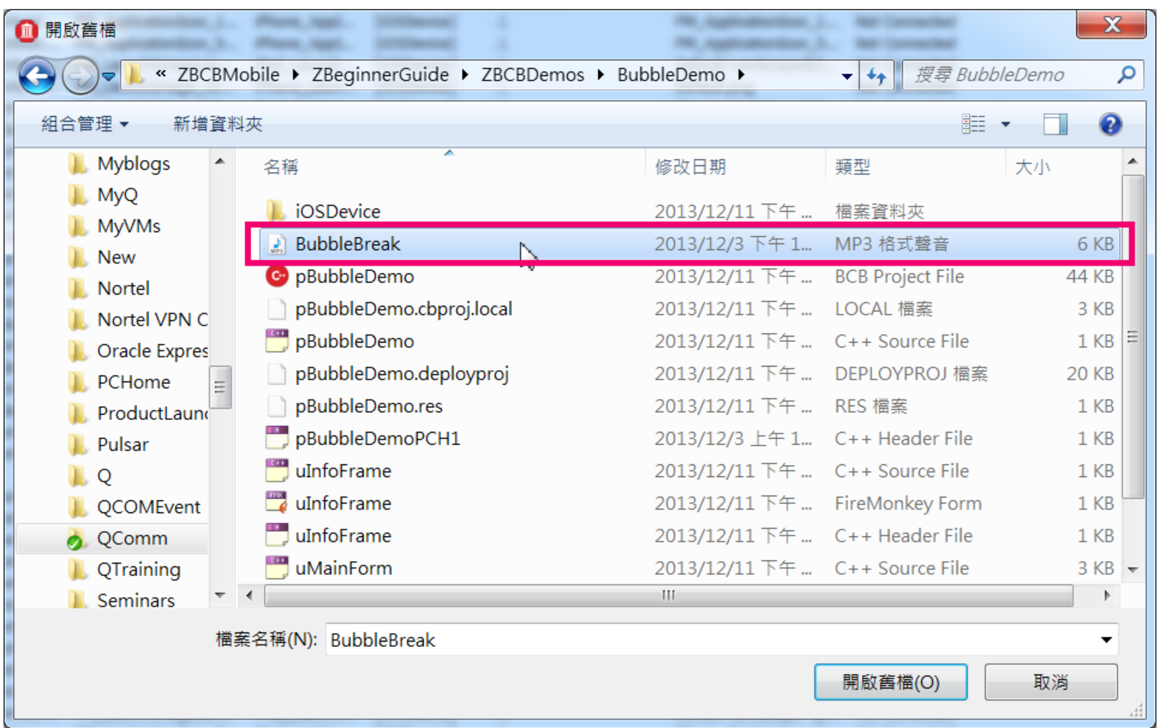
好了，到這裡此範例 App 就可以先執行看看了，但在執行之前我們需要連同聲音檔"BubbleBreak.mp3"一起部署到 iPhone 5 手機中。請在 IDE 中點選 Project | Deployment 選單，IDE 會啟動部署精靈，如下所示：



在部署精靈中點選左方上的『Add Files』按鈕以便加入聲音檔"BubbleBreak.mp3"：

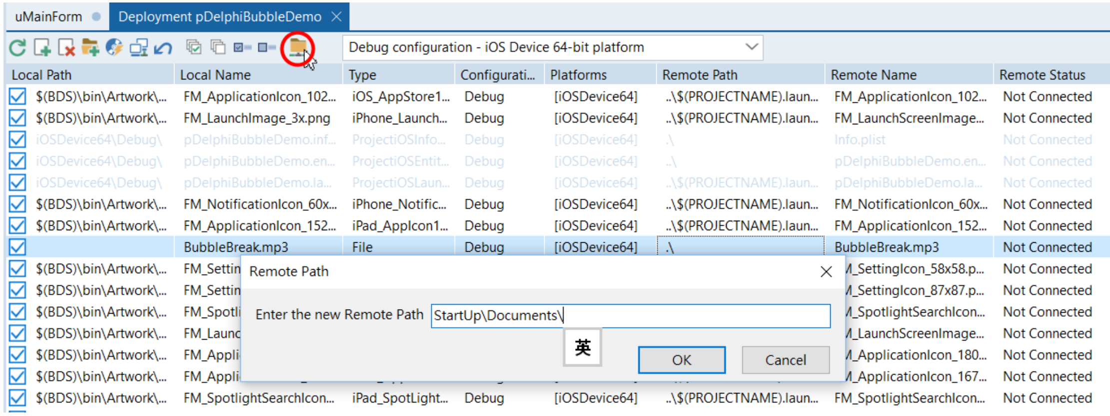


部署精靈此時會顯示開啟舊檔對話盒讓您載入聲音檔：



加入了聲音檔"BubbleBreak.mp3"之後請點選部署精靈的『Change Remote Path For Selected Items』按鈕 把音效檔的遠端部署位置設定為『StartUp\Documents\』，如下所示，請注意由於 iOS 平臺會區分大小寫，因此您必須依照大小寫確實輸入 StartUp\Documents\：

版權所有 請勿翻印



現在請確定您的 iPhone 或是 iPad/iPad Mini 手機已經藉由 USB 連結到 Mac 機器上，那麼請在 IDE 中編譯和執行此範例 App，之後就應該可以在 iPhone 手機中看到如下成功執行的畫面了：

版權 印

11-2 點選捏破泡泡

當使用者點選主表單中的正常泡泡圖像時就會執行前面設定的 OnBubbleClick 事件處理及式，在 OnBubbleClick 中我們需要把正常泡泡圖像改成捏破泡泡的圖像並且播放前面部署的聲音檔"BubbleBreak.mp3"。因此在下面的 OnBubbleClick 事件處理中在 003/004 行我們先算出是那一個正常泡泡圖像被點選，009 行把音效檔的播放位置重置回開始的位置，010 行判斷如果目前被點選的泡泡圖像是正常泡泡圖像的話就把目前傳入的 Sender 參數的型態轉換成 TImage 型態，再從 imgOriBreakBubble 中載入捏破泡泡的圖像，最後再 014 行藉由 TMediaPlayer 組件播放音效檔"BubbleBreak.mp3"：

```
001    procedure TfmMainForm.OnBubbleClick(Sender: TObject);
002    var
003      iRow, iCol : Integer;
004    begin
005      iRow := Trunc( (Sender as TImage).Position.Y / BUBBLESIZE);
006      iCol := Trunc( (Sender  as TImage).Position.X / BUBBLESIZE);
```

```
007
008     MediaPlayer1.Stop();
009     MediaPlayer1.CurrentTime := 0;
010     if (not BubblePoppedStatus[iRow, iCol]) then
011     begin
012       BubblePoppedStatus[iRow, iCol] := true;
013       (Sender  as
TImage).MultiResBitmap.Assign(imgOriBreakBubble.MultiResBitmap);
014       MediaPlayer1.Play();
015     end;
016   end;
```

再次編譯和執行此範例 App 並且隨意點選泡泡就可以看到如下的畫面被點選的泡泡就會顯示被捏破並且會聽到泡泡被捏破的聲音了。

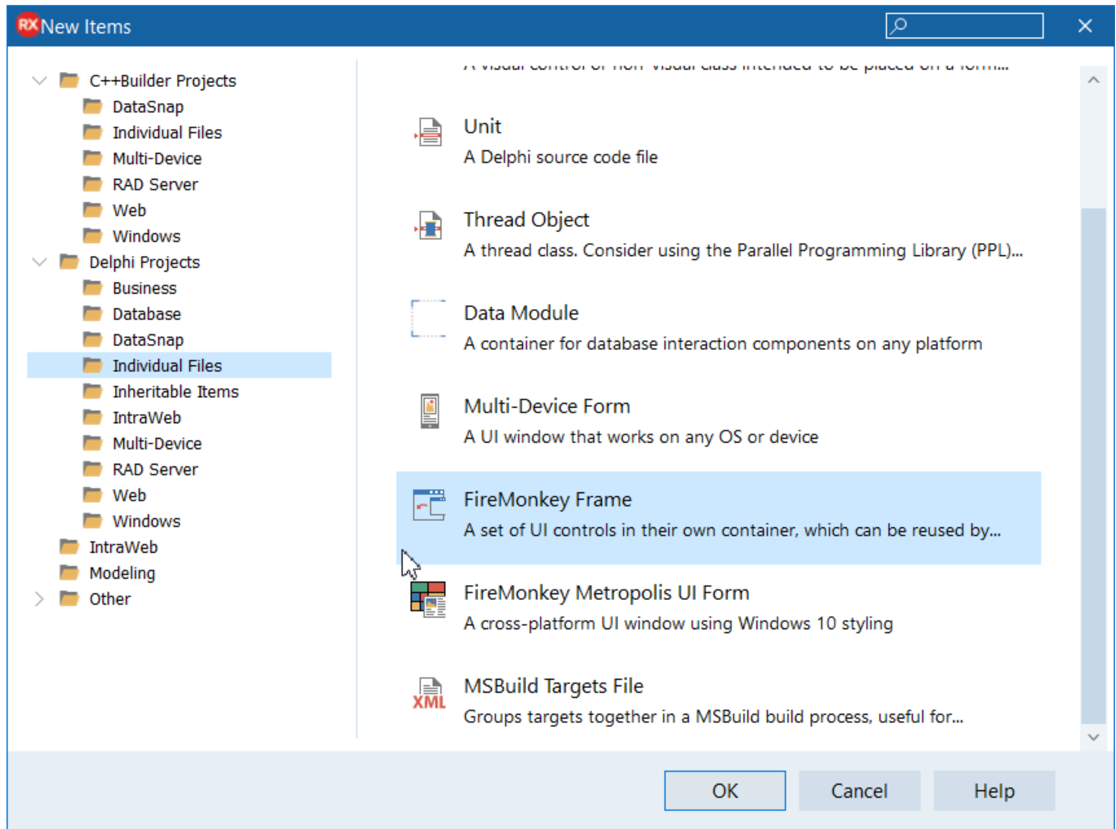
版權

印

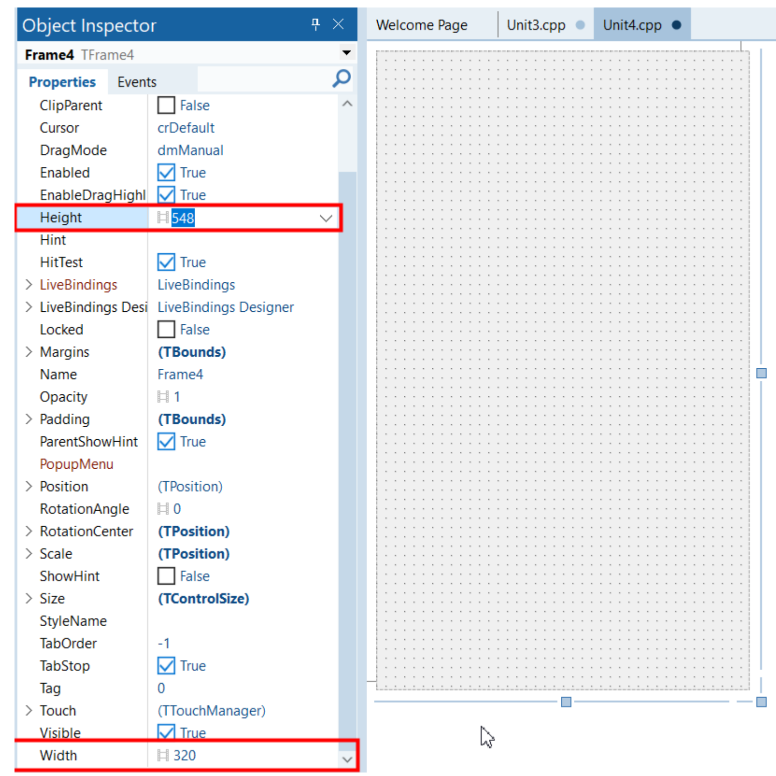
11-3 加入手勢功能

接下來讓我們在這個範例 App 中加入使用手勢的功能，當玩家在主表單中使用手勢向左方滑動時，讓我們藉由 FireMonkey 的動畫功能動態的顯示一個此遊戲的關於資訊，同時讓我們說明如何使用 FireMonkey 的 Frame 功能來顯示關於資訊。

現在請在 IDE 中點選 New Items 快速鍵於 New Items 對話盒中 Delphi Files 專案中選擇建立 FireMonkey Frame 物件，如下圖所示：

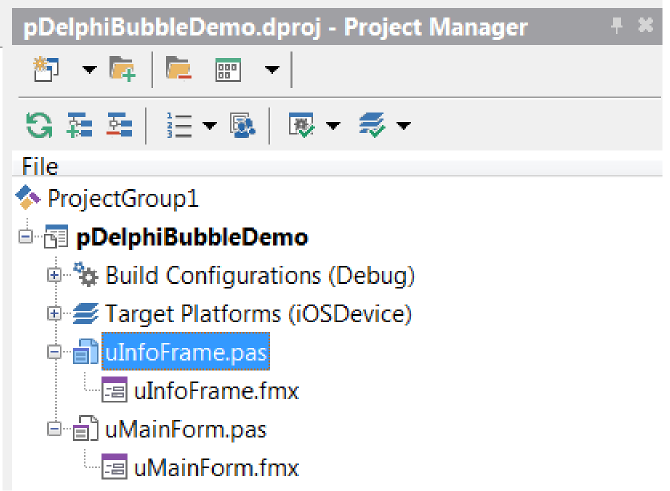


點選建立 FireMonkey Frame 物件之後 IDE 便會建立一個空白的 Frame 表單，請在物件檢視器中設定它的 Height 和 Width 特性值和主表單中 Layout 元件一樣的 Height 和 Width 特性值，如下所示：

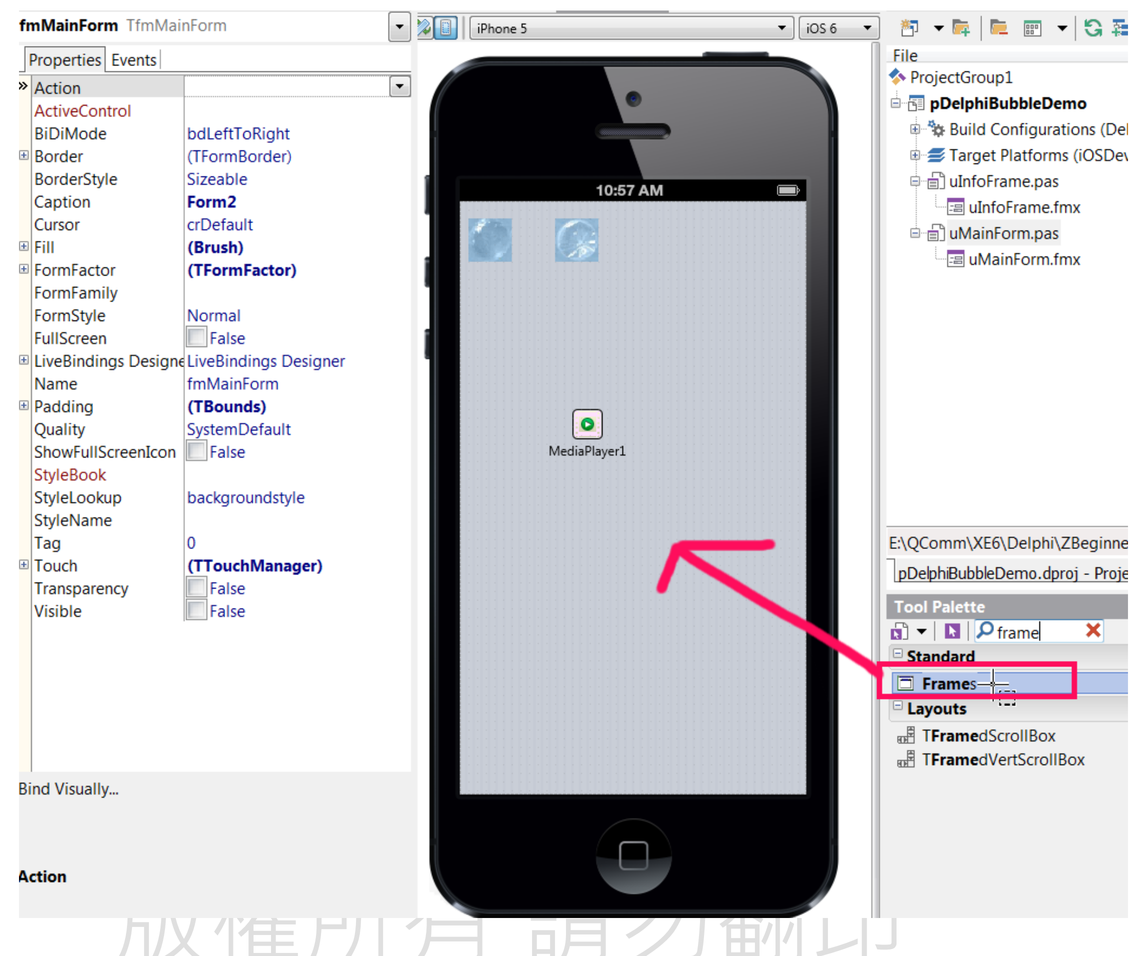


最後設定它的 Name 特性值為 frmGameInfo 並且以 uInfoFrame 儲存這個 FireMonkey Frame 物件，此時專案管理員的內容應如下所示：

版權所有 請勿翻印

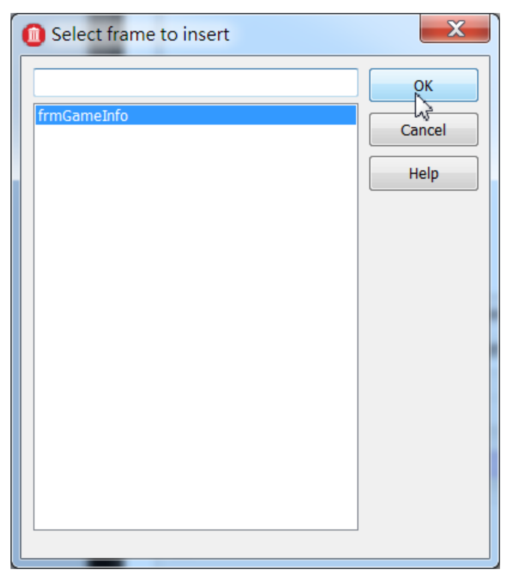


接著回到專案中的主表單，在工具盤中找到 Frame 元件並且把它拖曳到主表單中如下所示：

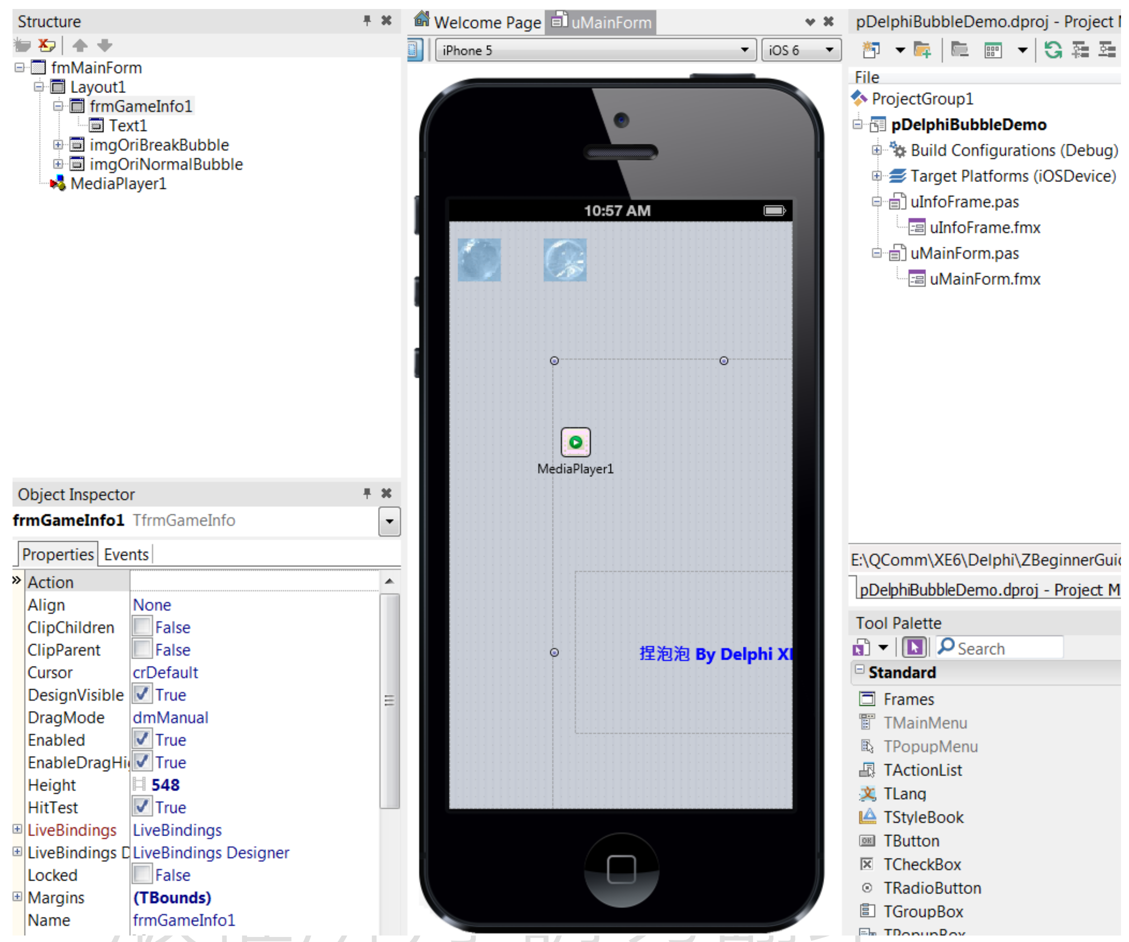


版權所有 請勿翻印

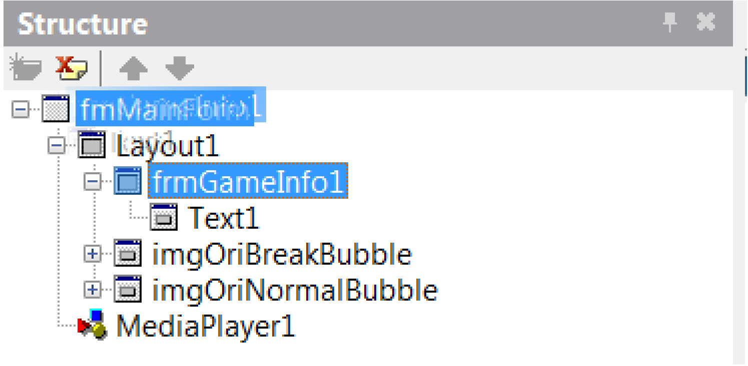
當您拖曳 Frame 元件到主表單後 IDE 便會顯示如下的對話盒詢問您這個 Frame 元件要使用的 FireMonkey Frame 物件是什麼，由於在前面我們已經建立了 frmGameInfo，因此就請在對話盒中選擇 frmGameInfo，如下所示：



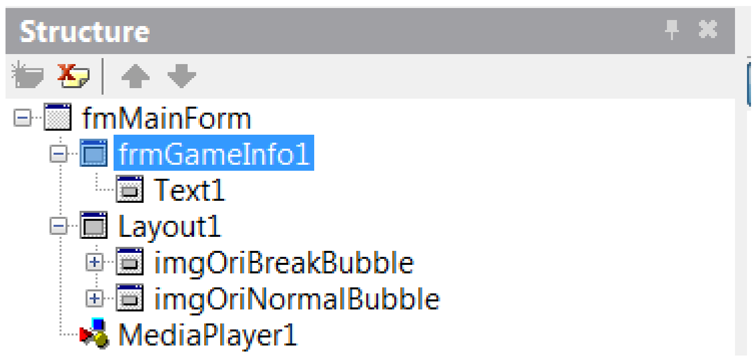
在上面的對話盒中選擇了 frmGameInfo 之後我們就可以在主表單中看到 frmGameInfo 也顯示在主表單中了，如下所示：



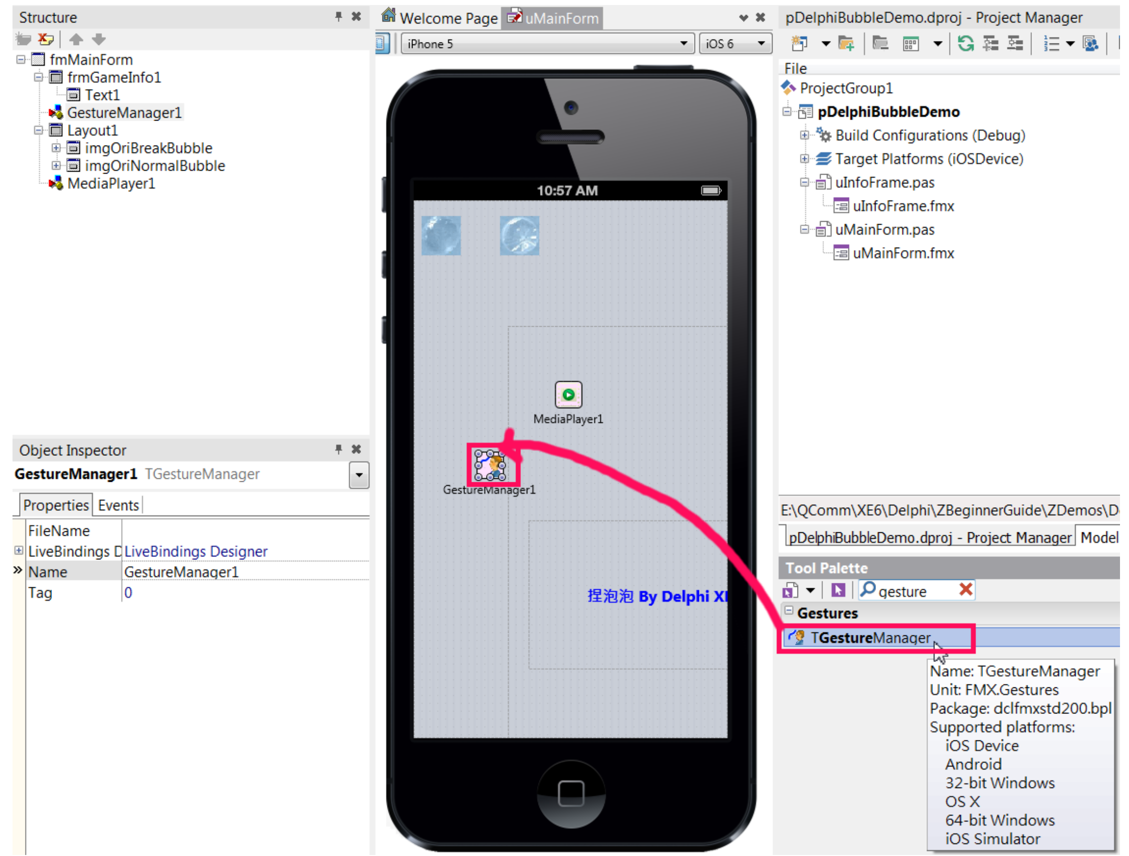
現在請在 IDE 左上方的架構視窗中查看 frmGameInfo 的父代元件是什麼，如果 frmGameInfo 是在 Layout1 元件之下就代表 Layout1 是它的父代組件，那麼就請在架構視窗中點選 frmGameInfo，在保持按著滑鼠左鍵的狀況下拖曳 frmGameInfo 到 MainForm 之下再釋放滑鼠左鍵讓 MainForm 成為 frmGameInfo 的父代組件，如下所示：



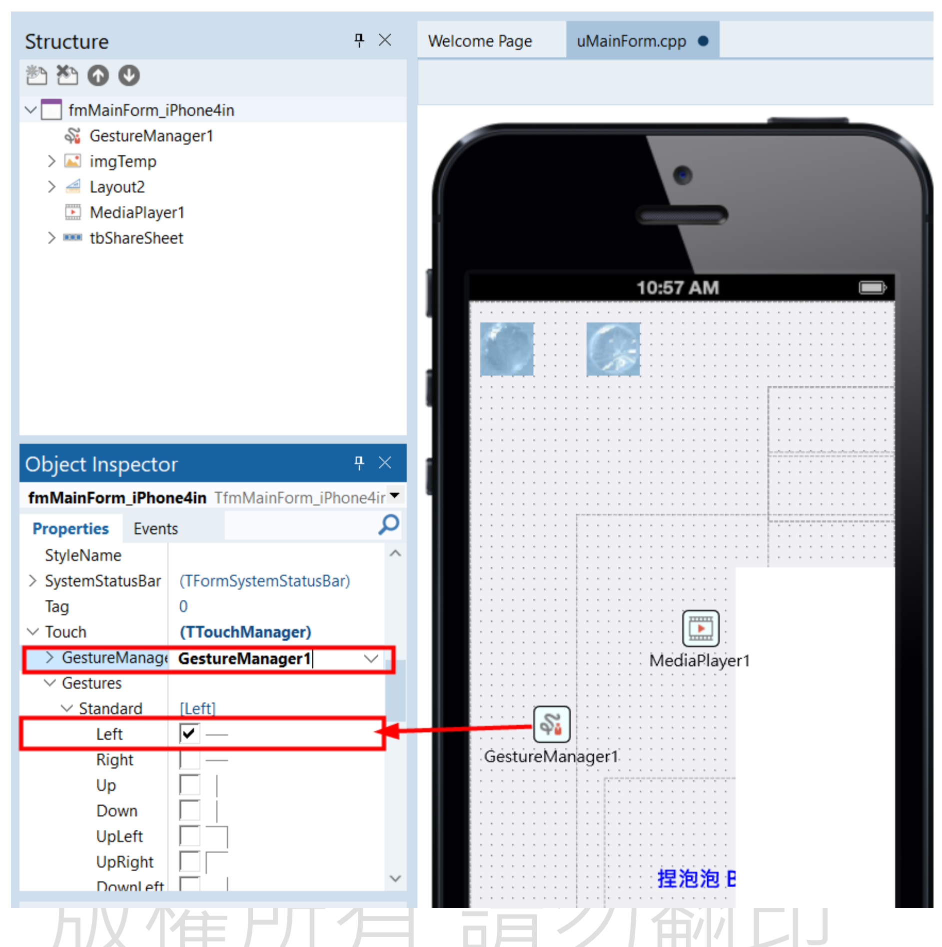
拖曳完成之後架構視窗應該如下所示：



好了，接下來再於主表單中拖入 TGestureManager 元件準備加入處理手勢的功能，如下所示：

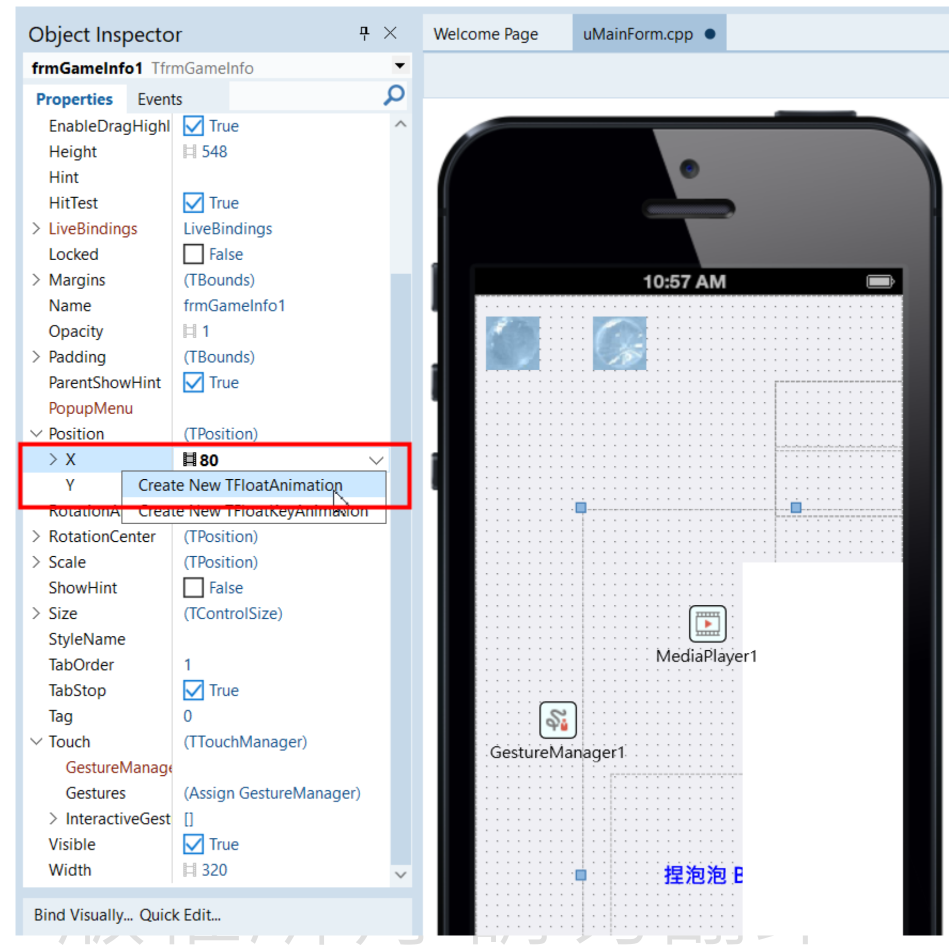


在物件檢視器中設定 MainForm 的 GestureManager 特性值為剛才拖入的 TGestureManager 組件，展開 MainForm 的 Gestures 特性在它的 Standard 子特性中勾選 Left 子特性代表主表單 MainForm 將處理向左滑動的手勢，如下所示：

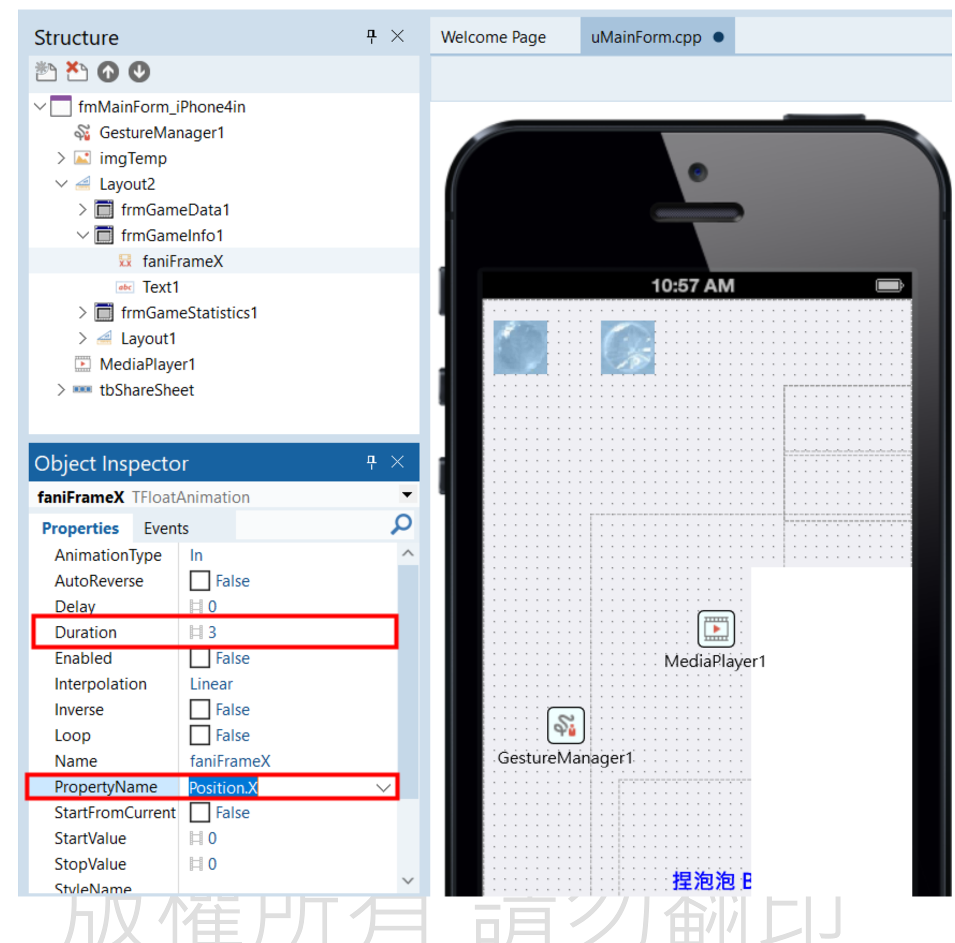


版權所有 請勿翻印

為了讓 frmGameInfo 物件在玩家向左滑動手勢後動態的顯示出來，讓我們為 frmGameInfo 物件的 X 軸特性加入一個動畫的功能。請點選主表單中的 frmGameInfo，在物件檢視器中點選它的 Position 特性下的 X 子特性，在 X 子特性的下拉盒中選擇建立一個 TFloatAnimation 物件，如下所示：



在 IDE 左上方的架構窗口中點選新加入的 FloatAnimation1 物件，再於物件檢視器中設定它的 Duration 特性為 3 代表這個動畫將持續 3 秒的時間，如下所示：



版權所有 請勿翻印

再把此此 FloatAnimation1 物件的 Name 特性值更改為 faniFrameX。

現在就可以撰寫處理向左滑動的手勢了，點選主表單的 MainForm 物件在物件檢視器中建立它的 OnGesture 事件處理函式，並且在其中撰寫如下的程式碼：

```
001   procedure TfmMainForm.FormGesture(Sender: TObject;
002     const EventInfo: TGestureEventInfo; var Handled: Boolean);
003   begin
004     if (EventInfo.GestureID = sgiLeft ) then
005     begin
006       faniFrameX.Enabled := false;
007       faniFrameX.StartValue := Layout1.Width;
008       faniFrameX.StopValue := 0;
009       faniFrameX.Enabled := true;
010       Handled := true;
011     end;
012   end;
```

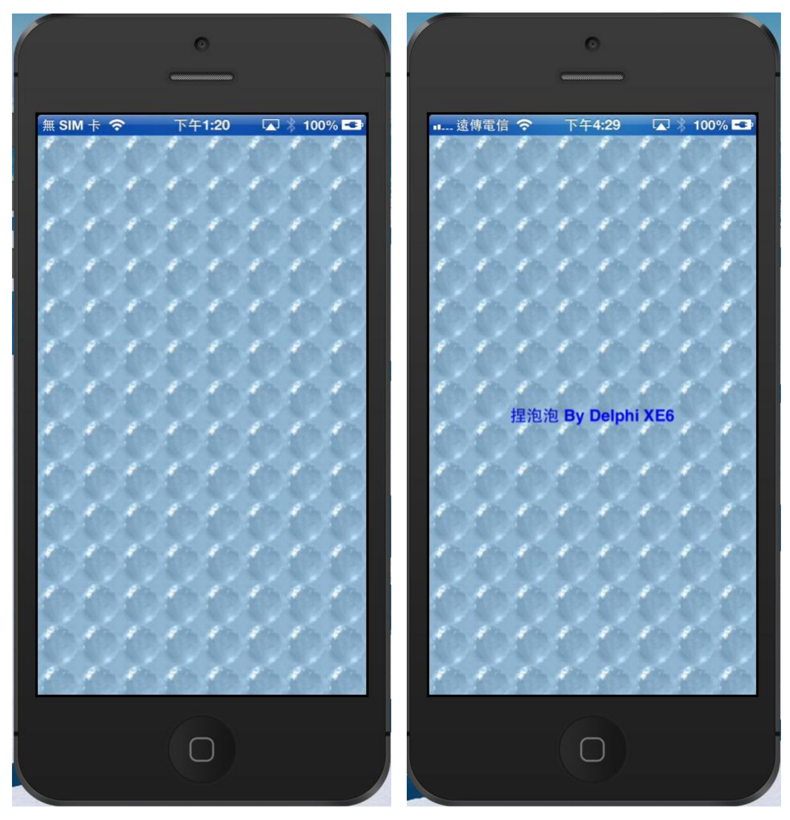
當玩家在手機螢幕做出向左滑動的手勢後 TGestureManager 就會觸發 MainForm 的 OnGesture 事件處理函式。OnGesture 事件處理函式的第 2 個參數 TGestureEventInfo 包含此手勢的相關資訊，在 TGestureEventInfo 中的 GestureID 變數代表使用者做出的手勢種類，因此我們只要判斷 GestureID 就可以知道玩家是不是做出了左向滑動的手勢。

FireMonkey 框架以 sgiLeft 常數代表左向滑動的手勢，因此在上面的 004 行判斷玩家做出的手勢是不是 sgiLeft，如果是的話就設定 faniFrameX 動畫物件開始執行的位置在 X 軸 Lauout1 的寬度處，也就是手機螢幕的最右方，向手機螢幕的最左方向出現，設定好動畫的方向和位置之後 009 行就啟動 faniFrameX 動畫物件開始執行。

完成了 OnGesture 事件處理函式之後我們需要在此 App 啟動時先把 frmGameInfo 物件設定在螢幕的最右方，如此一來 faniFrameX 動畫物件才能正確的工作。因此我們需要在 MainForm 的 OnActivate 事件處理函式中加入一行程式碼呼叫 SetupGameInfoFrame() 方法，而它的工作就是設定 frmGameInfo 物件的正確啟動位置：

```
procedure TfmMainForm.SetupGameInfoFrame;
begin
  frmGameInfo1.Position.X := Layout1.Width;
  frmGameInfo1.Position.Y := 0;
end;
```

現在編譯和執行此範例 App 並且隨意點選泡泡，再使用手指向手機螢幕的左方滑動，就可以看到 frmGameInfo 物件從右向左慢慢的滑動出現了，如下所示：

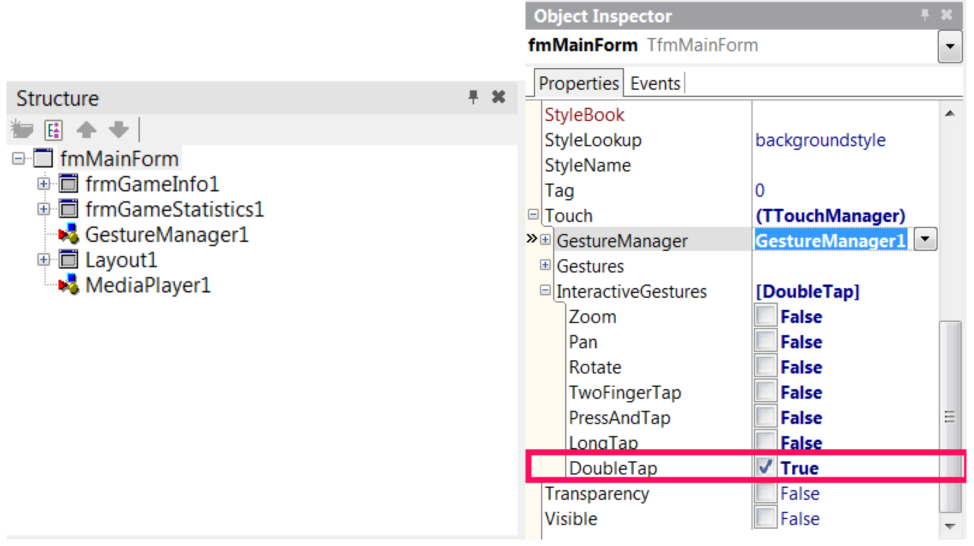


11-4 重新開始遊戲

版權所有 請勿翻印

現在再讓我們為這個小遊戲加入重新開始的功能，當玩家想重新開始時只需要在遊戲中輕點 2 次螢幕就可以讓遊戲回到起始的狀態，要如此做我們回需要再讓這個 App 支援 DoubleTap 手勢即可。

回到 MainForm 在它的 GestureManager|InteractiveGestures 特值中勾選 DoubleTap 選項代表 App 要處理使用者在 App 輕點 2 次螢幕的手勢：



接著在 MainForm 的 FormGesture 事件處理函式中加入如下處理 DoubleTap 手勢的程式碼：

```
    if (EventInfo.GestureID = igiDoubleTap ) then
    begin
      ResetGame();
    end;
```

而 ResetGame()方法只是呼叫 SetupGameInfoFrame()重新設定 2 個 TFrame 物件的位置，再把畫面中已經被捏破的泡泡恢復原狀而已：

```
procedure TfmMainForm.ResetGame();
var
  iRow, iCol : Integer;
begin
  SetupGameInfoFrame();
  for iRow := 0 to IROWS -1 do
  begin
    for iCol := 0 to ICOLS - 1 do
    begin
      if (BubblePoppedStatus[iRow, iCol]) then
      begin
        pBubbles[iRow,
iCol].MultiResBitmap.Assign(imgOriNormalBubble.MultiResBitmap);
        BubblePoppedStatus[iRow, iCol] := false;
      end;
    end;
  end;
end;
```

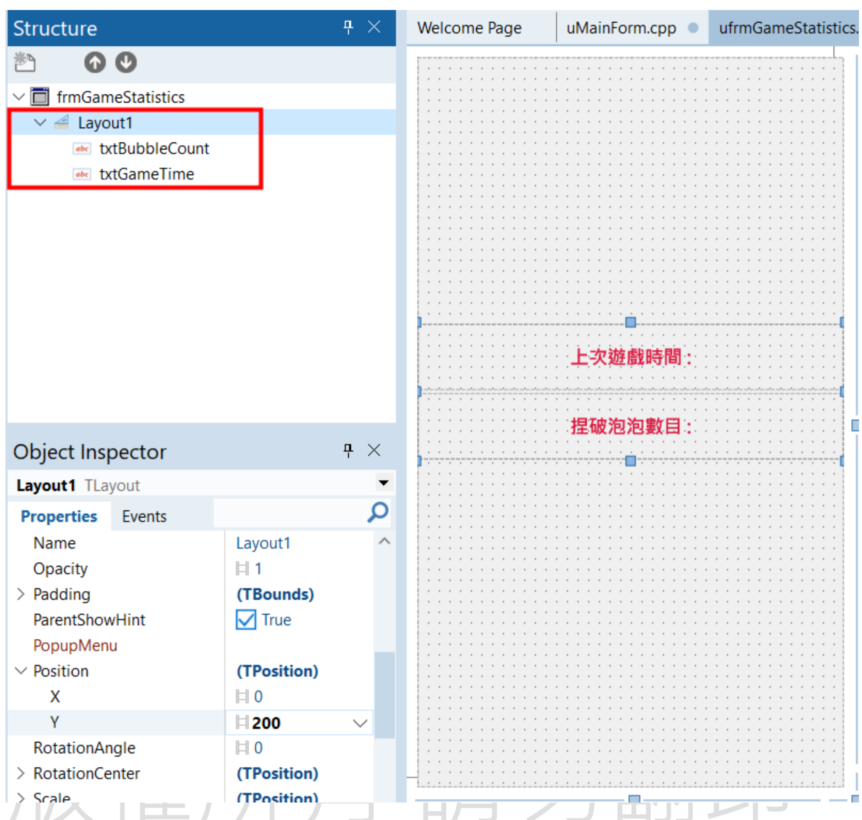
現在再次執行此範例 App，試著捏破一些泡泡再輕點 2 次螢幕，您會發現此範例 App 又恢復到了原始的執行狀態了。

11-5 處理遊戲資訊

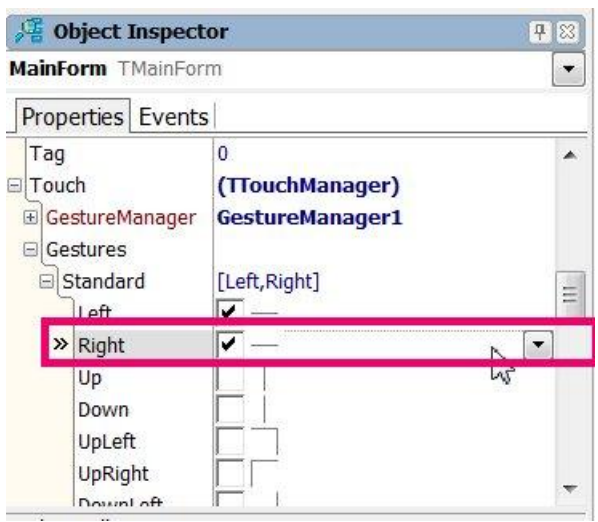
現在讓我們試著把玩家每次玩捏泡泡的資訊儲存起來，並且讓玩家能夠使用向右的手勢來顯示上次玩遊戲的資訊。

再次讓我們使用一個 Frame 物件來顯示遊戲的資訊，請使用前面介紹的方式再建立專案中第 2 個 FireMonkey Frame 物件，設定它的 Height 和 Width 特性和主表單中的 Layout1 一樣，再於其中放入一個 TLayout 元件再於其中放

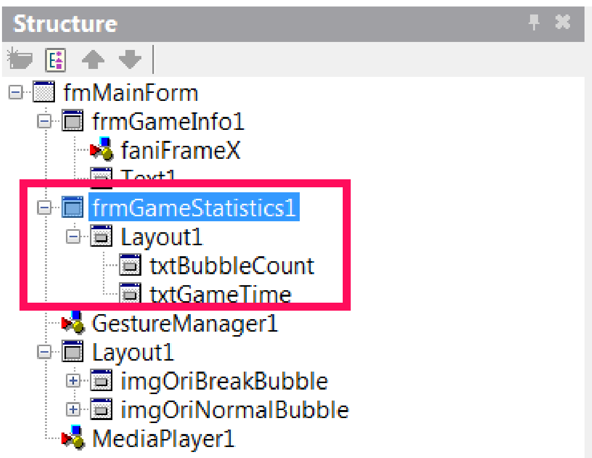
入 2 個 Text 物件並且設定 TLayout 元件的 Align 特性為 alCenter，設定 Name 特性為 frmGameStatistics，如下所示：



回到主表單讓主表單支援由左向右滑動的手勢：

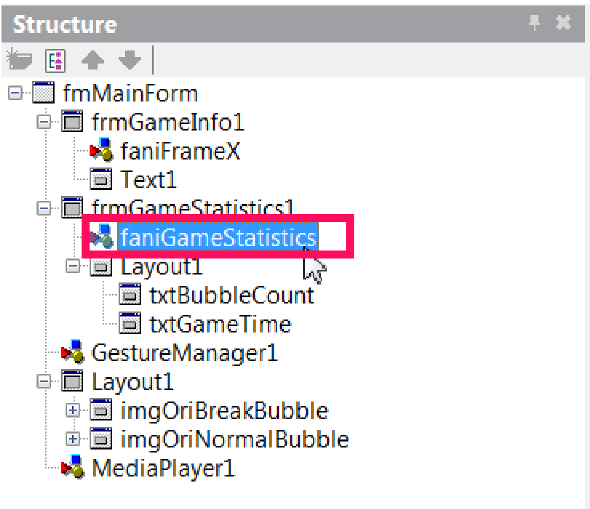


在 IDE 左上方的架構窗口中拖曳 frmGameStatistics 的父代組件為 MainForm，如下所示：



此時主表單就會出現第 2 個 Frame 物件 frmGameStatistics，如下所示：

接著如同前面為 frmGameInfo 物件的 X 軸特性加入一個動畫的功能，現在也請為 frmGameStatistics 物件的 X 軸特性加入一個動畫的功能物件 faniGameStatistics：



修改 SetupGameInfoFrame() 方法，把第 2 個 Frame 物件 frmGameStatistics1 的起始位置設定為手機螢幕最左方之外：

```
procedure TfmMainForm.SetupGameInfoFrame;
begin
  frmGameInfo1.Position.X := Layout1.Width;
  frmGameInfo1.Position.Y := 0;
  frmGameInfo1.Parent:= Self;

  frmGameStatistics1.Position.X := -Layout1.Width;
  frmGameStatistics1.Position.Y := 0;
  frmGameStatistics1.Parent := Self;
end;
```

WriteGameInfo()方法則是使用 TIniFile 物件把上次遊戲的時間以及上次玩家捏破了多少個泡泡的數目寫到名為"BubbleGameInfo.ini"的檔案中，請注意"BubbleGameInfo.ini"也必須儲存在這個 App 沙箱的 Documents 目錄下。

因此在下面的 013 行先取得遊戲資訊組態檔案的正確路徑之後 014 行便根據遊戲資訊組態檔案建立 TIniFile 物件，然後呼叫 TIniFile 物件的 WriteString() 方法把現在的時間和計算出來的捏破泡泡的數目寫入這個檔案中：

```
001    const sBubbleGameInfo = 'BubbleGameInfo.ini';
002
003    function TfmMainForm.GetGameInfoFile() : String;
004    begin
005      Result := TPath.GetDocumentsPath() + PathDelim +
sBubbleGameInfo;
```

```
006    end;
007
008    procedure TfmMainForm.WriteGameInfo();
009    var
010      sGameInfoFile : String;
011      pConfigFile : TIniFile;
012    begin
013      sGameInfoFile := GetGameInfoFile();
014      pConfigFile := TIniFile.Create(sGameInfoFile);
015      try
016        pConfigFile.WriteString('遊戲', '時間', DateTimeToStr(Now()));
017        pConfigFile.WriteString('遊戲', '捏破泡泡數目',
IntToStr(GetBrokenBubles())) ;
018      finally
019        pConfigFile.Free;
020      end;
021    end;
022
023    function TfmMainForm.GetBrokenBubles() : Integer;
024    var
025      iRow, iCol : Integer;
026    begin
027      Result := 0;
028
029      for iRow := 0 to IROWS - 1 do
030      begin
031      for iCol := 0 to ICOLS - 1 do
032        if (BubblePoppedStatus[iRow, iCol]) then
033           Inc(Result);
034      end;
035    end;
```

當然我們還需要加入處理玩家由左向右滑動的手勢，因此我們需要修改 MainForm 的 OnGesture 事件處理函式，請在 OnGesture 中加入如下的程式碼：

```
          if (EventInfo.GestureID = sgiRight) then
          begin
             frmGameStatistics1.LoadGameStatistics(GetGameInfoFile());
```

```
            faniGameStatistics.Enabled := false;
            faniGameStatistics.StartValue := -Layout1.Width;
              faniGameStatistics.StopValue := 0;
              faniGameStatistics.Enabled := true;
              Handled := true;
          end;
```

在上面的程式碼中先判斷是由左向右滑動的手勢的話就先呼叫 **frmGameStatistics** 的 **LoadGameStatistics()**方法從遊戲資訊組態檔案中讀出上次儲存的資訊。

現在在 **IDE** 中開啟第 **2** 個 **Frame** 物件 **frmGameStatistics**，並且在其中加入 **LoadGameStatistics()**方法並實作程式碼如下：

```
001   procedure TfrmGameStatistics.LoadGameStatistics(const
sGameInfoFile: String);
002   var
003     pConfigFile : TIniFile;
004     sDT : String;
005     sBubbles : String;
006   begin
007     if (FileExists(sGameInfoFile)) then
008     begin
009       pConfigFile := TIniFile.Create(sGameInfoFile);
010       try
011     sDT := pConfigFile.ReadString('遊戲', '時間', '');
012     sBubbles := pConfigFile.ReadString('遊戲', '捏破泡泡數目', '') ;
013       finally
014     pConfigFile.Free;
015       end;
016
017       txtGameTime.Text := '上次遊戲時間 : ' + sDT;
018       txtBubbleCount.Text := '捏破泡泡數目 : ' + sBubbles;
019     end
020     else
021     begin
022       txtGameTime.Text := '上次遊戲時間 : 無';
023       txtBubbleCount.Text := '捏破泡泡數目 : 0';
024     end;
025   end;
```

LoadGameStatistics()同樣使用 TIniFile 物件從遊戲資訊組態檔案中讀出上次儲存的遊戲時間和捏破泡泡的數目並且顯示在 Frame 的 2 個 Text 組件中。

最後當然要在 ResetGame()方法中加入呼叫 WriteGameInfo()方法把上次捏破泡泡的數目等資訊寫入 ini 檔案中：

```
procedure TfmMainForm.ResetGame();
var
  iRow, iCol : Integer;
begin
  WriteGameInfo();
  SetupGameInfoFrame();
  …
```

現在編譯和執行此範例 App 並且隨意點選泡泡，再搖動手機重新開始遊戲並且儲存遊戲資訊，使用手指向手機螢幕的左方滑動，再用手指向手機螢幕的右方滑動，就可以看到 2 個 Frame 物件從右向左以及從左向右慢慢的滑動出現了，如下所示：

版權所 翻印

到此我們已經成功的說明了如何在 iOS App 中儲存資訊到檔案的方法了，但目前只能儲存一次的資訊，可不可以儲存任意數目的資訊呢？當然可以，讓我們順便說明如何在 iOS App 中使用資料庫的功能以便用資料庫來儲存資訊吧。

11-6 把遊戲資訊儲存到資料庫吧

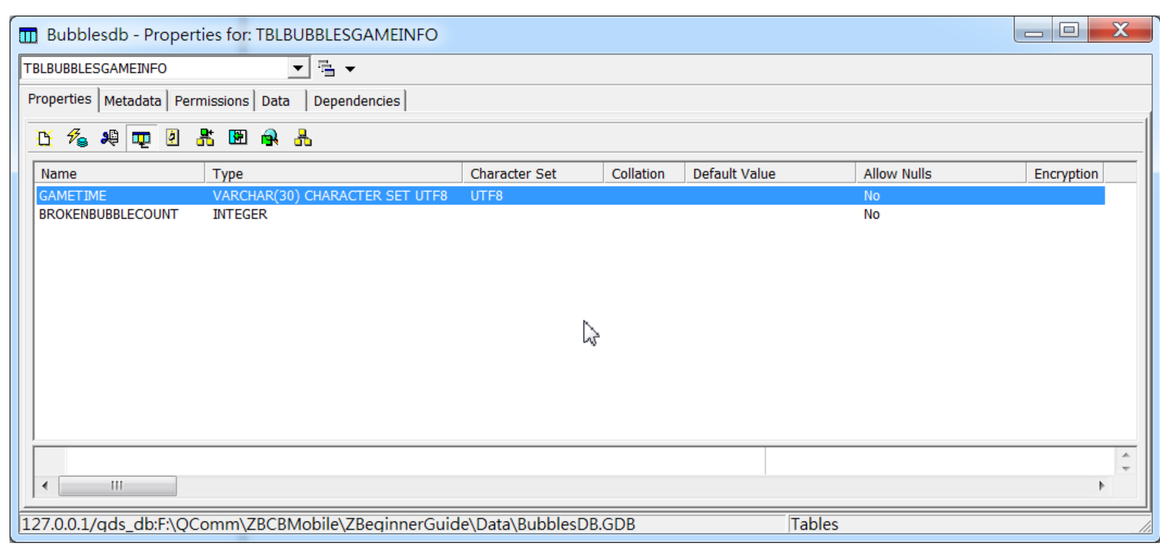
Delphi RIO 中包含了新的資訊庫存取元件框架 FireDAC，由於 FireDAC 比 dbExpress 更適合於使用來開發移動 App，因此筆者強烈建議讀者使用 FireDAC，本小節也將使用 FireDAC 來做為開發說明使用的資料庫存取技術，同時本小節使用的資料庫是包含在 RIO 中的 InterBase ToGo。

要使用 FireDAC 存取資料庫非常的簡單，基本上只需要使用 4 個 FireDAC 元件即可，下面的表格說明了本小節使用的 FireDAC 元件：

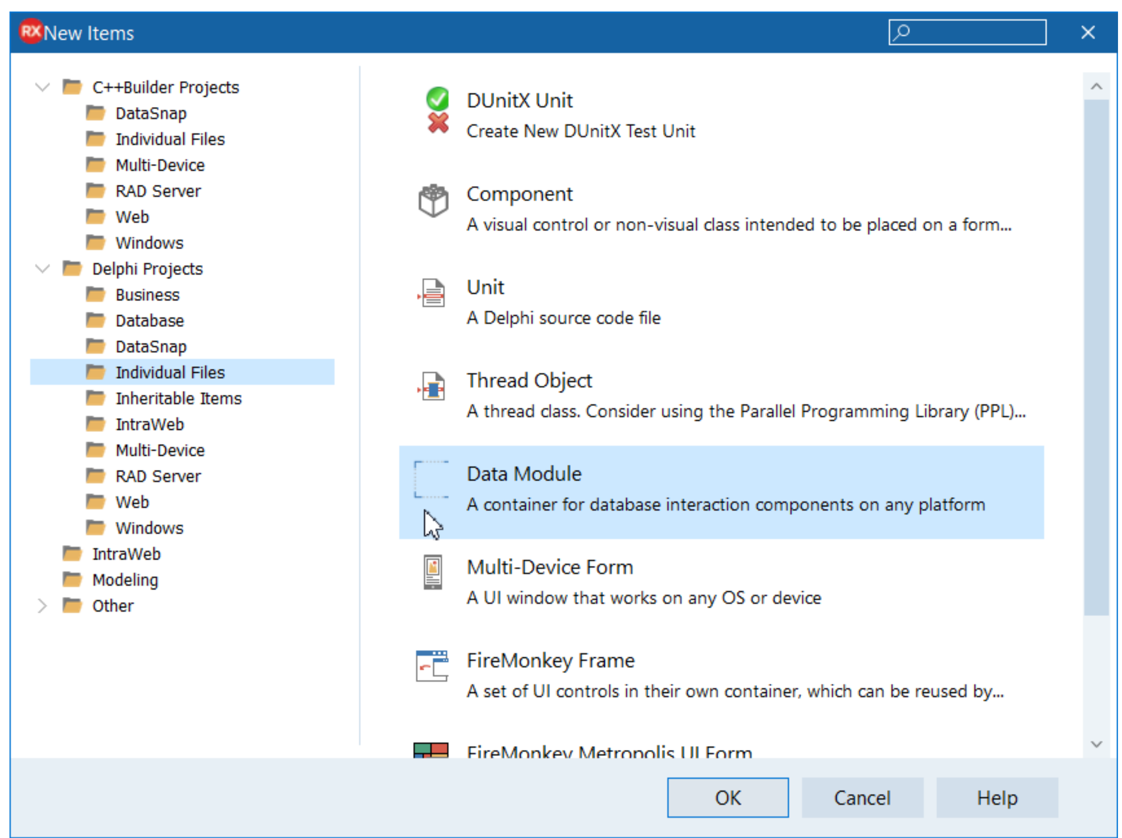
組件	名稱	說明
	TFDConnection	使用來連結資料庫的元件
	TFDQuery	使用來執行 SQL 命令的元件
	TFDPhysIBDriverLink	連結到 Interbase 的驅動程式元件
	TFDGUIxWaitCursor	控制 UI 等候游標的組件

在開始之前我們需要建立一個範例 InterBase 資料庫讀者可以使用 IDE 中的 Explorer 或是 IBConsole 建立 BUBBLESDB.GDB 範例資料庫，在其中建立 TBLBUBBLESGAMEINFO 資料表並且在其中建立下面的 2 個欄位物件(讀者也可以在本書的範例中找到這個資料庫)：

版權所有 請勿翻印

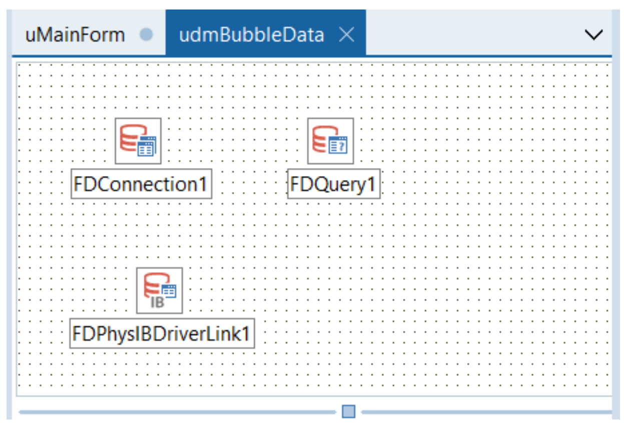


接著在 IDE 中為範例專案加入一個資料模組以便在其中使用 FireDAC 元件存取資料庫，您可以在 IDE 的 New Items 對話盒中選擇建立資料模組，如下所示：

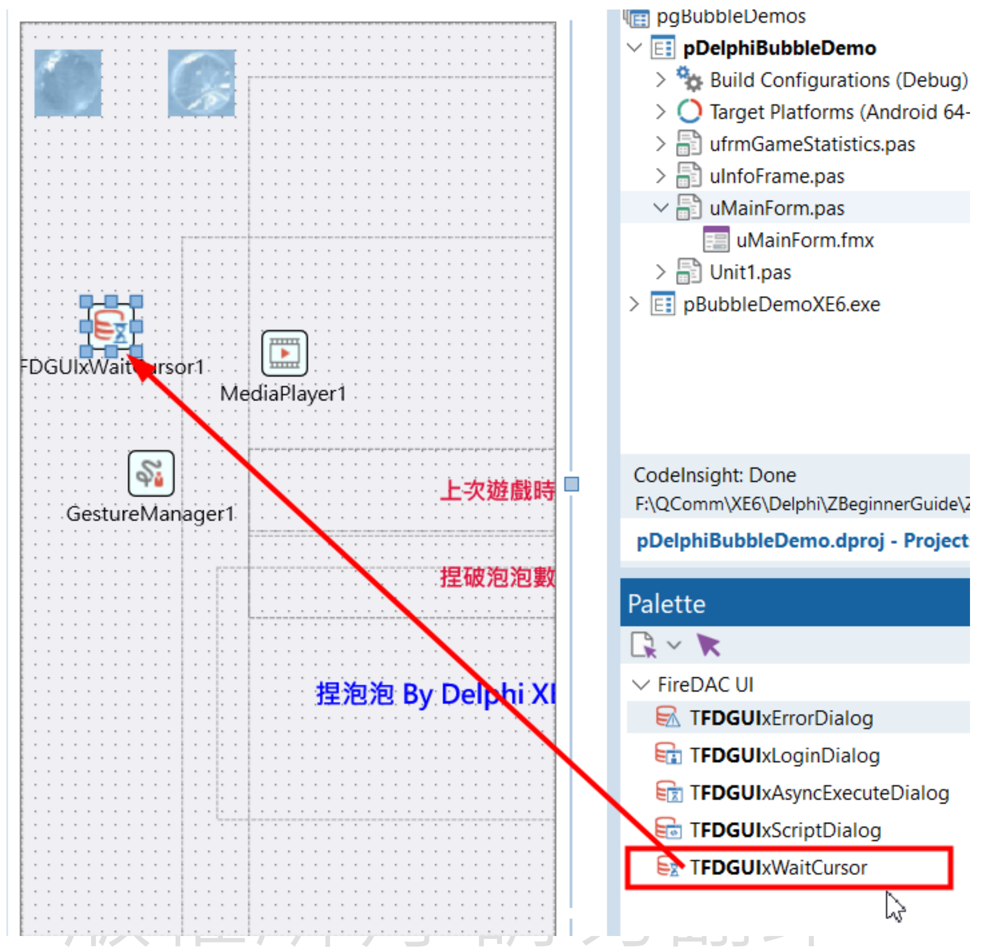


版權所有 請勿翻印

點選了上面的資料模組後 IDE 便會建立一個空白的資料模組，請在其中放入 TFDConnection，TFDQuery 和 TFDPhysIBDriverLink 組件，如下所示：

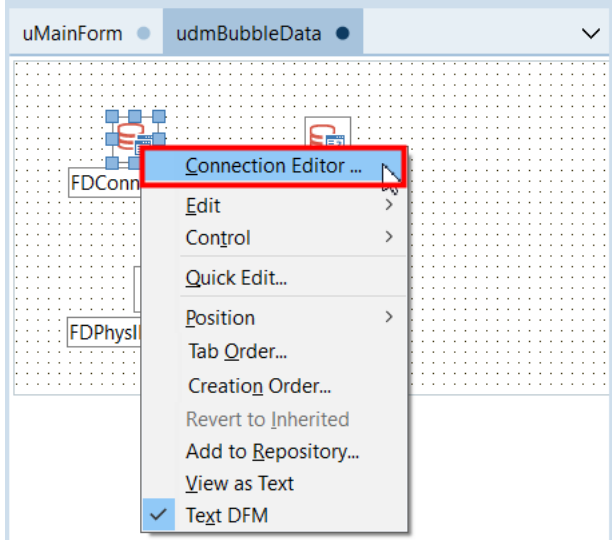


再把 TFDPhysIBDriverLink 元件放入到主表單之中：

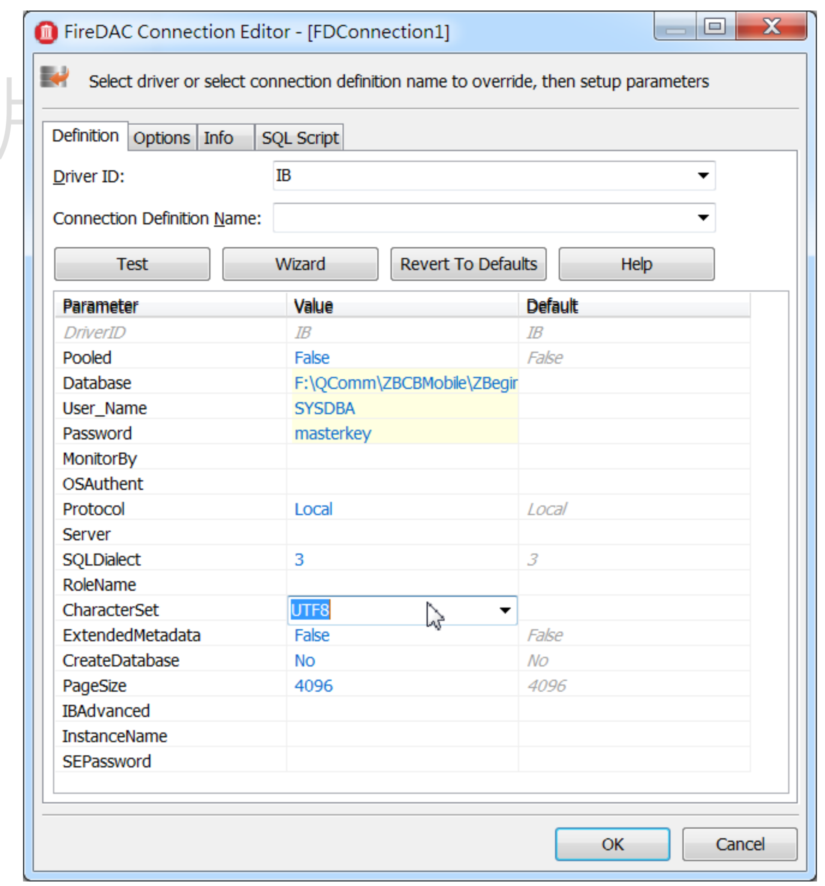


回到資料模組，現在我們要用 TFDConnection 來連結 InterBase ToGo 資料庫，由於現在我們是在設計時期因此可以先直接使用 TFDConnection 連結 InterBase ToGo，等設計好基本的工作之後再於執行時期動態連結真正部署到手機中的 BUBBLESDB.GDB 範例資料庫。

請在資料模組中點選 TFDConnection 再右擊滑鼠，此時便會出現 TFDConnection 元件的快顯選單，請選擇其中的 Connection Editor...選項：



TFDConnection 元件便會顯示下面的元件編輯器，請在下面的組件編輯器 Database 欄位載入 BUBBLESDB.GDB 範例資料庫，並且在 User_Name，Password，CharacterSet 欄位設定如下的數值：



在物件檢視器中設定 TFDConnection 組件的 LoginPrompt 為 false。接著點選資料模組中的 TFDQuery 元件在物件檢視器中設定它的 SQL 特性值為

```
select * from TBLBUBBLESGAMEINFO
```

以便從 TBLBUBBLESGAMEINFO 資料表中存取資料。

接著在資料模組的 OnCreate 事件處理函式和 OnDestry 事件處理函式中撰寫如下的程式碼開啟和 InterBase ToGo 的連結並且取得 TBLBUBBLESGAMEINFO 中的資料：

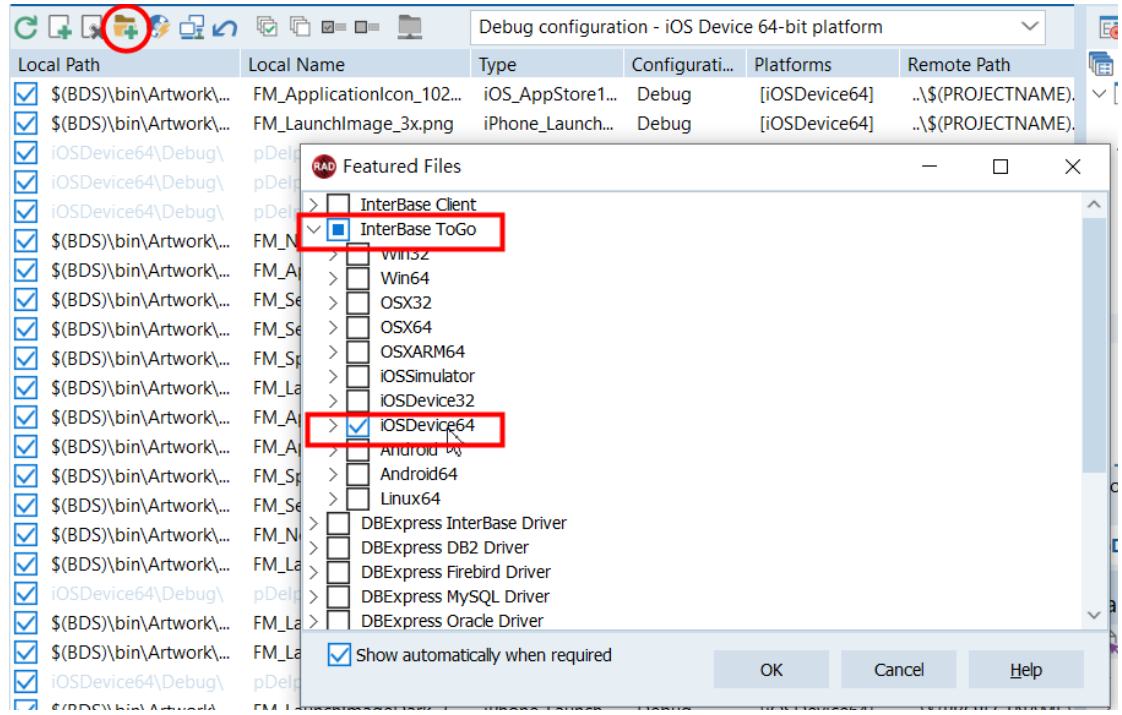
```
procedure TdmGameData.DataModuleCreate(Sender: TObject);
begin
  FDConnection1.Connected := true;
  FDQuery1.Active := true;
end;

procedure TdmGameData.DataModuleDestroy(Sender: TObject);
begin
  FDConnection1.Connected := false;
end;
```

再為 TFDConnection 組件的 OnBeforeConnect 事件處理函式撰寫如下的程式碼，在 OnBeforeConnect 我們需要從 App 的沙箱 Documents 目錄中載入 BUBBLESDB.GDB 範例資料庫：

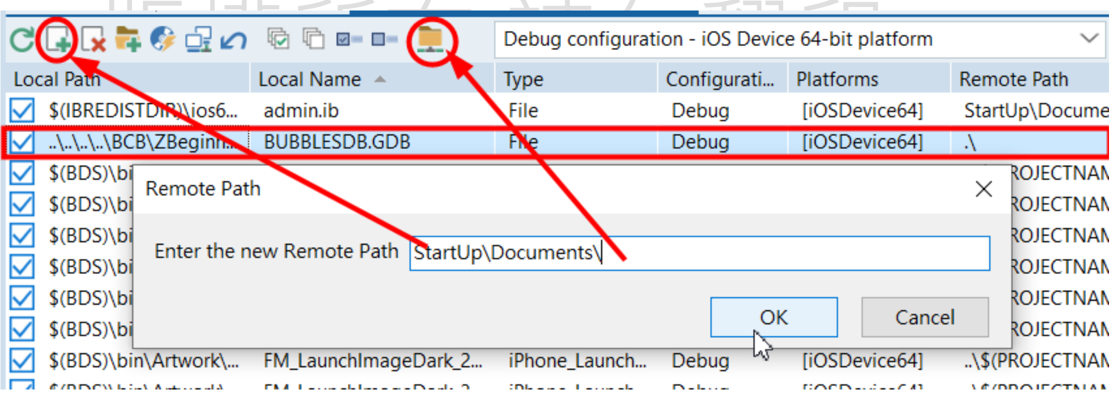
版權所有 請勿翻印

```
procedure TdmGameData.FDConnection1BeforeConnect(Sender: TObject);
begin
  FDConnection1.Params.Values['Database'] := TPath.GetDocumentsPath() +
PathDelim +'BUBBLESDB.GDB';
end;
```

因此我們也需要使用 IDE 的部署精靈把 BUBBLESDB.GDB 範例資料庫部署到 App 沙箱的 Documents 目錄中。請在 IDE 中先啟動部署精靈，先點選上方的 Add Featured Files 快速鍵加入要在此 iOS App 中使用 InterBase ToGo 的功能，Delphi 就會在部署 App 時也順便部署 InterBase ToGo 的相關函式庫：



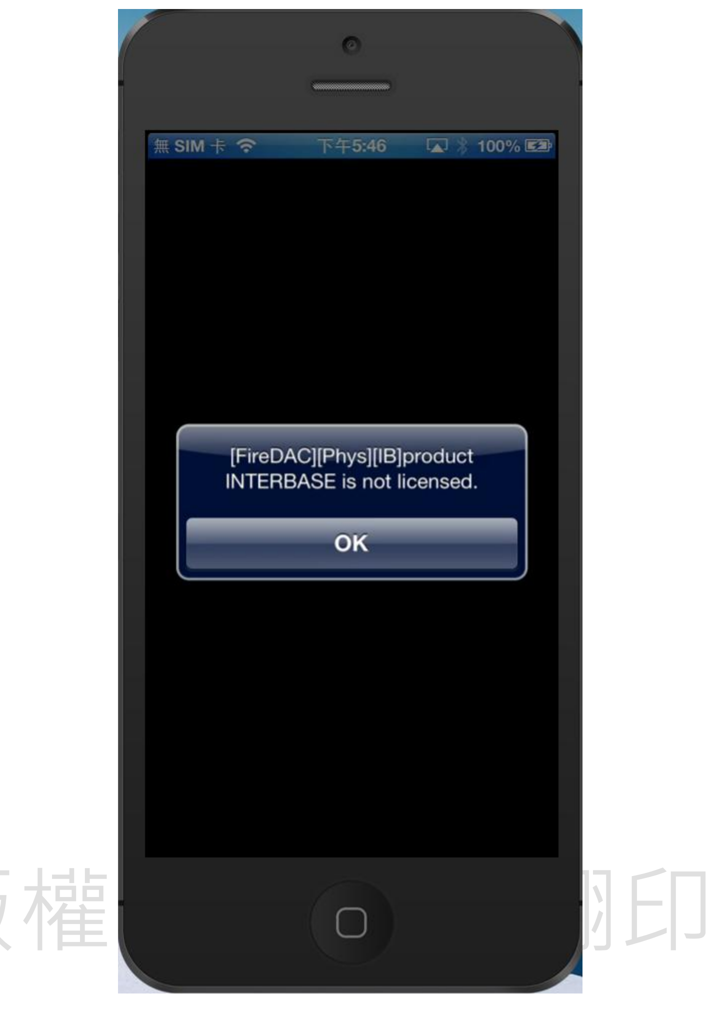
接著點選 Add Files 快速鍵加入部署 BUBBLESDB.GDB 到 StartUp\Documents\目錄中：



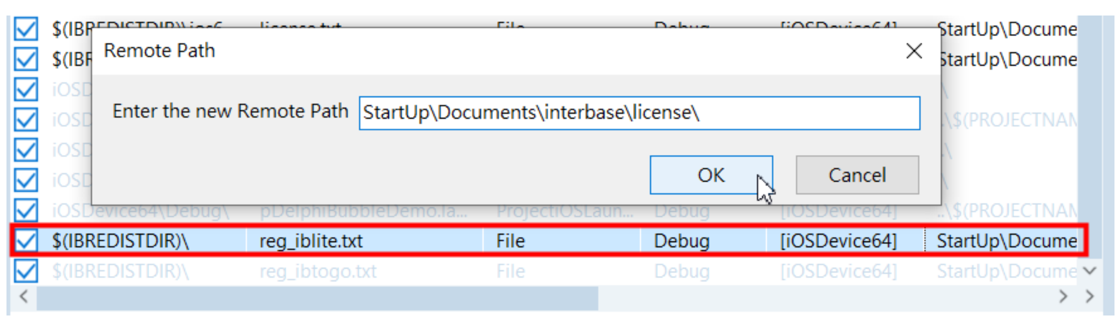
此外您需要到下面的ＵＲＬ取得 InterBase ToGo 的部署授權檔一起部署：

```
http://docwiki.embarcadero.com/RADStudio/RIO/en/IBLite_and_IBToG
o_Test_Deployment_Licensing
```

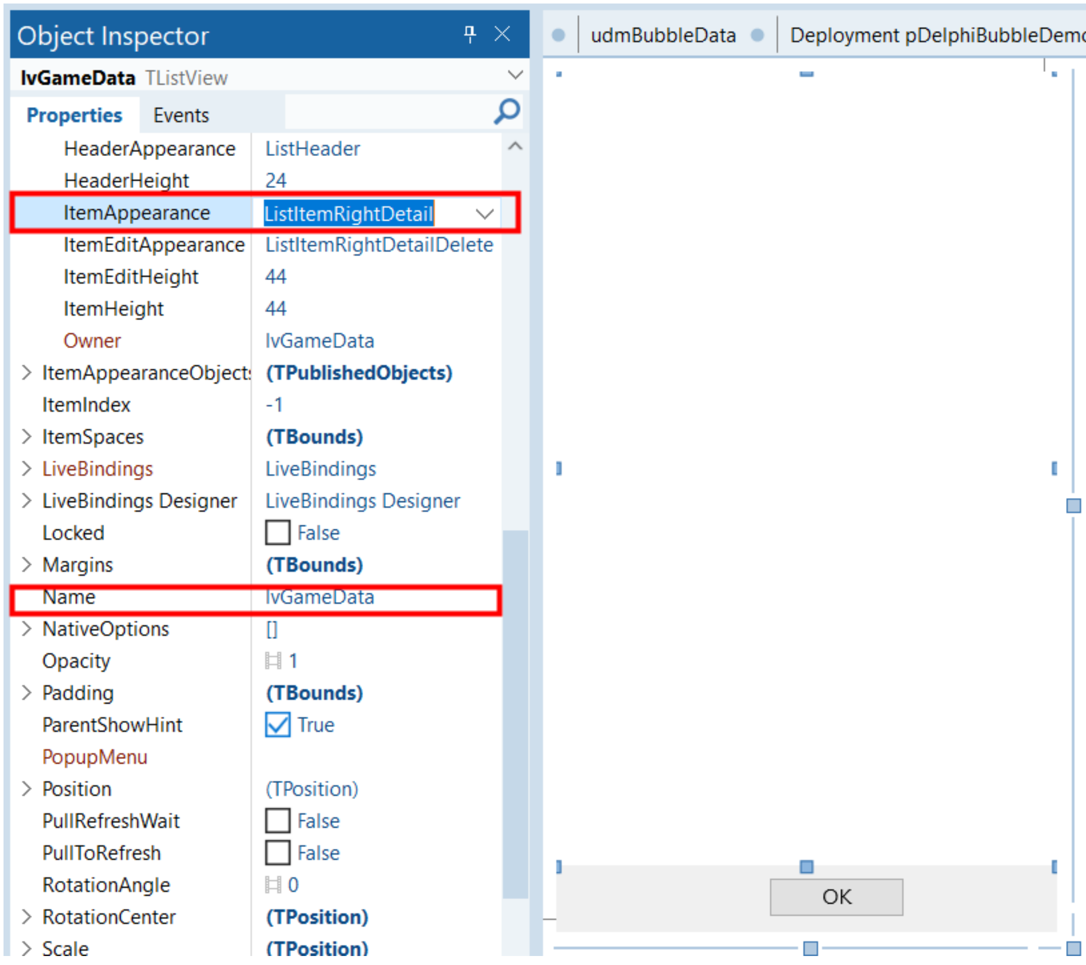
否則在此範例 App 執行時您便會看到下面的錯誤訊息：



最得了 InterBase ToGo 的部署授權檔之後請再使用 Add Files 快速鍵加入部署 InterBase ToGo 的部署授權檔到 StartUp\Documents\interbase\license\目錄中，如下所示：

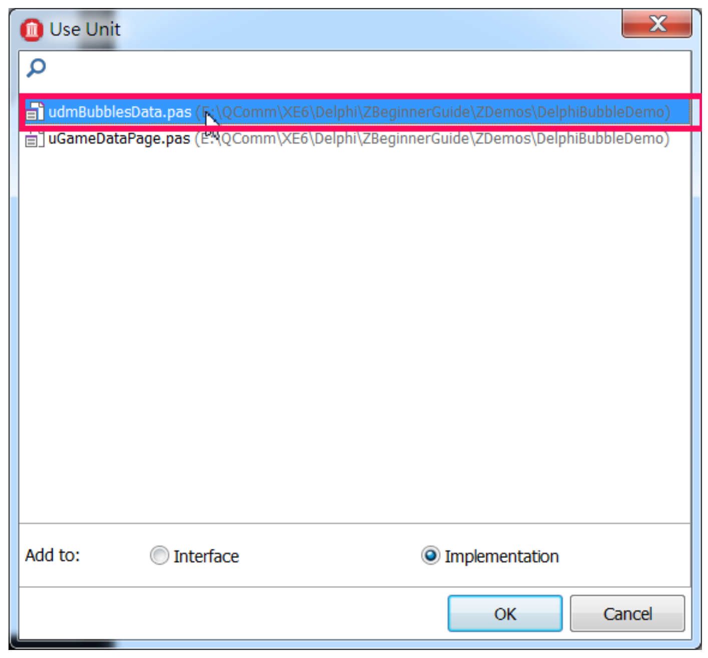


好了， 現在我們還需要一個ＵＩ來顯示資料庫中的資料，請於專案中再建立一個新的 Frame 物件設定它的 Name 特性值為 frmGameData，在其中放入 TListView，ToolBar 和 TButton 組件，設定 TListView 的 Align 特性值為 alClient，再設定它的 ItemAppearance 特性值為 ListItemRightDetail，設定它的 Name 特性值為 lvGameData，如下所示：



同時在主表單中為這個 Frame 物件的 Y 軸加入一個 TFloatAnimation 物件，稍後當玩家使用由上往下的手勢時就動態顯示此 Frame 物件。

當然我們也需要在主表單中使用 Frame 元件把這個 Frame 物件加入到主表單，接著在主表單中點選 File | Use Unit...把資料模組加入到主表單中：



最後的工作就是撰寫實作程式碼了，首先在 ResetGame 方法中加入呼叫 WriteGameData()方法把遊戲資料寫入資料庫中：

```
procedure TfmMainForm.ResetGame();
var
  iRow, iCol : Integer;
begin
  WriteGameInfo();
  WriteGameData();
  SetupGameInfoFrame();
  for iRow := 0 to IROWS -1 do
  begin
    for iCol := 0 to ICOLS - 1 do
    begin
      if (BubblePoppedStatus[iRow, iCol]) then
      begin
        pBubbles[iRow,
iCol].MultiResBitmap.Assign(imgOriNormalBubble.MultiResBitmap);
        BubblePoppedStatus[iRow, iCol] := false;
      end;
    end;
  end;
end;
```

WriteGameData()方法是呼叫資料模組的 WriteGameData()方法同時傳入目前遊戲時間和搗破的泡泡數傳給它：

```
procedure TfmMainForm.WriteGameData();
begin
  dmGameData.WriteGameData(Now, GetBrokenBubles());
end;
```

資料模組的 WriteGameData()方法非常簡單，它使用 TFDQuery 元件把資料新增到 TBLBUBBLESGAMEINFO 資料表中：

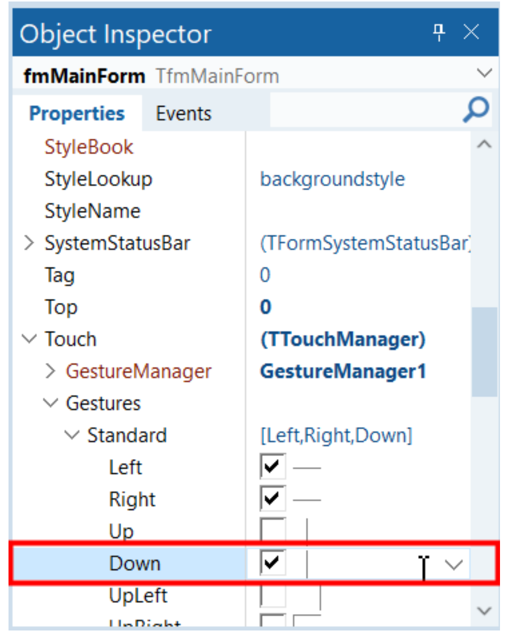
```
procedure TdmGameData.WriteGameData(const dt : TDateTime; const iBubbles :
Integer);
begin
    FDQuery1.Insert();
    FDQuery1.Fields.Fields[0].Value := dt;
    FDQuery1.Fields.Fields[1].Value := iBubbles;
    FDQuery1.Post();
    FDQuery1.Refresh();
end;
```

版權所有 請勿翻印

再回到主表單的 OnGesture 事件處理函式中加入呼叫 FillGameData()方法，FillGameData()方法的目的是把 TBLBUBBLESGAMEINFO 資料表中的資料顯示在由上往下出現的 frmGameData 物件中。

```
      if (EventInfo.GestureID = sgiDown) then
      begin
        FillGameData();
        faniGameDataY.Enabled := false;
        faniGameDataY.StartValue := -Layout1.Height;
        faniGameDataY.StopValue := 0;
        faniGameDataY.Enabled := true;
        Handled := true;
      end;
```

記得要也要讓 MainForm 支援向下的手勢：



當然我們也需要在 SetupGameInfoFrame()方法中設定 frmGameData 的起始位置，我們把它設定在主表單的上方位置：

```
procedure TfmMainForm.SetupGameInfoFrame;
begin
  frmGameInfo1.Position.X := Layout1.Width;
  frmGameInfo1.Position.Y := 0;
  frmGameInfo1.Parent:= Self;

  frmGameStatistics1.Position.X := -Layout1.Width;
  frmGameStatistics1.Position.Y := 0;
  frmGameStatistics1.Parent := Self;

  frmGameData1.Position.X := 0;
  frmGameData1.Position.Y := -Layout1.Height;
  frmGameData1.Parent := Self;
end;
```

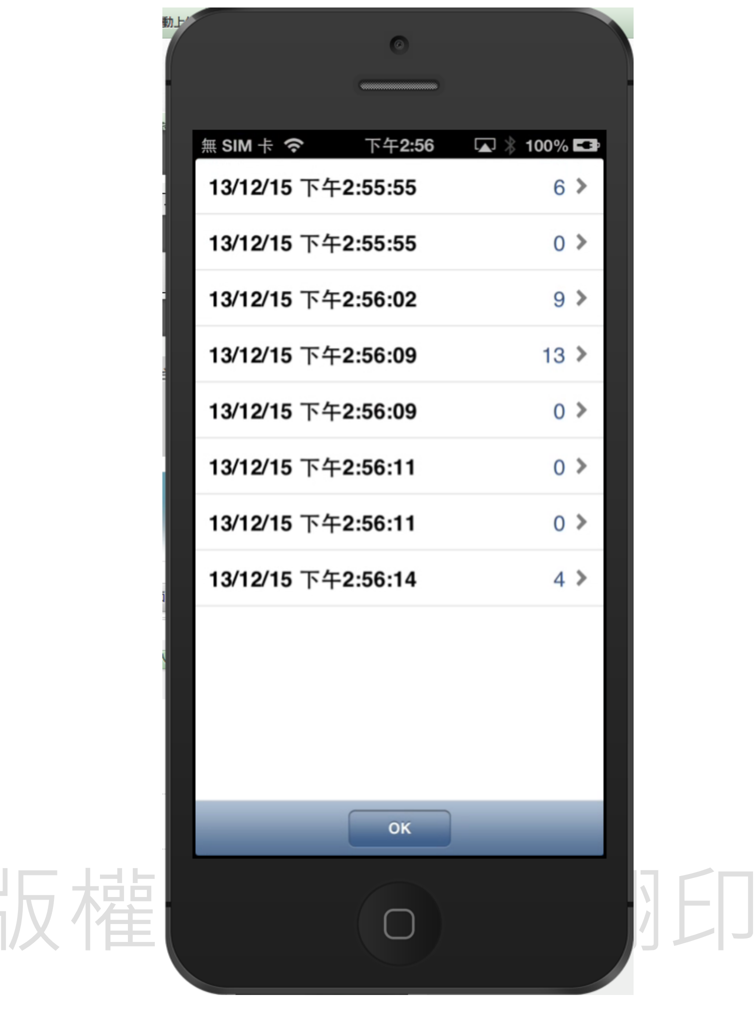
FillGameData() 則使用 TFDQuery 元件一一的從 TBLBUBBLESGAMEINFO 資料表中取出資料並且顯示在 frmGameData 的 lvGameData 之中：

```
procedure TfmMainForm.FillGameData;
var
  plvi : TListViewItem;
```

```
begin
  frmGameData1.lvGameData.Items.Clear();
  frmGameData1.lvGameData.Items.BeginUpdate();
  try
     dmGameData.FDQuery1.First();
    while (not dmGameData.FDQuery1.Eof) do
    begin
      plvi := frmGameData1.lvGameData.Items.Add();
      plvi.Text := dmGameData.FDQuery1.Fields.Fields[0].Value;
      plvi.Detail := dmGameData.FDQuery1.Fields.Fields[1].Value;
      dmGameData.FDQuery1.Next();
    end;
  finally
      frmGameData1.lvGameData.Items.EndUpdate();
  end;
end;
```

現在請編譯範例 App 點選泡泡再搖動手機儲存並重新開始遊戲,如此反復數次之後再使用手指從手機上方往下方滑動，就可以看到類似如下的畫面，範例 App 的遊戲資訊果然成功的儲存到 InterBase ToGo 資料庫並且能夠顯示出來了：

版權所有 請勿翻印



版權 印

11-7 分享遊戲的樂趣吧

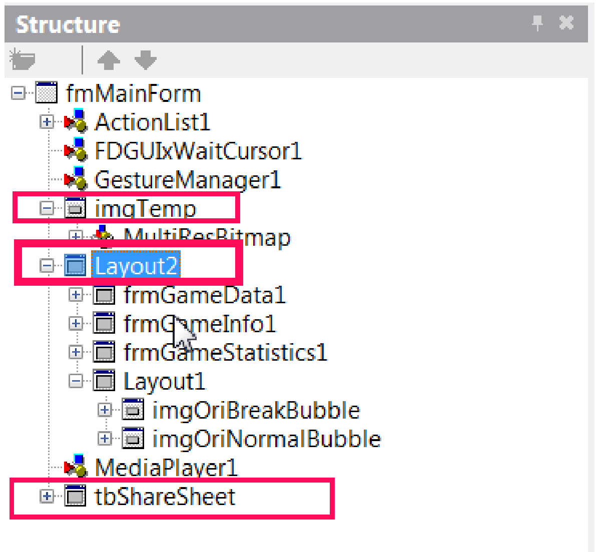
在結束本小節之前，讓我們再加入一個分享的功能，讓我們能夠把玩這個遊戲的樂趣分享給您的好友吧，這只需要用 Delphi RIO 中的 ShareSheet 功能就可以輕易的完成，而 ShareSheet 功能則內含在 TActionList 組件。

首先在主表單中加入一個 ToolBar 元件設定它的 Align 特性值為 alBottom，再於其中放入一個 TButton 元件設定它的 StyleLookup 特性值為 actiontoolbuttonbordered 以及 Name 特性值為 btnShareGameInfo，再放入一個 TActionList 元件，此時主表單如下所示：

接著繼續在主表單中加入一個 TLayout 元件 Layout2　，一個 TImage 元件設定它的特性值如下：

特性	特性值
Name	imgTemp
Align	Client
Visible	False

最後再于 IDE 左上方的架構視窗中移動前面 3 個 Frame 物件以及 Layout1 物件於新的 Layout2 物件之中，此時架構視窗會顯示主表單中所有元件的關係如下所示：

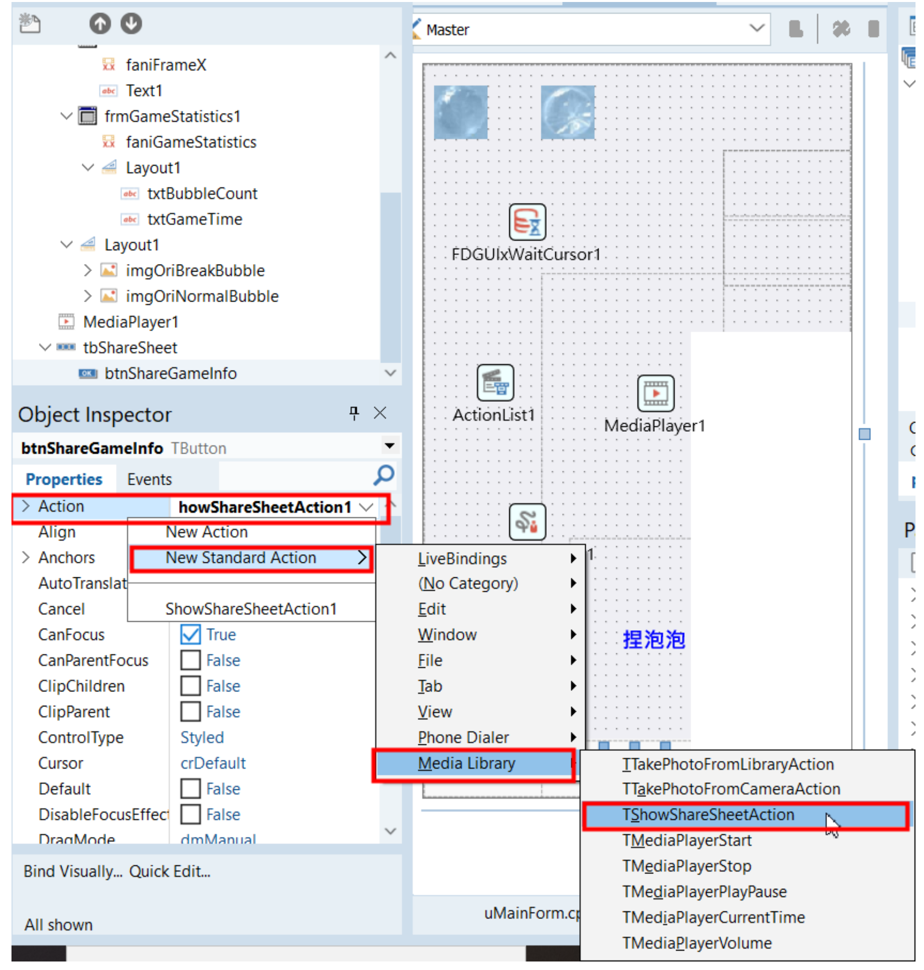


把前面 3 個 Frame 物件以及 Layout1 物件移動到新的 Layout2 物件之中是因為稍後我們要把手機遊戲畫面截取下來並且以圖形的方式分享出去，您很快會看到如何做到。

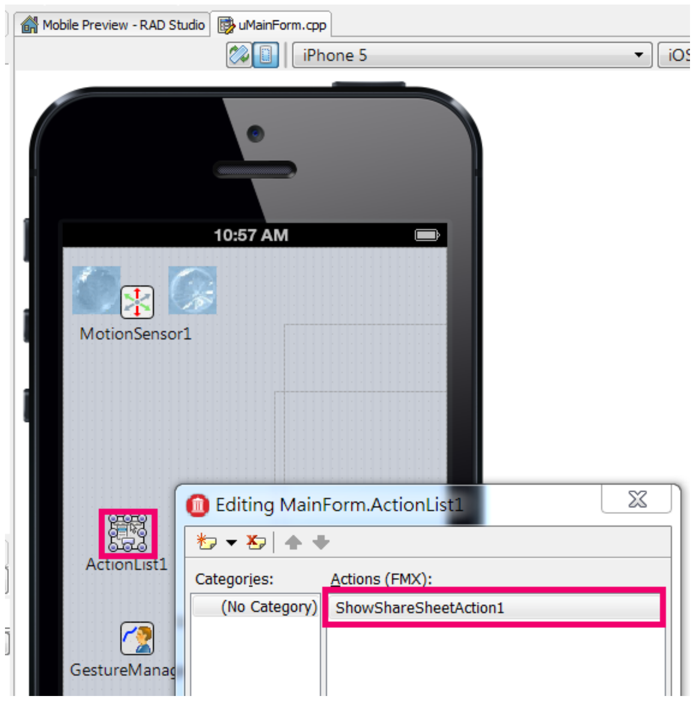
接著點選 ToolBar 中的 btnShareGameInfo 元件，我們希望玩家稍後點選這個 TButton 元件之後就可以把遊戲的資訊分享出去，這很容易就能做到。

版權所有 請勿翻印

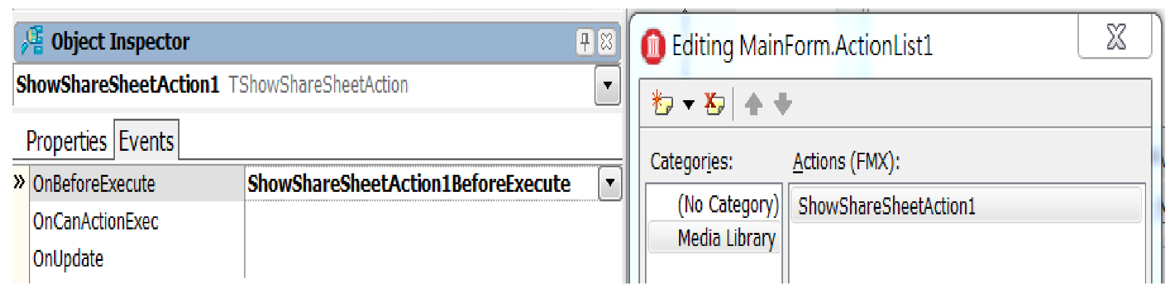
點選了 btnShareGameInfo 之後在物件檢視器中於它的 Action 特性中點選 New Standard Action | Media Library | TShowShareSheetAction 選項讓 btnShareGameInfo 的被點選時就觸發 TShowShareSheetAction 的動作，也就是分享的動作，如下所示：



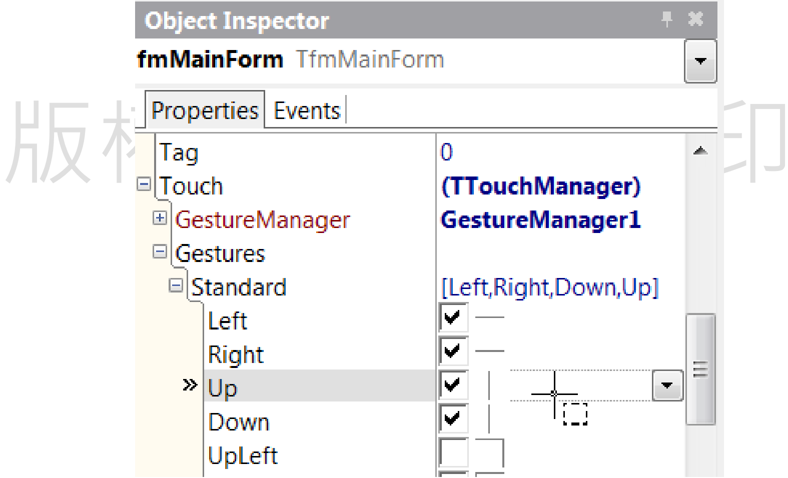
再按兩下主表單中的 TActionList 元件，此時在其中就會自動產生一個 TShowShareSheetAction 物件，如下所示。



請再點選 TShowShareSheetAction 物件，在物件檢視器的 Events 頁次中為它建立 OnBeforeExecute 事件處理函式，如下所示。OnBeforeExecute 事件處理函式會在 TShowShareSheetAction 物件分享行動之前執行，因此我們就可以在這個事件處理函式中把遊戲資訊準備好並且放入 TShowShareSheetAction 物件中再讓 TShowShareSheetAction 物件分享出去。



最後回到主表單，在它支援的手勢種類中再加入支持由下往上的手勢動作，如下所示：



好了，現在就可以開始撰寫實作程式碼了，首先到主表單的 OnGesture 事件處理函式中加入處理由下往上的手勢動作。在下麵的實作程式碼中當範例 App 查覺玩家做出了由下往上的手勢動作之後就呼叫 GetScreenImage() 和 SetupToolBarPosition(true)方法。

```
if (EventInfo.GestureID = sgiUp) then
begin
  GetScreenImage();
  SetupToolBarPosition(true);
end;
```

GetScreenImage()方法的工作就是把目前手機的遊戲畫面截取出來並且指定給前面加入的 imgTemp 物件。要截取手機畫面很簡單，因為現在顯示泡泡的

圖形以及 2 個 Frame 物件的內容都是包含在 Layout2 物件中，因此只需要呼叫 Layout2 物件的 MakeScreenshot()方法就可以快照手機遊戲畫面，而且 MakeScreenshot()方法會回傳代表手機遊戲畫面的 TBitmap 物件，因此我們只需要再直接把它指定給 imgTemp 物件的 Bitmap 特性即可：

```
procedure TfmMainForm.GetScreenImage;
begin
  imgTemp.Bitmap.Assign(Layout2.MakeScreenshot());
end;
```

SetupToolBarPosition()方法控制前面加入的 ToolBar 是否要出現在手機畫面之中，如果玩家做出了由下往上的手勢動作就代表玩家要分享遊戲資訊，就顯示出 ToolBar 好讓玩家可以點選其中的分享按鈕元件把遊戲資訊分享出去：

```
001    procedure TfmMainForm.SetupToolBarPosition(const bVisible :
Boolean);
002    begin
003      if (bVisible) then
004      begin
005          tbShareSheet.Align := TAlignLayout.Bottom;
006          tbShareSheet.BringToFront();
007      end
008      else
009      begin
010          tbShareSheet.Align := TAlignLayout.None;
011          tbShareSheet.Position.Y := Layout1.Height;
012      end;
013      tbShareSheet.Visible := bVisible;
014    end;
```

在上面的程式碼中如果參數 bVisible 是 true 的話 005 行就設定 Toolbar 的 Align 特性值為 alBottom 讓它出現在手機畫面的下方，006 行把 Toolbar 提升到手機畫面的最上方讓它不會被其他的物件遮擋。如果參數 bVisible 是 false 的話就代表分享動作已完成或是取消了，010 行就設定 Toolbar 的 Align 特性值為 alNone，再把 Toolbar 的位置移出手機畫面之外讓玩家看不到它。

接受我們需要修改 SetupGameInfoFrame()方法把 3 個 Frame 物件的父代設定為 Layout2。

```
procedure TfmMainForm.SetupGameInfoFrame;
begin
```

```
  frmGameInfo1.Position.X := Layout2.Width;
  frmGameInfo1.Position.Y := 0;
  frmGameInfo1.Parent:= Layout2;

  frmGameStatistics1.Position.X := -Layout2.Width;
  frmGameStatistics1.Position.Y := 0;
  frmGameStatistics1.Parent := Layout2;

  frmGameData1.Position.X := 0;
  frmGameData1.Position.Y := -Layout2.Height;
  frmGameData1.Parent := Layout2;
end;
```

當玩家點選分享按鈕之後就會觸發 TShowShareSheetAction 物件的 OnBeforeExecute 事件處理函式，在其中我們先把目前玩家捏破的泡泡數指定給 TShowShareSheetAction 物件的 TextMessage 特性，再把剛才截取的手機遊戲畫面指定給 TShowShareSheetAction 物件的 Bitmap 特性，這樣 TShowShareSheetAction 物件就會把這些訊息分享出去了。

```
procedure TfmMainForm.ShowShareSheetAction1BeforeExecute(Sender:
TObject);
begin
    ShowShareSheetAction1.TextMessage := '這次捏破了' +
IntToStr(GetBrokenBubles()) + '泡泡';
    ShowShareSheetAction1.Bitmap.Assign(imgTemp.Bitmap);
    SetupToolBarPosition(false);
end;
```

現在編譯和執行此範例 App 並且隨意點選泡泡，再搖動手機重新開始遊戲並且儲存遊戲資訊，使用手指向手機螢幕的左方滑動，再用手指向手機螢幕的右方滑動，就可以看到 2 個 Frame 物件從右向左以及從左向右慢慢的滑動出現，再使用手指向手機螢幕的上方滑動就可以看到 ToolBar 出現在手機畫面的下方：

版權所有 請勿翻印

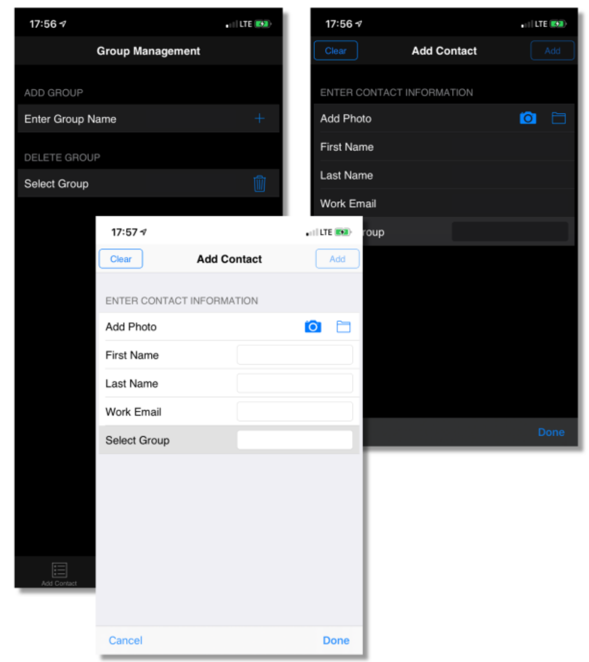
點選下方 Toolbar 中的分享按鈕物件就可以看到手機自動出現可以分享資訊的目的地，讓我們選擇分享到 Facebook，

之後就可以看到如下的畫面，您可以看到在 ShowShareSheetAction1BeforeExecute 事件處理函式中設定的文字資訊和截取的手機遊戲畫面都正確的出現了：

版權所有 請勿翻印

最近如果點選上面畫面中的發佈按鈕之後就可以真的在 Facebook 的網頁中看到如下的執行結果畫面，我們成功的使用 Delphi RIO 的 ShareSheet 功能分享範例 App 的遊戲資訊了，Cool！

Delphi 10.3.3 版開始支持 iOS 13 和 Mac 64 位 Catalina 版，在 iOS 13 方面 Delphi 可開發最新的 Dark Theme App，如下所示：



在 Mac 64 位 Catalina 版方面則支持最新的 notarization 功能。讀者在使用 Delphi 開發 iOS 和 Mac 平臺時必須知道和瞭解這 2 個平台最新的發展。

版權所有 請勿翻印

Delphi for Android入門指引手冊

RIO 強調的功能就是一套原始程式碼就可同時開發多個平臺的應用程式，因此在前面已經說明了如何使用 RIO 開發 iOS App，那麼您現在就可以使用相同的技巧來開發 Android App。不過開發 Android 和開發 iOS 不同的地方是在開發 Android 之前您需要先安裝好 Android SDK 和 Android NDK，並且在 Delphi IDE 中設定好開發環境。

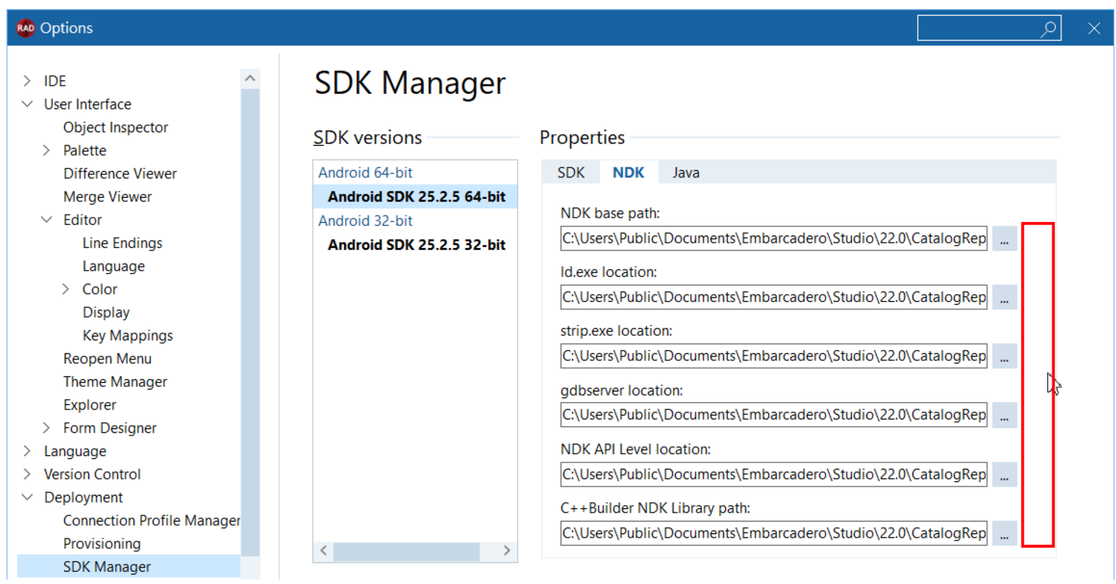
版權所有 請勿翻印

從 RAD Studio 10.3.3 版開始，Delphi 正式支援開發 Andriod 64 位的 App．如果讀者想升級原本的 Android 32 位 App 項目到 64 位，那麼由於 Andriod 64 位項目的設定和 Android 32 位專案有所不同，因此建議讀者重新建立一個 Andriod 64 位專案，再把 Android 32 位元專案中的檔案加入新的 Andriod 64 位元專案。

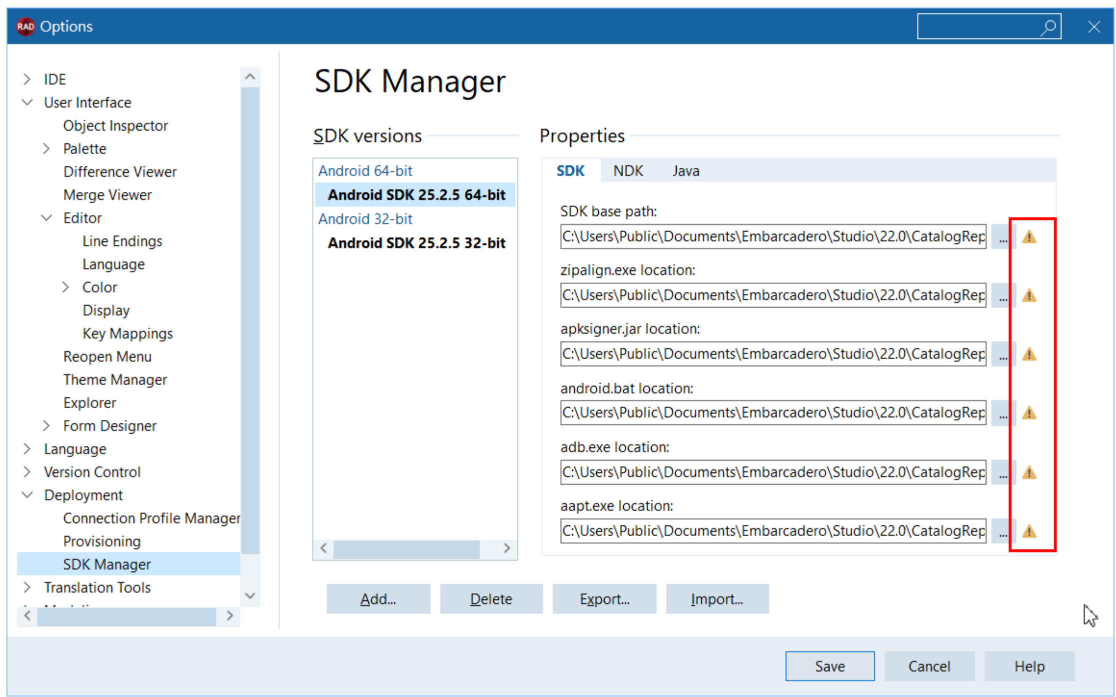
12.安裝和設定 Delphi for Android 開發環境

RAD Studio 的 Android SDK/NDK 安裝方法更改了非常多次，而且許多使用者在使用 RAD Studio 安裝完之後 Android SDK/NDK 可能仍然沒有正確安裝，這對於剛開始學習 Android App 開發的人來說是很挫折的事情。因此在您安裝完 RAD Studio 之後請立刻點選 IDE 的 Tools|Options|Deployment|SDK Manager 查看 SDK/NDK/Java 是不是正確的安裝了。

如果您看到如下的畫面在每一個...按鈕後沒有黃色三角形的圖像就代表安裝正確：



而如果是像如下所示看到黃色三角形圖像就代表安裝不正確，也就是說 RAD Studio IDE 不知道這些需要的相關 Android 程式或是檔案在那裡，而需要您的設定告訴 IDE，下面的錯誤是因為 IDE 找不到 Android 的 Build Tool：



在這裡有數種方法可以解決，首先您可以到 RAD Studio 安裝 Android SDK 的目錄中找到 SDKManager.bat 並執行它來安裝。例如您可以上面 NDK 的設定中找到這個目錄，筆者的安裝目錄便是：

```
c:\Users\Public\Documents\Embarcadero\Studio\22.0\CatalogReposit
ory\AndroidSDK-2525-22.0.42600.6491
```

那麼在上面目錄的\cmdline-tools\bin 子目錄下便有 SDKManager.bat。SDKManager.bat 是一個命令列工具，要使用它您必須下達命令列，例如下面的命令便可安裝 SDK 29 和相關的檔案：

```
sdkmanager "build-tools;29.0.0" "extras;google;usb_driver"
"platforms;android-26" "tools" "platform-tools"
--sdk_root=c:\Users\Public\Documents\Embarcadero\Studio\22.0\Cat
alogRepository\AndroidSDK-2525-22.0.42600.6491
```

不過除非您非常熟悉 Android SDK/NDK，否則筆者會建議您使用 GUI 版本的 SDK 安裝程式用點選的方式安裝會比例方便和齊全。

您可到

```
https://dl.google.com/android/repository/tools_r25.2.5-windows.z
ip
```

下載並解壓縮到一個目錄，然後在\tools 子目錄中執行 Android.bat。

> 在前個 RAD Studio 版本中來有包含 Android.bat 可直接在 RAD Studio 的程式群組中執行，但目前版本又把它拿掉了，因此才需要去下載上面的檔案，

版權所有 請勿翻印

下面就是筆者使用 Android.bat 程式安裝了 Android SDK：

Android 9 (API 28)	
SDK Platform	28
Android TV Intel x86 Atom System Ir	*28*
Intel x86 Atom_64 System Image	*28*
Intel x86 Atom System Image	*28*
Google Play Intel x86 Atom_64 Syste	*28*
Google Play Intel x86 Atom System I	*28*
Google APIs Intel x86 Atom_64 Syste	*28*
Google APIs Intel x86 Atom System I	*28*
Sources for Android SDK	28

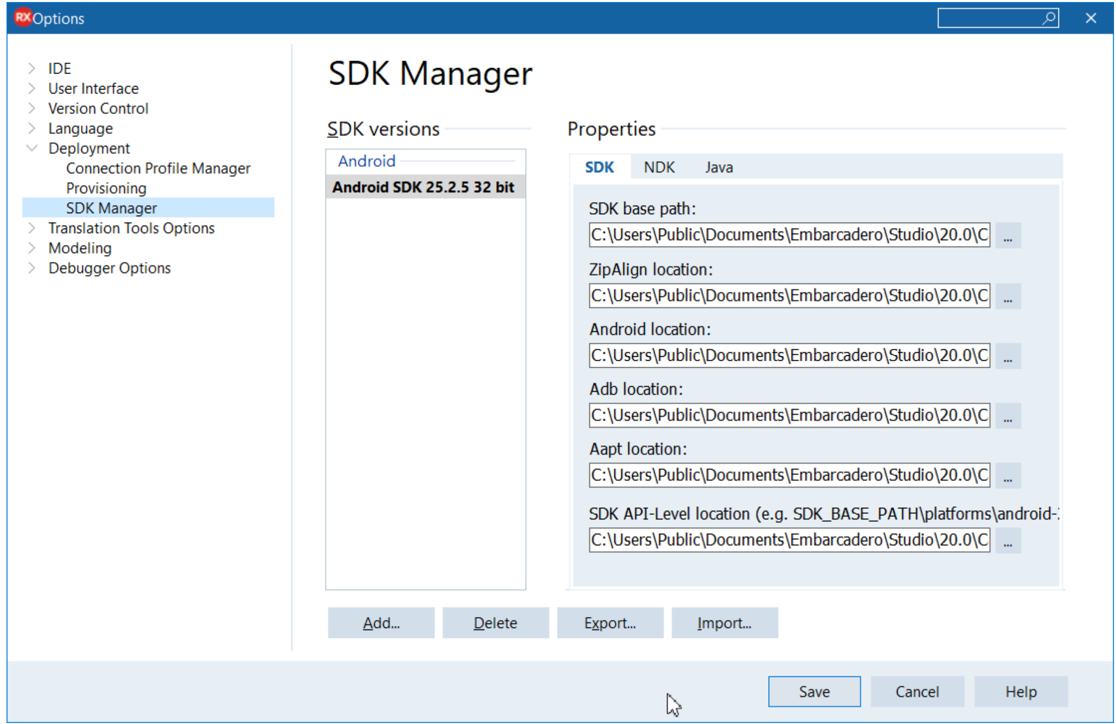
完成安裝 Android SDK 和 Android NDK 後您可以把 Android.bat 安裝的檔案移到和 NDK 一樣的目錄，例如前面的 c:\Users\Public\Documents\Embarcadero\Studio\22.0\CatalogRepository\AndroidSDK-2525-22.0.42600.6491 或是就留在 Android.bat 安裝的地方。但接下來需要在 Delphi IDE 中設定 Android SDK 和 Android NDK 的安裝路徑，以便讓 IDE 能夠找到相關的檔案和工具。例如 Android Tools 把 Android SDK 安裝在：

```
c:\Users\Public\Documents\Embarcadero\Studio\22.0\CatalogRepository\A
ndroidSDK-2525-22.0.42600.6491
```

Android NDK 安裝在：

```
C:\Users\Public\Documents\Embarcadero\Studio\22.0\CatalogRepository\A
ndroidNDK-21-22.0.42600.6491\android-ndk-r21
```

接著請回到 RIO 的 IDE 中，點選 Tools|Options...功能表，並且點選 Options 對話盒中的 SDK Manager 選項，如下所示：

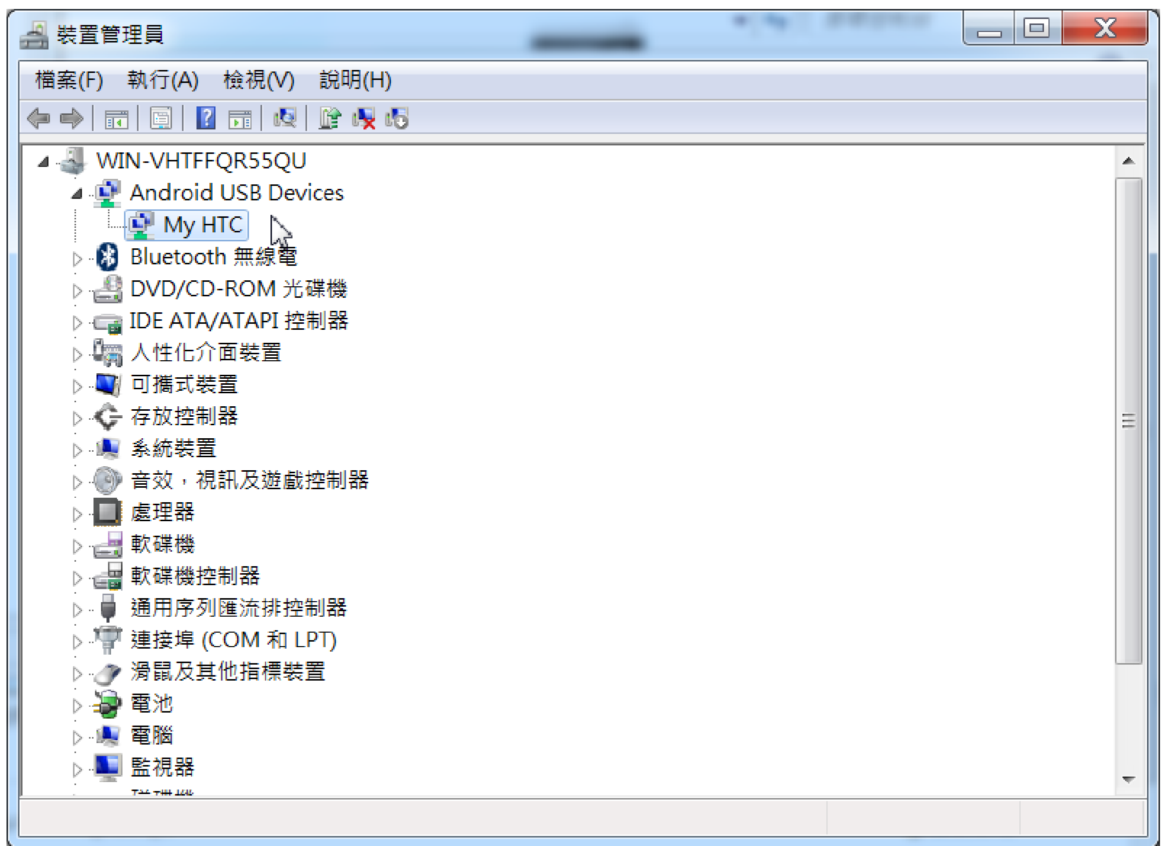


我們只需要把上 SDK 工具設定成 Android Tools 安裝的路徑即可。

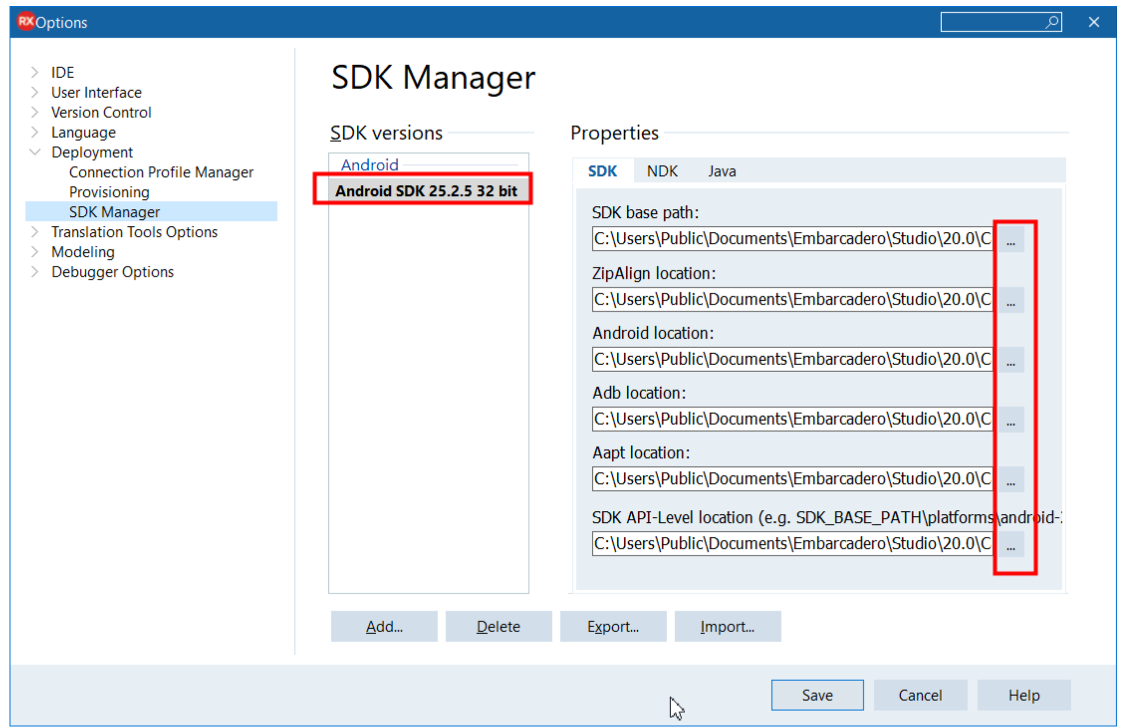
之後讀者需要安裝使用開發的 Android 手機的驅動程式，才能讓 Delphi IDE 部署和測試。讀者需要到您的手機廠商網站下載，例如筆者是使用 hTC Incredible S，因此筆者到 hTC 下列的 URL 下載驅動程式：

```
http://www.htc.com/tw/software/htc-sync-manager/
```

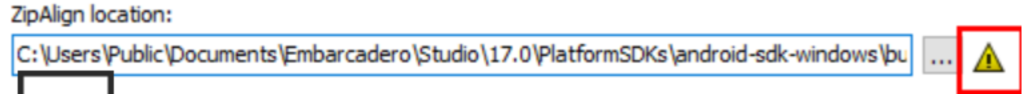
一旦安裝了 hTC 的 Sync Manager 之後筆者在作業系統的裝置管理員中就可以看到 HTC 手機：



在 XE7 之後到 RIO 的版本中，當您安裝完 Delphi 之後 IDE 會在您啟動 Android 開發時自動安裝 Android SDK，並且會正確替您設定好 SDK Manager 中的設定，只要您開啟 Options 對話盒並在 SDK Manager 選項中看到右方沒有顯示這個⚠符號即代表一切安裝和設定是正常的，那您就可以開始 Android 的開發工作：



而如果您在任一選項的右方看到⚠符號，那代表此設定不正確，通常的錯誤原因是因為在左方的目錄中找不到需要的 exe 檔。



例如上方圖形有⚠符號就代表在左方的

```
C:\Users\Public\Documents\Embarcadero\Studio\18.0\PlatformSDKs\a
ndroid-sdk-windows\build-tools\23.0.2\zipalign.exe
```

目錄中找不到 ZipAlign.exe 這個檔案，您只需要在左方指到包含 ZipAlign.exe 檔案的目錄即可修正此錯誤。

接下來我們就可以開始開發 Android App 了。

13.開發 Delphi for Android App

從 RIO 之後 Delphi 正式支援 Android 的開發，而且 iOS 的程式碼可以再使用於 Android 平臺，例如在前面章節示範的 iOS 範例 App 您可以試著把它們移植到 Android 平臺中執行。

在本小節中我們將示範如何使用 Delphi 開發 Android 的 App，由於前面的章節已經說明了基本的概念和技巧，因此在本小節中讓我們試著來開發一些比較有趣的 Android App，我們將使用臺北市政府提供的公共資訊做為範例。

臺北市政府在下面的 URL 提供了許多的公共資訊讓市民能夠查詢使用：

http://data.taipei.gov.tw/opendata

在其中有許多的公共資訊已經是封裝成 JSON 的形式，例如在下面的 URL 中臺北市政府提供了臺北市旅館資料庫的資訊讓市民或是旅遊人士能夠查詢臺北市旅館的相關資訊：

http://data.taipei.gov.tw/opendata/apply/NewDataContent?oid=4418C600-D7F5-46D6-BB1F-1DA29DAB91E9

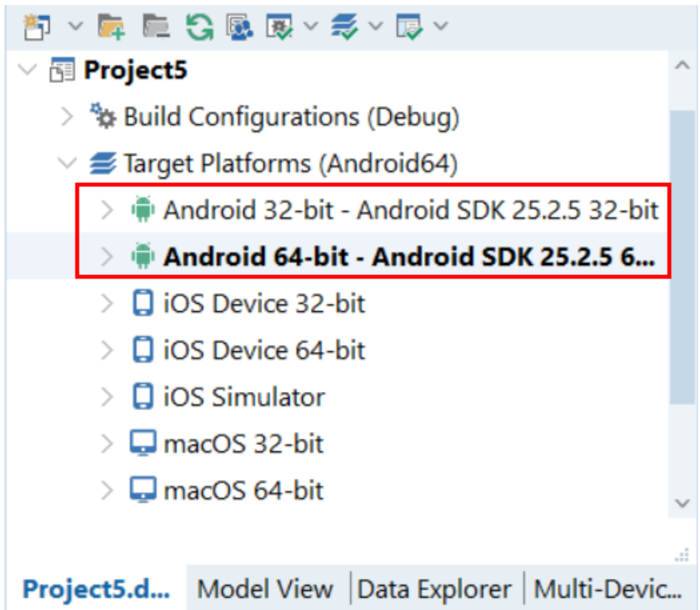
現在讓我們使用 Delphi 來開發一個臺北市旅館查詢 App，在這個範例 Android App 中將提供如下的功能：

- 查詢臺北市旅館
- 聲控查詢
- 啟動流覽器查詢旅館網站主頁
- 在雲端為旅館寫評價

版權所有 請勿翻印

這個範例將觸及許多移動平臺開發的技術，在下面的章節中我們將試著一一的完成上面的功能。

在 Delphi 10.3.3 版之後同時支援 32 和 64 位 Android App 的開發：



在下面的內容中讀者可自行選擇開發 32 或/和 64 位 Android App。

13-1 開發查詢臺北市旅館 App

在下麵的 URL 中：

版權所有 請勿翻印

http://data.taipei.gov.tw/opendata/apply/NewDataContent?oid=4418C600-D7F5-46D6-BB1F-1DA29DAB91E9

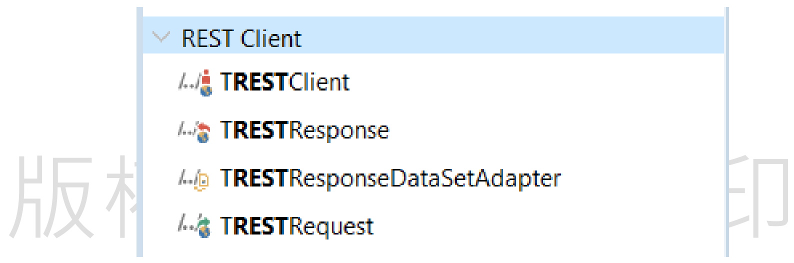
提供的旅館資訊是使用 JSON 封裝的資料，下面是取得結果的部份資料：


```
[{"rownumber":"33","ref_wp":"6","cat1":"住宿","cat2":"一般旅館
","serial_no":"B0225
","memo_tel":"0227735177","memo_fax":"0227727569","memo_cost":"1
280 以上","memo_time":"","stitle":"友統旅館
","xbody":"","avbegin":"2008-10-20","avend":"2010-03-17","idpt":
"臺北旅遊網","xurl":"http://yotong.ffh.com.tw/","address":"臺北市大
安區忠孝東路四段 197 號 13 樓
","xpostdate":"2010-03-17","langinfo":"10","poi":"Y","info":"","
longitude":"121.551654","latitude":"25.041705","file":"<file><im
g description=\"友統旅館
1\">http://www.taipeitravel.net/d_upload_ttn/frontsite/tw/hotel/
B0225/B0225_1.jpg</img><img description=\"友統旅館
2\">http://www.taipeitravel.net/d_upload_ttn/frontsite/tw/hotel/
B0225/B0225_2.jpg</img><img description=\"友統旅館
3\">http://www.taipeitravel.net/d_upload_ttn/frontsite/tw/hotel/
```

```
B0225/B0225_3.jpg</img></file>"},{"rownumber":"364","ref_wp":"6"
,"cat1":"住宿","cat2":"一般旅館","serial_no":"B0138
","memo_tel":"0225971281","memo_fax":"0225971288","memo_cost":"1
400 以上","memo_time":"","stitle":"慶天閣大飯店
","xbody":"","avbegin":"2008-10-20","avend":"2009-07-21"
…
```


從上面的結果中我們可以瞭解整個資料是以 JSON 陣列物件封裝的，而其中每一筆旅館資訊是使用一個 JSON 物件封裝的。

瞭解了這個臺北市政府公共服務的資料封裝規則之後要解析其中的資訊就非常的簡單了，我們可以使用 Indy HTTP 元件取得這個資料再使用 Data.DBXJSON 中的 JSON 相關類別來解析其中的資訊。但是在 Delphi XE5 之後提供了新的 REST Client 元件組讓我們可以更輕鬆的成為 REST 用戶端以使從任何提供 REST 公共服務的來源取得需要的資料。



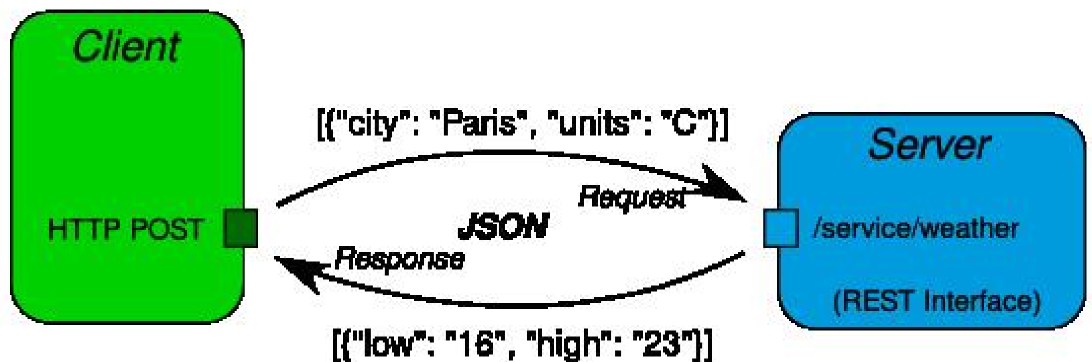
下面的表格說明了上面 4 個 REST Client 元件組的功能：

組件	說明
TRESTClient	指定提供 REST 服務的 REST 伺服器
TRESTRequest	提出 REST 請求向 REST 伺服器要求服務
TRESTResponse	REST 伺服器執行結果回傳到此組件
TRESTResponseDataSetAdapter	把 TRESTResponse 組件中的 JSON 結果轉換成資料集(DataSet)的形式

我們可以使用面的圖形來簡單的說明如何使用上面的元件。在下圖說明瞭一個 RESTful 架構的基本執行流程。左方的 REST 用戶端藉由 HTTP/HTTPS 向 REST 伺服器要求服務，這個要求的服務也是使用 JSON 封裝的，在 REST 伺服器接受到要求並且執行完畢之後就會把執行結果再封裝成 JSON 的形式回傳給 REST 用戶端。

因此我們可以使用上表中的 TRESTClient 元件指定右方 REST 伺服器的所在地，使用 TRESTRequest 提出要求服務，右方 REST 伺服器會把執行結果回傳到 TRESTResponse 元件中，最後我們可以使用

TRESTResponseDataSetAdapter 元件把 JSON 結果換成資料集再儲存於 TClientDataSet 等元件中就可以存取其中的資料了。



瞭解了上面的概念之後我們就可以輕易的使用 REST Client 中的元件完成查詢臺北市旅館資訊的工作：

版權所有 請勿翻印

1. 使用 TRESTClient 元件設定指向 http://data.taipei.gov.tw/opendata/apply/NewDataContent?oid=4418C600-D7F5-46D6-BB1F-1DA29DAB91E9 取得服務

2. 使用 TRESTRequest 元件提出 REST 請求

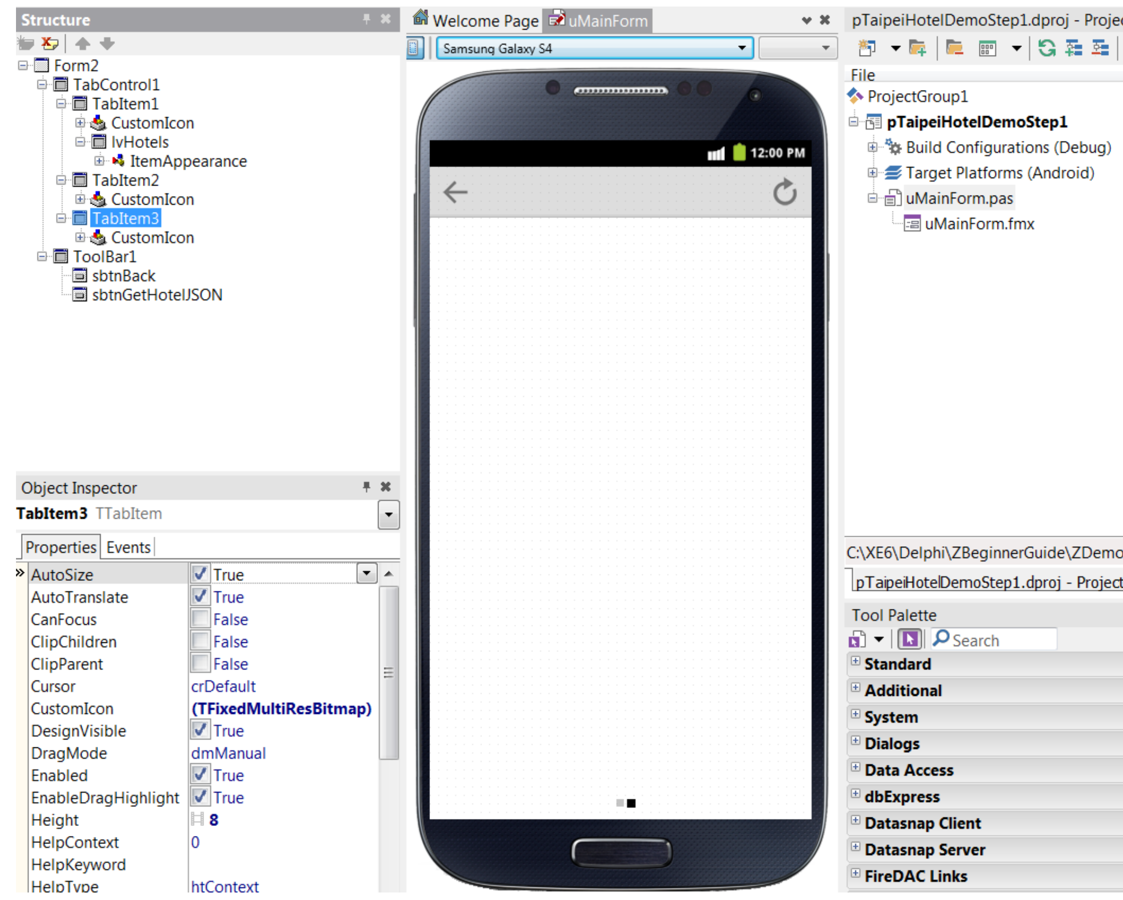
3. REST 請求結果會自動回傳到 TRESTResponse 組件

4. 再使用 TRESTResponseDataSetAdapter 元件把 TRESTResponse 元件中的 REST 執行結果換成資料集

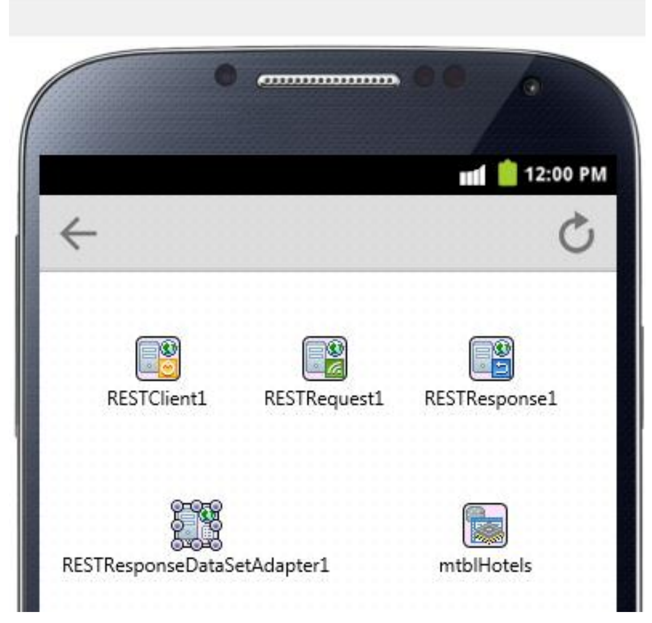
5. 把 TRESTResponseDataSetAdapter 元件的結果儲存在 TClientDataSet 或是 FireDAC 的 TFDMemTable 元件中

臺北市政府的公開資訊網站已經改成 https://gis.taiwan.net.tw/XMLReleaseALL_public/hotel_C_f.json，並且有相關 JSON 欄位說明，請讀者閱讀並瞭解本書範例程式碼說明之後再自行修改範例程式碼以便讓此範例 App 可正常工作，這也可以做為一個練習之目的.

現在請使用 Delphi RIO 建立一個 FireMonkey Mobile 空白專案，在主表單中放入一個 TabControl 並在其中建立個 TabItem，在第 1 個 TabItem 中放入名為 lvHotels 的 TListView 組件並且設定此 TListView 組件的 ItemAppreance|ItemAppeance 子特性值為 ListItemRightDetail。最後再放入 ToolBar 組件並且在其中放入 2 個 TSpeedButton，此時主表單如下所示：



再於主表單中放入前面介紹的 4 個 REST Client 中的元件以及一個 TFDMemTable 元件，如下所示：



然後設它如下的特性值：

TRESTClient 組件：

特性	特性值
BaseURL	http://data.taipei.gov.tw/opendata/apply/query/ NDQxOEM2MDAtRDdGNS00NkQ2LUJCMUYtMURBMjlEQUI5MUU5?$format=json

版權所有 請勿翻印

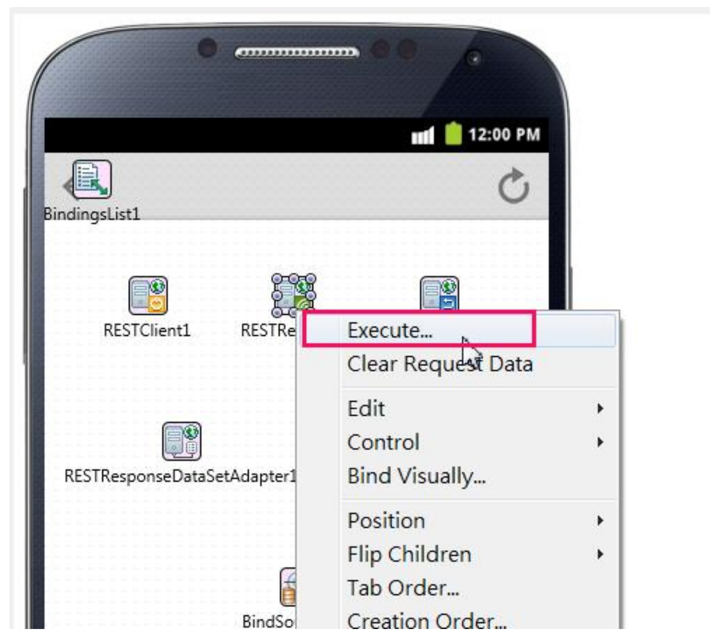
TFDMemTable 組件：

特性	特性值
Name	mtblHotels

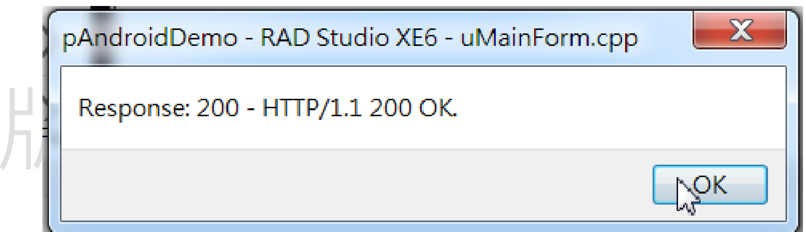
RESTResponseDataSetAdapter1 組件：

特性	特性值
ResponseJSON	RESTResponse1
DataSet	mtblHotels

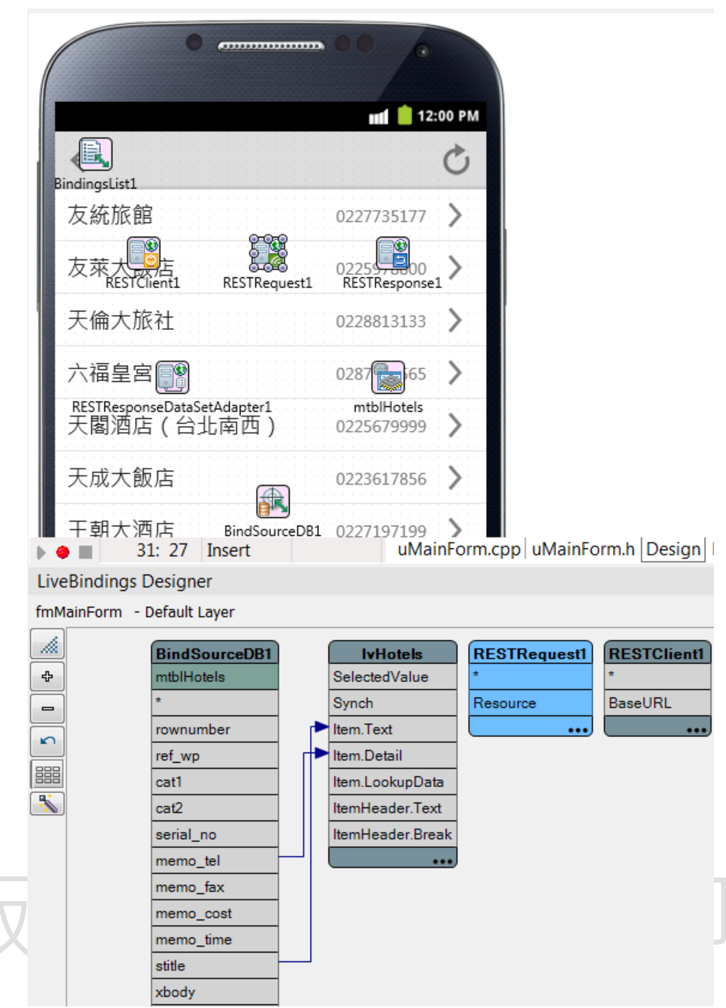
接著我們就可以開始設計臺北市旅館顯示在 lvHotels 之中了，為了方便設計 UI，現在請點選主表單中的 RESTReuqest1 元件，點選滑鼠右鍵選擇 Execute...選項(請確定您的網路連結在工作中)：



過了數秒之後您會看到 IDE 顯示如下的訊息代表 RESTReuqest1 元件提出的 REST 請求已成功執行且結果已回傳到 RESTResponse1 組件中：



接著設定 RESTResponseDataSetAdapter1 組件的 Active 特性值為 true，開啟 LiveBinding 設計家，把 mtblHotelsm元件的 stitle 欄位連結到 lvHotels 的 Item.Text，再把 memo_tel 欄位連結到 lvHotels 的 Item.Detail，此時就可以在 lvHotels 元件中看到臺北市旅館的資訊了：



完成初步的設計之後就可以開始撰寫些簡單的程式碼讓這個範例 App 工作了。首先在主表單的 OnActivate 事件處理函式中關閉 RESTResponseDataSetAdapter1：

```
procedure TfmMainForm.FormActivate(Sender: TObject);
begin
  RESTResponseDataSetAdapter1.Active := false;
end;
```

接著在主表單的中加上方的 TSpeedButton 的 OnClick 事件處理函式中呼叫 GetTaipeiHotelsJSON()方法提出 REST 請求：

```
procedure TfmMainForm.sbtnGetHotelJSONClick(Sender: TObject);
begin
  GetTaipeiHotelsJSON();
end;
```

GetTaipeiHotelsJSON()方法先呼叫 RESTRequest1 的 Execute()方法正式提出 REST 請求，在執行完畢之後就開啟 RESTResponseDataSetAdapter1 以顯示最及時的臺北市旅館的資訊：

```
procedure TfmMainForm.GetTaipeiHotelsJSON;
begin
  RESTRequest1.Execute();
  RESTResponseDataSetAdapter1.Active := true;
end;
```

現在編譯並且執行您應該就可以在您的 Android 手機中點選右上方的按鈕取得臺北市旅館的資訊，例如下圖就是此時的範例 App 執行在筆者的 S4 和 HTC Inredible S 手機中：

旅館	電話
友統旅館	0227735177
友萊大飯店	0225978800
天倫大旅社	0228813133
六福皇宮	0287706565
天閣酒店（台北南西）	0225679999
天成大飯店	0223617856
王朝大酒店	0227197199
丹迪旅店-天母店	0228732222
比堤飯店	0225066768
天津飯店	0225365678
丹迪旅店（大安店）	0227076899
友星大飯店	0223943121
月光莊旅社	0228914478

旅館	電話
友統旅館	0227735...
友萊大飯店	0225978...
天倫大旅社	0228813...
六福皇宮	0287706...
天閣酒店（台北南...	0225679...
天成大飯店	0223617...
王朝大酒店	0227197...
丹迪旅店-天母店	0228732...
比堤飯店	0225066...
天津飯店	0225365...

13-2 加入聲控查詢功能

能夠成功顯示所有旅館資訊之後當然我們會希望能查詢特定的旅館，為了讓此範例程式更有趣和有用，讓我們試著加入語言查詢的功能，讓這個範例 App 能夠允許使用者藉由說出旅館的部份名稱後就可以自動幫助使用者查詢旅館。由於 FireDAC 能夠使用部份字串查詢，因此在結合 Android 的語音功能後可以達成非常有意思的效果。

Google 在 Intenet 上提供了語音辨識的功能，因此我們只需要在範例 App 中啟動 Google 提供的這個功能就可以取得用戶說出的旅館名稱。由於這個語音辨識功能是由 Android 系統提供的而且在執行完畢之後會把執行結果回傳給我們的範例 App，因此我們必須能夠啟動我們想要的系統功能(也可啟動系統中其他的 App，稍後會說明)，而且我們必須能夠取得回傳結果。

RIO 中提供了 JActivity 介面允許程式師啟動系統功能或是啟動其他 App 以及其他多項功能：

```
JActivity = interface(JContextThemeWrapper)
```

由於現在我們需要啟動語音辨識功能並且取得回傳結果，因此我們可以使用 JActivity 介面中的 startActivityForResult 函式來完成這個工作：

```
  procedure startActivityForResult(intent: JIntent; requestCode:
Integer); cdecl; overload;
  procedure startActivityForResult(intent: JIntent; requestCode:
Integer; options: JBundle); cdecl; overload;
```

startActivityForResult 基本上接受 2 個參數，第 1 個參數 JIntent 是指程式師想啟動的功能或是 App(在 Android 中稱為 Activity)，第 2 個參數則指明程式師想啟動的呼叫，而 startActivityForResult 在執行完畢之後會藉由回叫原始程式取得回傳的執行結果：

版權所有 請勿翻印

```
onActivityResult(int, int, Intent)
```

但 onActivityResult 是 Java 方的程式碼，那我們如何要從 onActivityResult 中取得回傳的執行結果呢？

這裡我們需要解決 2 個問題：

1. 如何呼叫 startActivityForResult？

2. 如何從 onActivityResult 中取得回傳的執行結果

第 1 個問題很簡單，因為在 FMX.Platform.Android 程式單元中定義了 FireMonkey Android App 本身的 Activity : MainActivity：

```
function MainActivity: JFMXNativeActivity;
```

MainActivity 的型態是 JFMXNativeActivity， JFMXNativeActivity 介面是從 JNativeActivity 介面繼承下來，而 JNativeActivity 則是從 JActivity 介面繼承下來，因此我們可以使用 MainActivity 來呼叫 startActivityForResult。

第 2 個問題則可以使用 TMessageManager 類別來解決，TMessageManager 類別中的 SubscribeToMessage 方法可以讓我們訂閱接收 onActivityResult 的回傳結果。SubscribeToMessage 方法是 overload 方法：

```
  function SubscribeToMessage(const AMessageClass: TClass; const
AListener: TMessageListener): Integer; overload;
  function SubscribeToMessage(const AMessageClass: TClass; const
AListenerMethod: TMessageListenerMethod): Integer; overload;
```

這 2 個 overload 方法的差別是開發人員可以使用一般函式或是使用類別方法來接收回傳結果：

```
 TMessageListener = reference to procedure(const Sender: TObject; const
M: TMessage);
 TMessageListenerMethod = procedure (const Sender: TObject; const M:
TMessage) of object;
```

好了，原理說明到這裡已經差不多了，讓我們看看如何把這個功能實作出來。

要完成語音辨識功能，我們的範例 App 需要完成下面的工作：

1. 註冊 Delphi 程式碼中的回叫函式

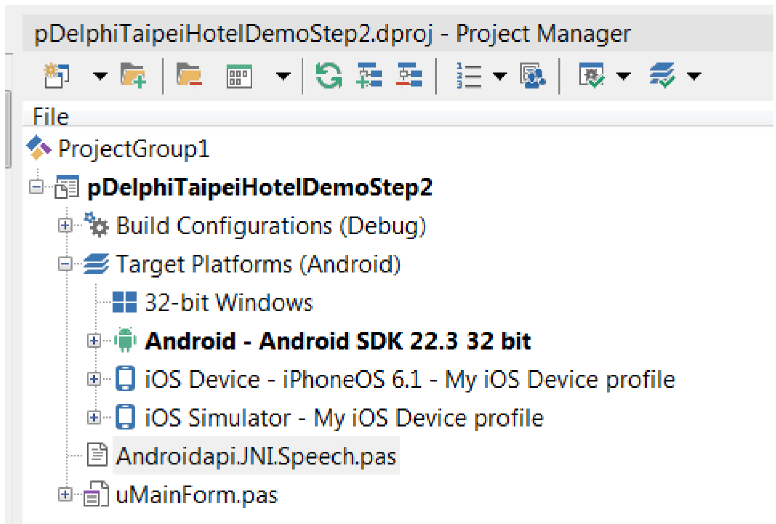
版權所有 請勿翻印

2. 藉由 JActivity 啟動 Google 的語音辨識功能

3. 從回叫函式中取出結果並且使用它進行搜尋

現在就讓我們完成這些工作，首先先在範例 App 中加入封裝 Google 語音辨識的程式單元 Androidapi.JNI.Speech.pas 到範例專案中：

> RIO 還沒有封裝 android.speech.tts 的 API，在未來的 Delphi 版本應該會內建 android.speech.tts 的封裝 API 程式單元



註冊 Delphi 程式碼中的回叫函式

首先在主表單的 OnCreate 事件處理函式中加入如下的程式碼，使用 TMessageManager 註冊 IntentCallback 函式成為處理稍後啟動 Google 語音辨識功能回傳結果的處理函式：

```
procedure TfmMainForm.FormCreate(Sender: TObject);
begin

TMessageManager.DefaultManager.SubscribeToMessage(TMessageResultNotif
ication, IntentCallback);
end;
```

當 IntentCallback 函式式呼叫時就是結果回傳的時候，它會接收到以 TMessageResultNotification 類別封裝的回傳結果：

```
TMessageResultNotification = class(TMessage<JIntent>)
public
  RequestCode: Integer;
  ResultCode: Integer;
end;
```

TMessageResultNotification 中的 RequestCode 欄位是我們要求的呼叫種類而 ResultCode 則是呼叫的結果，如果 ResultCode 是 RESULT_OK 就代表呼叫成功而且結果也成功的回傳到 TMessageResultNotification 物件中。

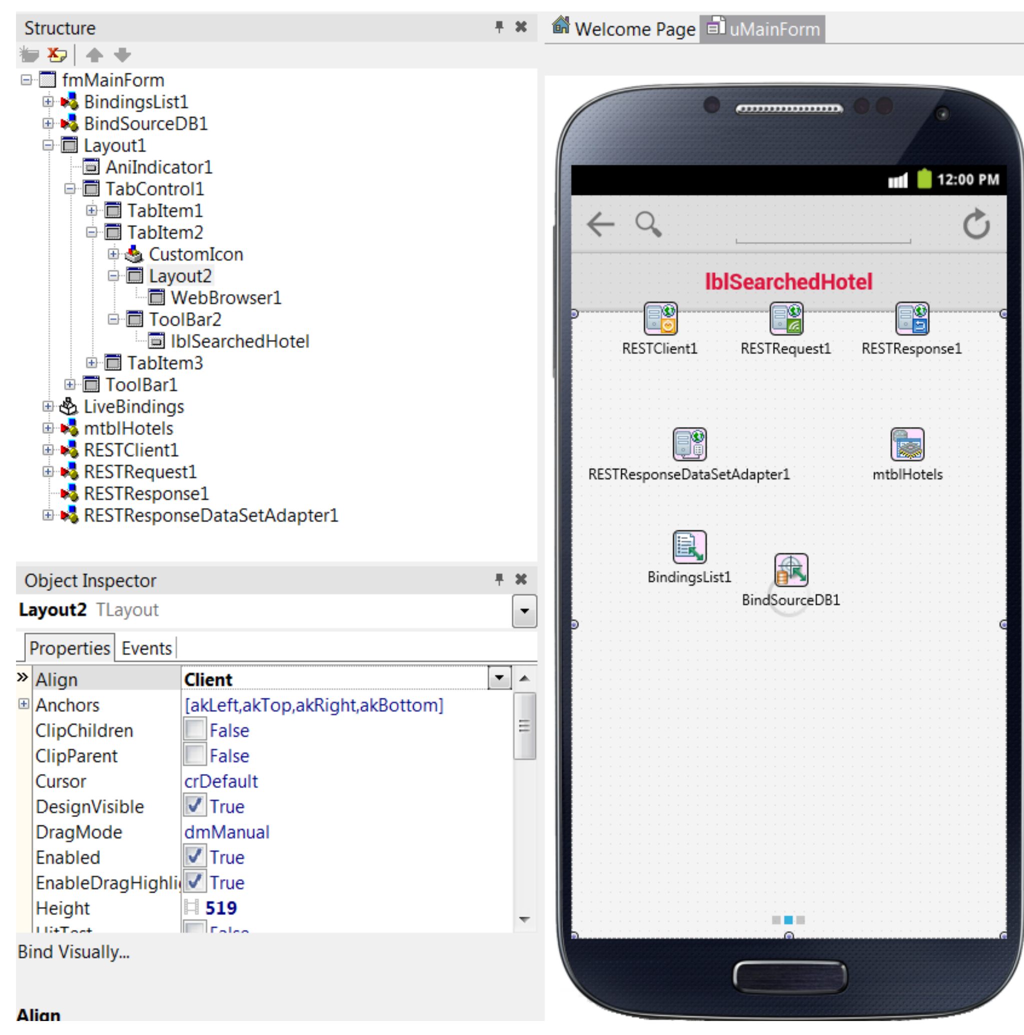
由於 TMessageResultNotification 是從 TMessage 泛型類別繼承下來，因此我們可以從 TMessage 的 Value 特性中最得回傳結果：

```
TMessage<T> = class (TMessage)
```

```
protected
  FValue: T;
public
  constructor Create(const AValue: T);
  destructor Destroy; override;
  property Value: T read FValue;
end;
```

藉由 JActivity 啟動 Google 的語音辨識功能

回到主表單，在 TabControl 的 TabItem2 中加入一個 TLayout:Layout2，在 Layout2 中加入一個 ToolBar，一個 TLabel 和一個 TWebBrowser，最後在主表單的 ToolBar 中加入一個代表搜尋的 TSpeedButton ,此時主表單如下所示:



在搜尋的 TSpeedButton 的 OnClick 事件處理函式中先呼叫 StartSpeechRecognizer 方法啟動 Google 語音辨識功能:

```
procedure TfmMainForm.sbtnSpeechSearchClick(Sender: TObject);
begin
  StartSpeechRecognizer;
end;
```

StartSpeechRecognizer 可回傳語音辨識結果的字串，也就是使用就說出的旅館名稱。StartSpeechRecognizer 先建立 JIntent 型態的物件 Recognizer，設定必要的參數之後就藉由 MainActivity 呼叫 startActivityForResult 方法並且傳入代表要啟動 Google 語音辨識的 ID：RecognizerRequestCode。

```
function TfmMainForm.StartSpeechRecognizer: String;
var
  Recognizer: JIntent;
begin
  Recognizer :=
TJIntent.JavaClass.init(TJRecognizerIntent.JavaClass.ACTION_RECOGNIZE
_SPEECH);

Recognizer.putExtra(TJRecognizerIntent.JavaClass.EXTRA_LANGUAGE_MODEL
,

TJRecognizerIntent.JavaClass.LANGUAGE_MODEL_FREE_FORM);
  Recognizer.putExtra(TJRecognizerIntent.JavaClass.EXTRA_PROMPT,
                    StringToJString('請說您的命令!'));
  Recognizer.putExtra(TJRecognizerIntent.JavaClass.EXTRA_MAX_RESULTS,
10); // default 5
  Recognizer.putExtra(TJRecognizerIntent.JavaClass.EXTRA_LANGUAGE,
                    StringToJString('zh-TW'));

  MainActivity.startActivityForResult(Recognizer,
RecognizerRequestCode);
end;
```

RecognizerRequestCode 的 ID 值是 1112：

```
const
  RecognizerRequestCode = 1112;
```

從回叫函式中取出結果並且使用它進行搜尋

當前面成功啟動 Google 語音辨識功能並且在用戶說出要查詢的旅館名稱之後便會回叫我們前面註冊的 IntentCallback 方法。因此在 IntentCallback 中我們先把第 2 個參數轉換型態為正確的 TMessageResultNotification 型態之後先判斷 ResultCode 是不是回傳成功執行，如果是的話就取出回傳結果並且讓 GetTextFromRecognizer 方法處理回傳的語音辨識結果：

```
procedure TfmMainForm.IntentCallback(const Sender: TObject; const M:
TMessage);
var
  Notification: TMessageResultNotification;
begin
  Notification := TMessageResultNotification(M);
  if Notification.ResultCode = TJActivity.JavaClass.RESULT_OK then
    GetTextFromRecognizer(Notification.Value);
end;
```

由於在前面實作 StartSpeechRecognizer 方法時我們使用了如下的程式碼：

版權所有 請勿翻印

```
  Recognizer.putExtra(TJRecognizerIntent.JavaClass.EXTRA_MAX_RESULTS,
10); // default 5
```

這代表當使用者說出要查詢的旅館名稱時 Google 的語音辨識功能最多會回傳 10 個最符合的辨識結果值，因此先取出所有的辨識結果值到 guesses 變數中，再把它們組合成一個結果字串回傳給呼叫函式，最後呼叫主表單中的 HandleVoiceSearch 方法使用辨識結果值來搜尋旅館：

```
function GetTextFromRecognizer(Intent: JIntent): string;
var
  guesses: JArrayList;
  guess: JObject;
  x: Integer;
begin
  guesses :=
intent.getStringArrayListExtra(TJRecognizerIntent.JavaClass.EXTRA_RES
ULTS);
  for x := 0 to guesses.Size - 1 do
  begin
    guess := guesses.get(x);
```

```
    result := Format('%s%s', [result, JStringToString(guess.toString)])
+ sLineBreak;
  end;
  fmMainForm.HandleVoiceSearch(Intent);
end;
```

HandleVoiceSearch 方法很簡單只是一一的取出每一個辨識結果值並且根據它來搜尋旅館：

```
procedure TfmMainForm.HandleVoiceSearch(Intent: JIntent);
var
  guesses: JArrayList;
  guess: JObject;
  x: Integer;
  sData : String;
begin
  guesses :=
intent.getStringArrayListExtra(TJRecognizerIntent.JavaClass.EXTRA_RES
ULTS);
  for x := 0 to guesses.Size - 1 do
  begin
    guess := guesses.get(x);
    sData := JStringToString(guess.toString);
    SearchHotel(sData);
  end;
end;
```

SearchHotel 使用主表單中的 TFDMemTable 在臺北市政府提供的旅館資訊中搜尋，如果搜尋成功就呼叫 DisplaySearchedHotel 顯示使用者要搜尋的旅館：

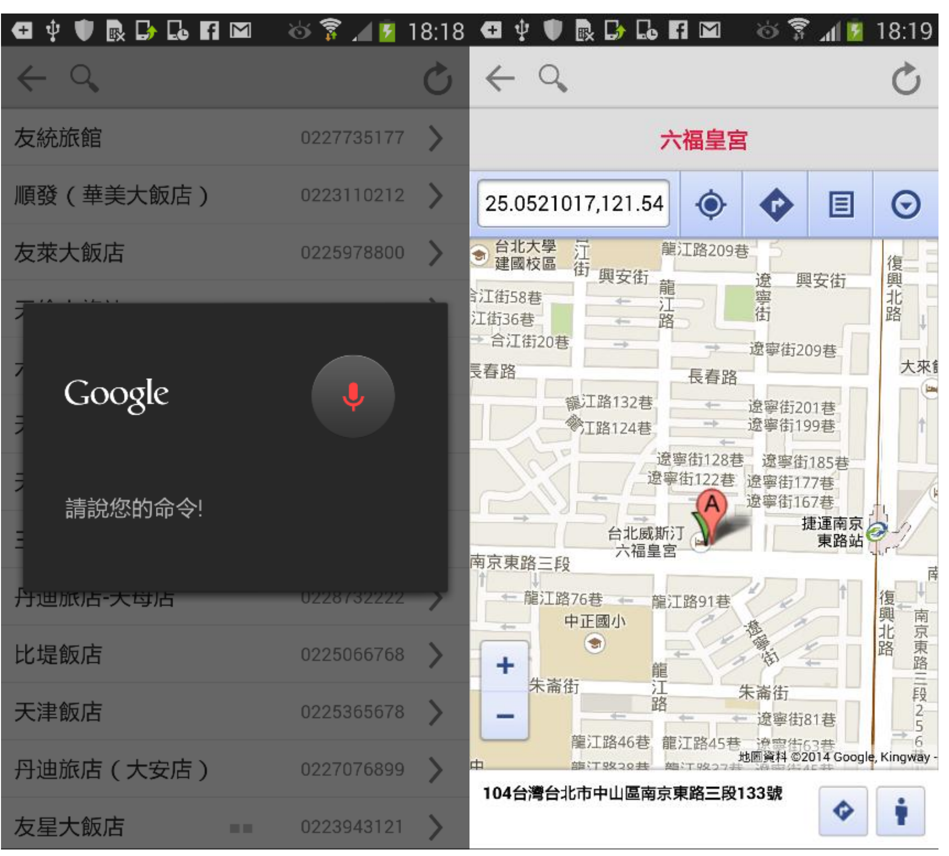
```
procedure TfmMainForm.SearchHotel(const sData: String);
begin
  if mtblHotels.LocateEx('stitle', sData, [lxoCaseInsensitive,
lxoPartialKey]) then
    DisplaySearchedHotel;
end;
```

DisplaySearchedHotel 方法從 TFDMemTable 元件中找到搜尋成功的旅館的經緯度值並且使用 TWebBrowser 元件定位搜尋的旅館：

```
procedure TfmMainForm.DisplaySearchedHotel;
```

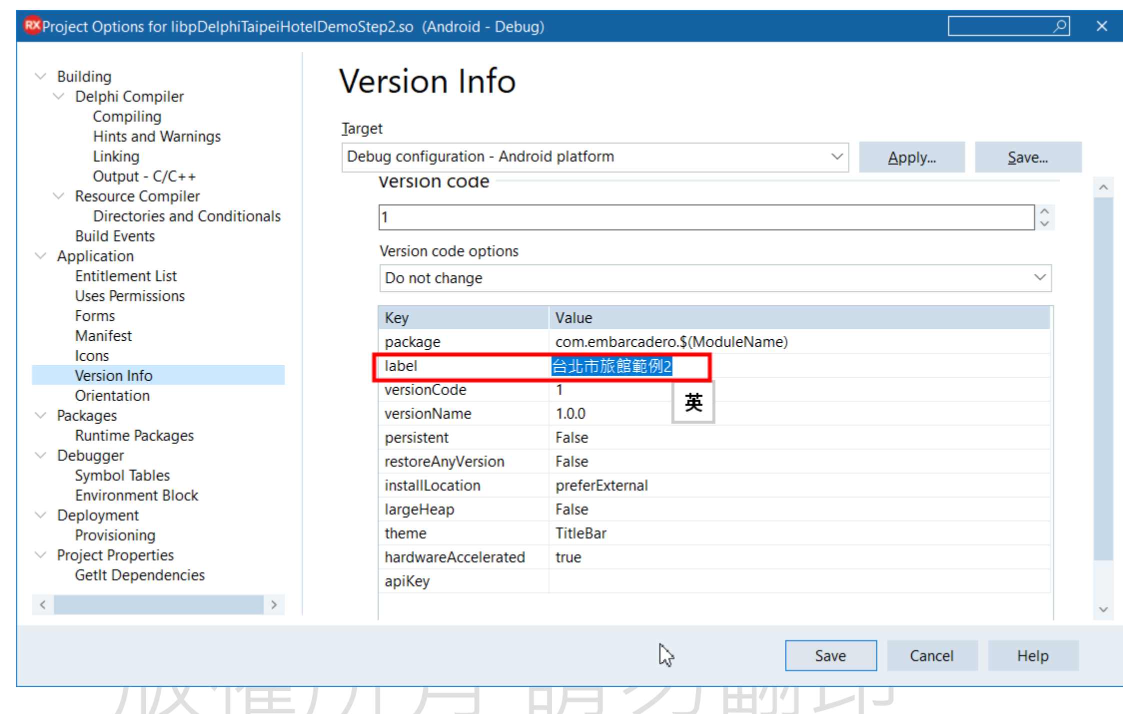
```
var
  iPos : Integer;
  sData : String;
  sHotelLongitude, sHotelLatitude : String;
  URLString : String;
begin
  lblSearchedHotel.Text := mtblHotels.FieldByName('stitle').Value;
  sHotelLongitude := mtblHotels.FieldByName('longitude').Value;
  sHotelLatitude := mtblHotels.FieldByName('latitude').Value;
  URLString := Format('https://maps.google.com/maps?q=%s,%s',
      [sHotelLatitude, sHotelLongitude]);
  WebBrowser1.Navigate(URLString);
  TabControl1.ActiveTab := TabItem2;
end;
```

現在請編譯此範例 App 並且執行它，從下圖中您可以看到筆者成功的使用範例 App 以說話的方式搜尋出了六福皇宮：

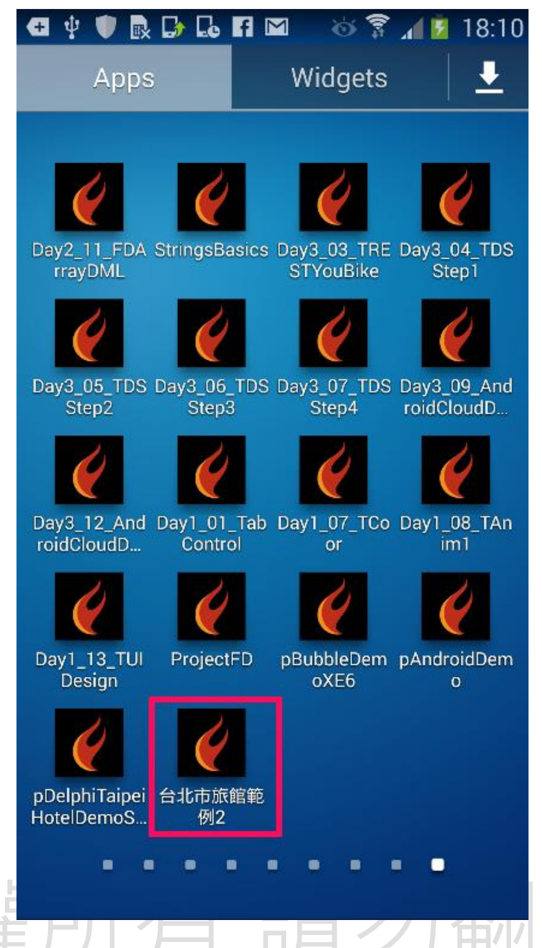


Cool！

到現在為止當我們部署範例 App 到手機中時看到的範例 App 名稱都是英文的項目名稱，現在讓我們為它設定中文的部署名稱吧。請在項目管理員中右擊專案開啟專案的 Options…對話盒，在 Version Info 選項中的 Label 選項中輸入您要設定的中文部署名稱，例如下圖使用了”臺北市旅館範例 2”:



現在您可再次部署此範例 App 到手機中，您應該就可以看到中文的 App 名稱了，例如下圖就是此範例 App 部署到 S4 手機中的結果畫面：

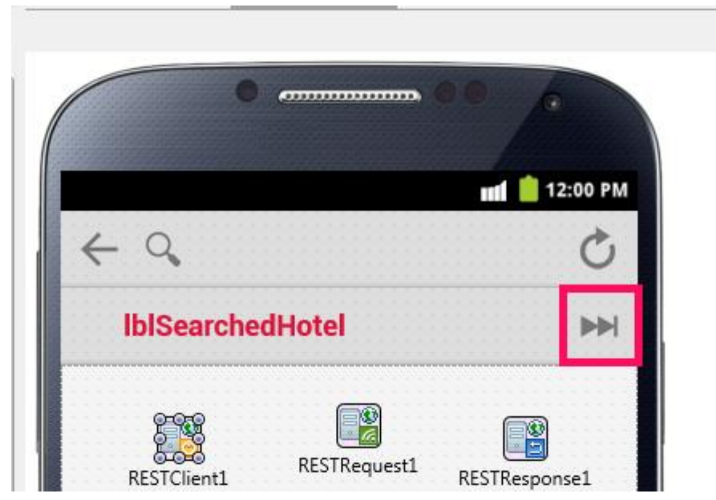


版權所有 請勿翻印

13-3 啟動流覽器查詢旅館網站主頁

現在讓我們再說明如何使用 Delphi RIO 來啟動其他的 Android App。在前面臺北市政府提供的旅館資訊中有一個 xurl 的欄位是每個旅館的網頁主頁位置，現在就讓我們使用 Android 的內建流覽器讓使用在搜尋到目標旅館之後能夠更進一步的流覽這個搜尋旅館的更多詳細資訊。

回到主表單中的 TabItem2 頁面並且在其中加入一個新的 TSpeedButton，設定它的 Name 特性值為 sbtnGoWebSite，如下所示：



在它的 **OnClick** 事件處理函式中呼叫 **NavigateToHotelWebSite** 方法流覽到旅館的主頁並且傳入旅館的 **xurl** 域值做為參數：

```
procedure TfmMainForm.sbtnGoWebSiteClick(Sender: TObject);
begin
  NavigateToHotelWebSite(mtblHotels.FieldByName('xurl').Value);
end;
```

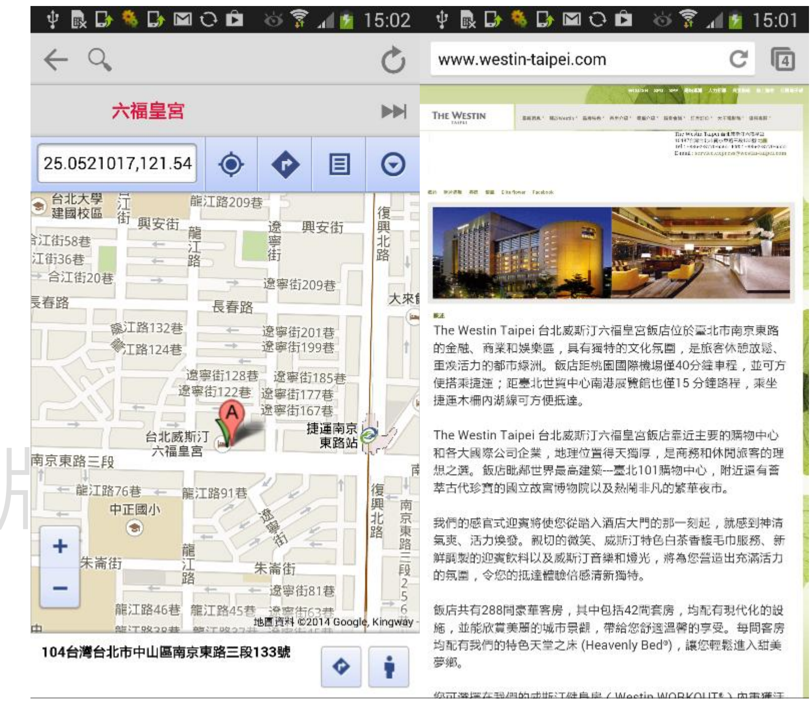
版權所有 請勿翻印

NavigateToHotelWebSite 方法在 **007~009** 行檢查需不需要為旅館主頁 **URL** 加入 **http/https** 通訊協定，接著在 **011** 行建立流覽 **JIntent** 物件然後在 **014** 行藉由 **MainActivity** 呼叫 **startActivity** 方法並且傳入 **browserIntent** 做為參數，如此一來就可以要求 **Android** 啟動流覽器並且流覽到旅館的主頁了：

```
001    procedure TfmMainForm.NavigateToHotelWebSite(const sWebSite:
String);
002    var
003      sURL : String;
004      browserIntent: JIntent;
005    begin
006      sURL := sWebSite;
007      if ( (Not sURL.startsWith('http://')) And
008          (Not sURL.startsWith('https://')) ) then
009        sURL := 'http://' + sURL;
010
011      browserIntent :=
TJIntent.JavaClass.init(TJIntent.JavaClass.ACTION_VIEW,
012        TJnet_Uri.JavaClass.parse(StringToJString(sURL)));
013      try
```

```
014        MainActivity.startActivity(browserIntent);
015      except
016        on e: Exception do
017          ShowMessage(e.Message);
018      end;
019    end;
```

編譯，部署和再次執行範例 App，使用語言搜尋旅館之後再點選右上方的按鈕就可以流覽到旅館主頁了，例如下圖就是範例 App 流覽到六福皇宮的主頁：



13-4 在雲端為旅館寫評價

也許現在是再展示如何結合移動和雲端應用的好時機，讓我們再為此範例 App 加入在雲端為旅館寫點評價的功能，本小節將說明如何在 Android App 中使用 Microsoft 的 Azure 雲端服務。我們將使用 Azure 的記憶體服務來儲存旅館評價，當然在您閱讀完本小節內容之後您可以使用您選擇的雲端提供者和服務。

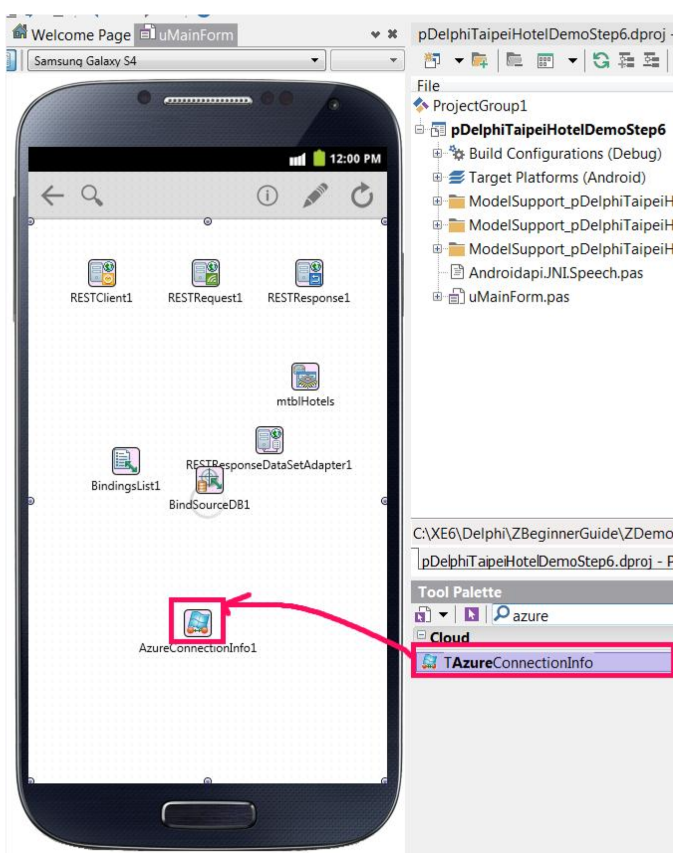
首先您需要申請一個 Azure 帳號，您可以在下面的 URL 申請：

```
http://azure.microsoft.com/zh-tw/
```

例如筆者申請了一個試用的 Azure 帳號：

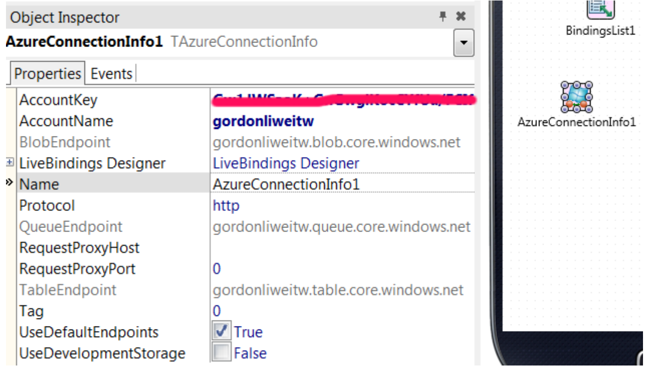
然後點選” 管理儲存金鑰” 按鈕取得您的帳戶名稱和金鑰：

現在回到範例 App，在主表單中加入一個 TAzureConnectionInfo 元件：

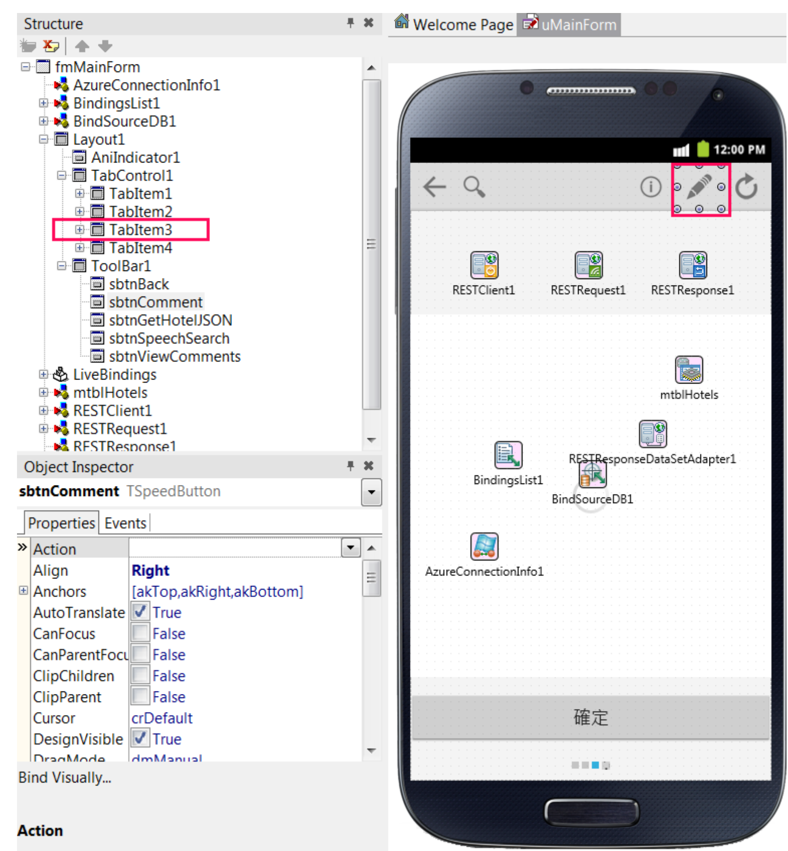


版權所有 請勿翻印

然後在物件檢視器中設定 AccountName 和 AccountKey 特性值為前面 Azure 的帳戶名稱和金鑰，如下所示：



接著在主表單的 TabControl 的 TabItem3 中加入一個可寫評價的 TMemo 元件和一個"確定"按鈕，再于上方的 ToolBar 中加入一個寫評價的 TSpeedButton：sbtnComment，如下所示：



版權所有 請勿翻印

在 sbtnComment 的 OnClick 事件處理函式中撰寫如下的程式碼顯示 TabItem3 準備讓使用者開始寫評論：

```
procedure TfmMainForm.sbtnCommentClick(Sender: TObject);
begin
  mmHotel.Lines.Clear;
  TabControl1.TabIndex := 2;
  mmHotel.SetFocus;
end;
```

在 TabItem3 頁面的”確定”按鈕的 OnClick 事件處理函式中呼叫 SaveDataToAzure 方法儲存評論到雲端：

```
procedure TfmMainForm.SpeedButton1Click(Sender: TObject);
begin
  SaveDataToAzure;
end;
```

SaveDataToAzure 方法是真正把評論資料儲存到 Azure 的函式，在 014 行先呼叫 CreateTableService 方法在 Azure 的記憶體中建立一個範例資料表以儲存旅館評論，016~017 行為每一筆評論資料建立一個唯一的 GUID 值，018 行在記憶體範例資料表建立一筆新的資料列，020~024 把評論資料儲存到資料

列的欄位中，026 行呼叫 InsertEntity 把評論資料資料列真正的儲存到 Azure 的記憶體中，最後記得要釋放資料列和記憶體物件：

```
001   const
002     XML_ROWKEY = 'RowKey';
003     XML_PARTITION = 'PartitionKey';
004     HOTELTABLENAME = 'HOTELCOMMENTSTBL';
005
006   procedure TfmMainForm.SaveDataToAzure;
007   var
008     RowObj : TCloudTableRow;
009     content : TBytes;
010     meta : TStringList;
011     aGUID : TGUID;
012     sGUID : String;
013   begin
014     CreateTableService;
015     try
016       CreateGUID(aGUID);
017       sGUID := GuidToString(aGUID);
018       RowObj := TCloudTableRow.Create;
019       try
020         RowObj.SetColumn(XML_ROWKEY,
mtblHotels.FieldByName('serial_no').AsString );
021         RowObj.SetColumn(XML_PARTITION, sGUID);
022         RowObj.SetColumn('HOTELNAME',
mtblHotels.FieldByName('stitle').AsString);
023         RowObj.SetColumn('COMMENT', mmHotel.Lines.Text);
024         RowObj.SetColumn('COMMENTDT', DateTimeToStr(Now));
025
026         TableService.InsertEntity(HOTELTABLENAME, RowObj);
027       finally
028         RowObj.Free;
029       end;
030     finally
031       DeleteTableService;
032     end;
033   end;
```

CreateTableService 方法在 008 行先建立 TAzureTableService 的物件，接著呼叫 QueryTables 查詢範例資料表是否已存在，如果沒有的話就先建立範例資料表：

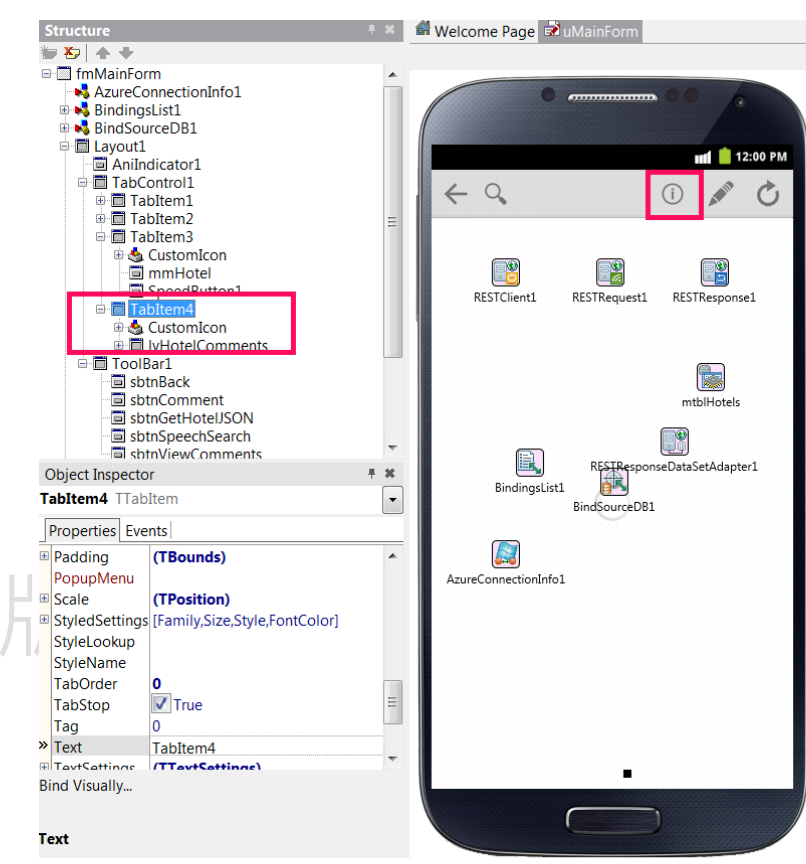
```
001   procedure TfmMainForm.CreateTableService;
002   var
003     tbls : TStringList;
004     iIndex: Integer;
005     sTable : String;
006     bCreate : Boolean;
007   begin
008     TableService :=
TAzureTableService.Create(AzureConnectionInfo1);
009     tbls := TableService.QueryTables('', nil) as TStringList;
010     bCreate := true;
011     try
012       for iIndex := 0 to tbls.Count - 1 do
013       begin
014         sTable := tbls.Strings[iIndex];
015         if (sTable = HOTELTABLENAME) then
016         begin
017           bCreate := false;
018           break;
019         end;
020       end;
021
022       if (bCreate) then
023         TableService.CreateTable(HOTELTABLENAME);
024     finally
025       tbls.Free;
026     end;
027   end;
```

至於 DeleteTableService 則是釋放 TableService 對象：

```
procedure TfmMainForm.DeleteTableService;
begin
  TableService.Free;
end;
```

完成了上面的程式碼之後旅館評論資料應該就可以成功儲存到 Azure 了，接下來再讓我們實作從 Azure 查詢所有旅館評論資料的功能。

在主表單的 TabControl 中加入 TabItem4 並且在其中加入一個名為 lvHotelComments 的 TListView 元件，再于上方工具列加入一個查詢旅館評論資料的 TSpeedButton：sbtnViewComments：



在 sbtnViewCommentsOnClick 事件處理函式中呼叫 GetDataFromCloud 方法從 Azure 的記憶體中取得所有旅館評論資料:

```
procedure TfmMainForm.sbtnViewCommentsClick(Sender: TObject);
begin
  GetDataFromCloud;
end;
```

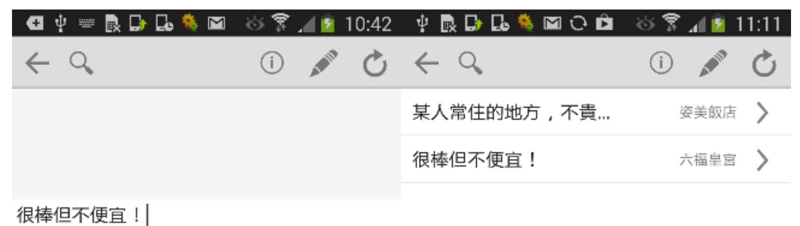
GetDataFromCloud 方法在 007 行使用 TableService 查詢 Azure 中的範例資料表並且儲存在 rowList，接著在 010~015 行把其中的資料一一的取出並且顯示在 lvHotelComments 元件中：

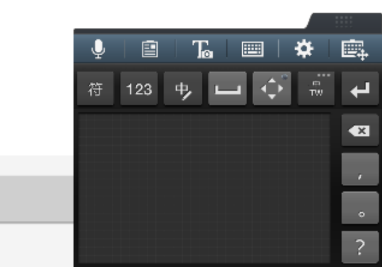
```
001    procedure TfmMainForm.GetDataFromCloud;
002    var
003      rowList : TList<TCloudTableRow>;
```

```
004      aRow : TCloudTableRow;
005      lvi : TListViewItem;
006    begin
007      rowList := TableService.QueryEntities(HOTELTABLENAME);
008      try
009        lvHotelComments.Items.Clear;
010        for aRow in rowList do
011        begin
012          lvi := lvHotelComments.Items.Add;
013          lvi.Detail := aRow.GetColumn('HOTELNAME').Value;
014          lvi.Text := aRow.GetColumn('COMMENT').Value;
015        end;
016      finally
017        rowList.Free;
018      end;
019    end;
```

現在編譯並且執行範例 App，試著先搜尋一些旅館再為它們寫一些範例評價並且儲存在 Azure 中，最後點選按鈕 ⓘ 查詢 Azure 雲端中所有的旅館評論就可以看到類似下面的執行結果畫面，範例 App 的確可提供 Azure 的雲端服務了。

版權所有 請勿翻印





手機 App 結合雲端的確可以提供非常方便且有趣的應用。

13-5 寫個可啟動所有範例 App 的 App 吧

到這裡為止我們已經撰寫了數個範例 App 了，最後讓我們再說明在 Android 系統中如何啟動其他的 App，例如現在就讓我們撰寫一個範例 App 讓它能夠啟動執行前面開發的範例 App。

先建立一個新的 FireMonkey Android 專案並且在主表單中放入數個 TSpeedButton 分別啟動不同的範例 App，例如下圖是本範例 App 的主表單：

版權 翻印

接下來我們要知道如何在 Android 系統中啟動其他的 App，事實上您只需要知道 2 件事，

1. 您要啟動 App 的 Package 名稱，以及
2. 您要啟動 App 的 Class 名稱

對於 Delphi 建立的 Android App 來說這 2 個資訊就在您的 App 專案的 Android 目錄下的 AndroidManifest.xml 之中，例如在前面 pDelphiTaipeiHotelDemoStep6 範例的 AndroidManifest.xml 中我們可以在其中的 Package 特性中找到此 App 的 Package 名稱並且在 Activity 特性中找到此 App 的 Class 名稱：

Welcome Page | uMainForm | AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- BEGIN_INCLUDE(manifest) -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.embarcadero.pDelphiTaipeiHotelDemoStep6"
        android:versionCode="1"
        ...

       <activity android:name="com.embarcadero.firemonkey.FMXNativeActivity"
                 android:label="台北市旅館範例6"
       ...
</manifest>
<!-- END_INCLUDE(manifest) -->
```

一般來說如果您沒有更改 Delphi Android App 的內建 Package 和 Class 名稱，那麼您的 App 的 Package 名稱一定是：

com.embarcadero. ＋ 您的專案名稱

例如前面最後一個臺北市旅館範例 App 的 Package 名稱就是

com.embarcadero.pDelphiTaipeiHotelDemoStep6

而 Class 名稱一定是：

'com.embarcadero.firemonkey.FMXNativeActivity'

版權所有 請勿翻印

瞭解了如何取得欲啟動的 App 的 Package 和 Class 名稱之後我們就可以開始實作這個範例了。

首先在範例中定義 2 個字串常數：

```
const
  AppPACKAGENAME = 'com.embarcadero.';
  AppCLASSNAME = 'com.embarcadero.firemonkey.FMXNativeActivity';
```

接著在主表單中的第 1 個 TSpeedButton 的 OnClick 事件處理及式中撰寫如下的程式碼：

```
procedure TfmMainForm.SpeedButton1Click(Sender: TObject);
var
  sPackageName : String;
begin
  sPackageName := AppPACKAGENAME + 'pDelphiTaipeiHotelDemoStep1';
  ActivateAndroidApp(sPackageName);
end;
```

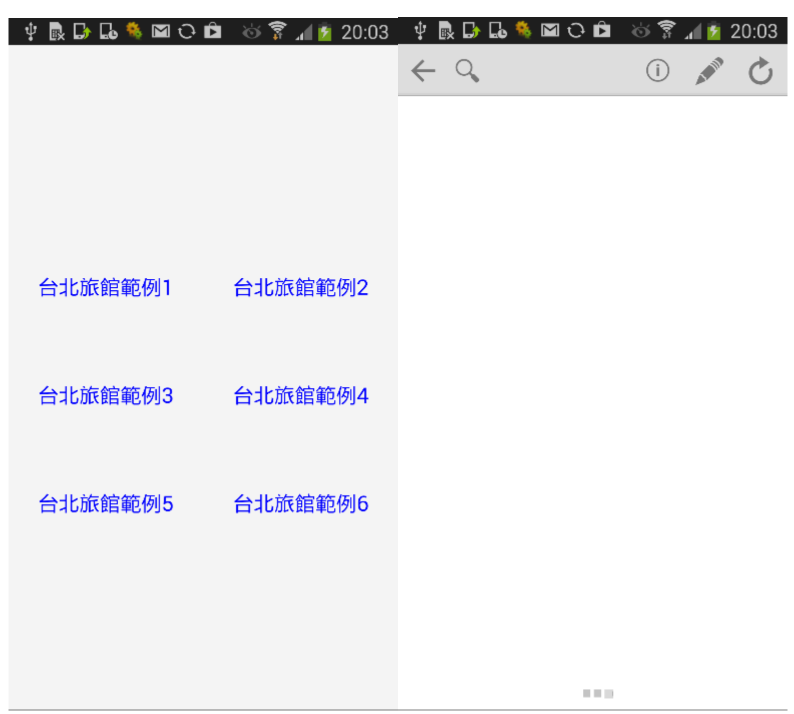
由於 Delphi Android App 的 Class 名稱是固定的，因此在上面的程式碼中只需要建立要啟動的 App 的 Package 名稱，因此上面的程式碼是要啟動前面的第 1 個臺北市旅館範例 App，最後呼叫 ActivateAndroidApp 方法來真正的啟動目標 App。

ActivateAndroidApp 方法同樣使用 JIntent 物件來啟動目標 App，005 行先建立要啟動 Android App 的 JIntent 物件，006 行再設定要啟動 Android App 的 Package 名稱和 Class 名稱，請注意由於 Package 名稱和 Class 名稱是字串內容，因此我們必需先呼叫 StringToJString 把 Delphi 的字串轉換為 Java 的字串型態再傳遞給 Android 系統：

```
001   procedure TfmMainForm.ActivateAndroidApp(const sPKG: String);
002   var
003     AppIntent: JIntent;
004   begin
005     AppIntent :=
TJIntent.JavaClass.init(TJIntent.JavaClass.ACTION_MAIN);
006
AppIntent.setComponent(TJComponentName.JavaClass.init( StringToJStrin
g(sPKG), StringToJString(AppCLASSNAME)));
007     MainActivity.startActivity(AppIntent);
008   end;
```

最後 007 行同樣藉由 MainActivity 呼叫它的 startActivity 方法來啟動目標 App。

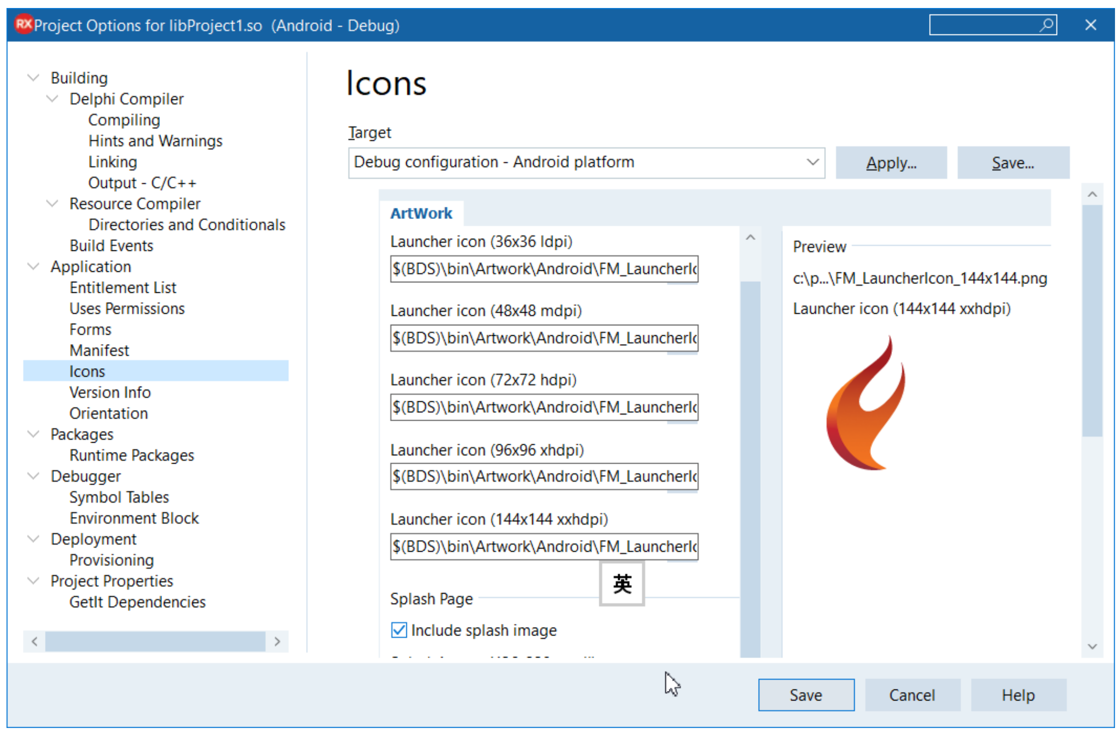
好了，您可以繼續使用類似的技術和程式碼在主表單中其他的 TSpeedButton 中啟動不同的範例 App。下圖就是本範例 App 在 S4 中執行然後點選”臺北旅館範例 6”按鈕成功啟動前面最後一個臺北市旅館範例 App：



13-6 為您的 App 設計個圖像吧

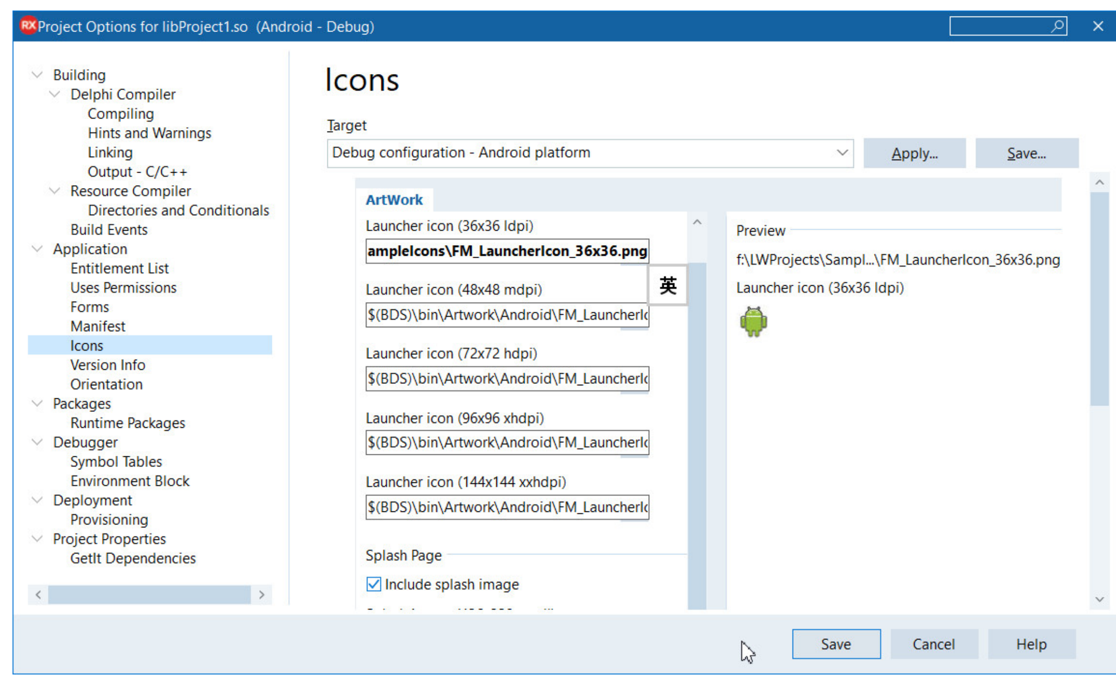
當您使用 Delphi 開發完 App 之後需要部署到手機中，那麼您一定希望能使用客制化的圖像來代表您的 App 而不是使用 Delphi 內定的圖像。在 RIO 中要為 App 更改客制化圖像非常的簡單，請點選 Project|Options 選單，在顯示的對話盒 Application 專案中可以看到 Delphi 內定使用的各種大小的 App 圖像，如下所示：

版權所有 請勿翻印



您可以設計同樣大小的圖像來使用，例如 36x36，48x48 等圖像，然後點選每個圖像旁的...按鈕並且載入使用您的客制化圖像。例如下圖就是筆者在此對話盒 Application 專案中加載客制化圖像：

版權所有 請勿翻印



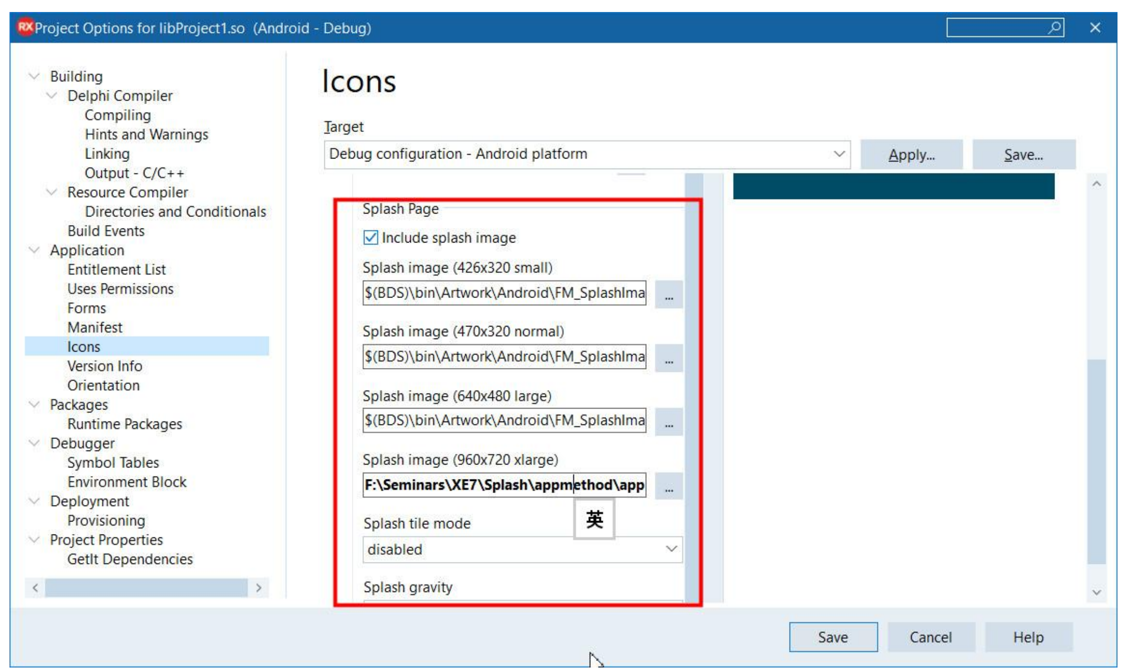
接著重新編譯並且部署您的 App，您就可以在手機中看到您的 App 現在使用了您自己的客制化圖像：

版權 翻印

13-7 為您的 App 加入啟動畫面吧

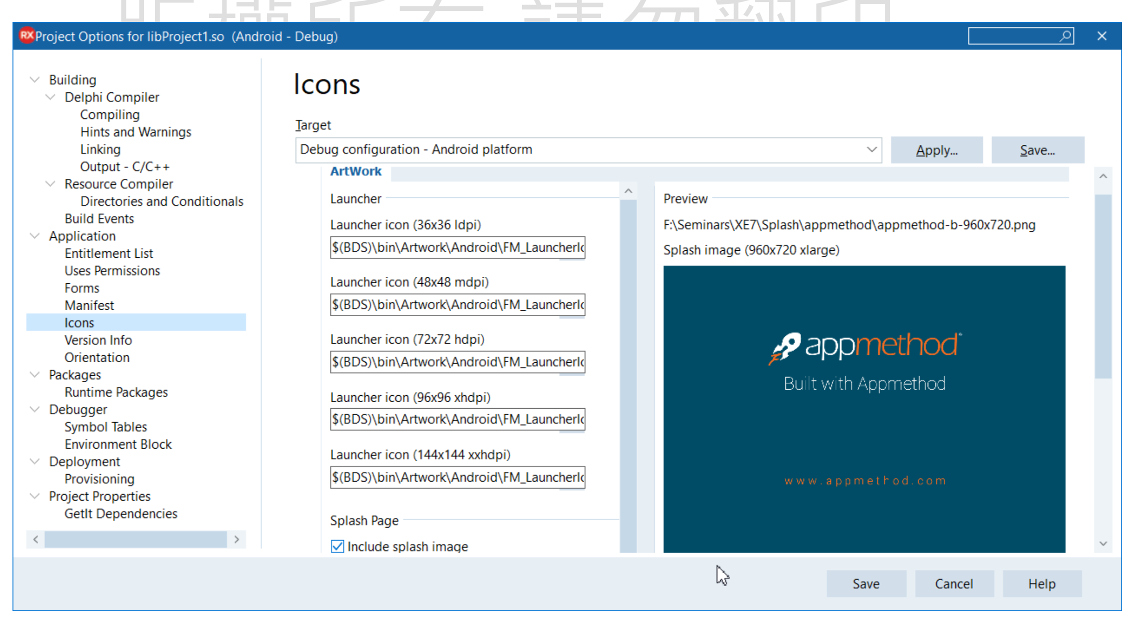
除了 App 圖像之外，RIO 也允許開發人員為 App 加入啟動時的歡迎畫面，由於 RIO 編譯出的 App 是原生的 App，App 體積比較大一點因此需要多一點的載入時間，為了讓您的 App 在一開始執行時有比較快的反應時間，您可以為 App 加入歡迎畫面。

您同樣可以藉由點選 Project | Options 選單，在剛才加入客制化圖像的下方可以看到 Splash Image 的選項：



同樣的您可以自行設計您的啟動畫面影像，然後點選每個圖像旁的...按鈕並且載入使用您的客制化影像即可，例如下圖就是加載客制化影像並且設定使用 fill_vertical 方式顯示：

版權所有 請勿翻印



部署此 App 並且執行就可以看到如下的啟動畫面：

13-8 為您的 App 加入 Provisioning 資訊吧

版權所有 請勿翻印

當您開發完並且決定部署您的 App 並且如上加入了客制化圖像和啟動畫面影像後，請記得再對 App 進行部署開通服務(Provisioning)的設定，一旦完成了部署開通服務您的 App 不但能夠部署到非開始模式的手機，也可以部署到 Application Store 中。

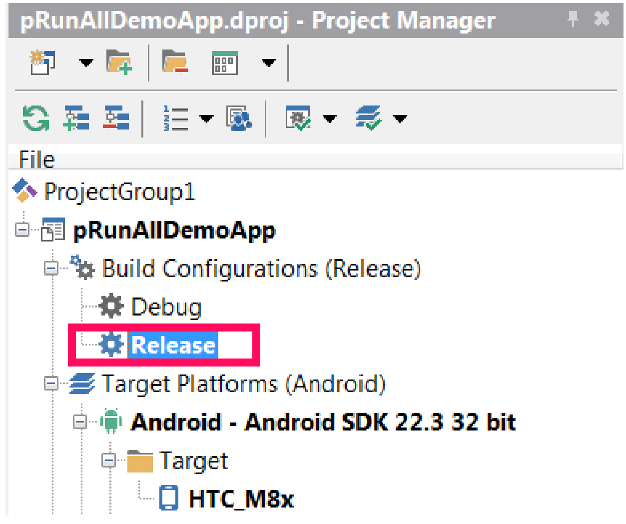
要對 App 進行 部署開通服務，您需要完成下列的工作：

1. 設定 Build Configurations 為 Release
2. 開啟 Target Platforms 設定為 Android 平臺並在 Configuration 中選擇 Application Store
3. 到 Project > Options > Provisioning 頁面填寫必要的資訊
4. 維護 App 版本資訊

下面進行詳細的說明。

設定 Build Configurations 為 Release

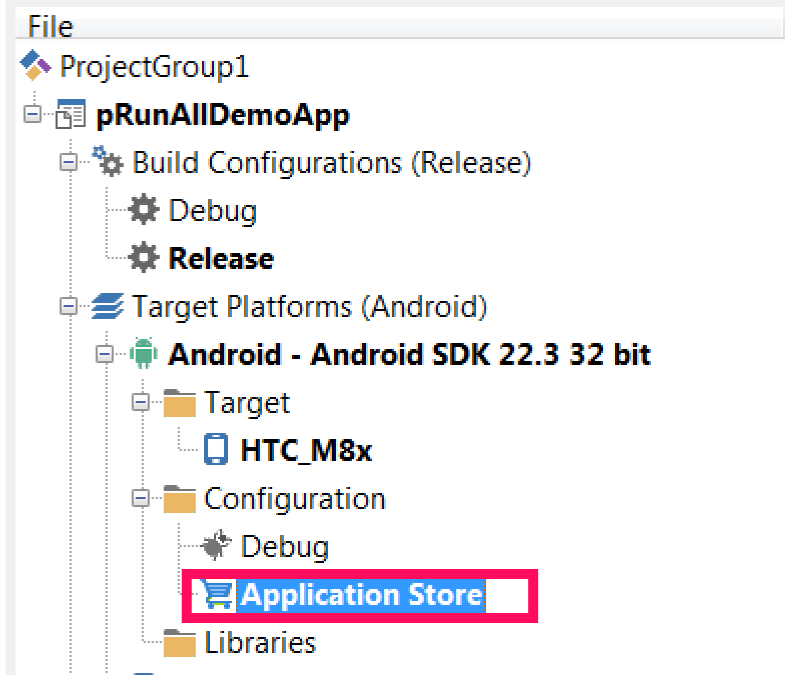
我們在 IDE 中開發 App 時都是使用 Debug 模式，但在實際部署時一定要改為 Release 模式重新編譯一次再部署。要設定為 Release 模式，請在專案管理員中按兩下 Build Configurations 中的 Release 節點：



開啟 Target Platforms 設定為 Android 平臺並在 Configuration 中選擇 Application Store

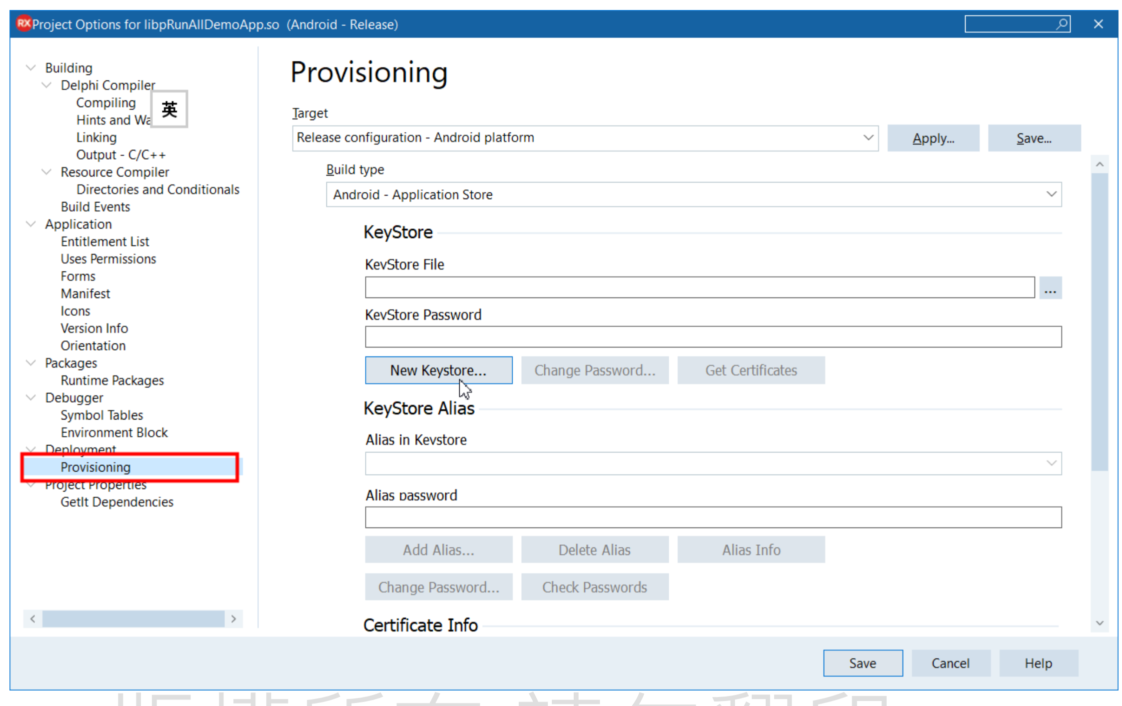
版權所有 請勿翻印

接著同樣在請在項目管理員中按兩下 Target Platforms | Configuration 中的 Application Store 節點:



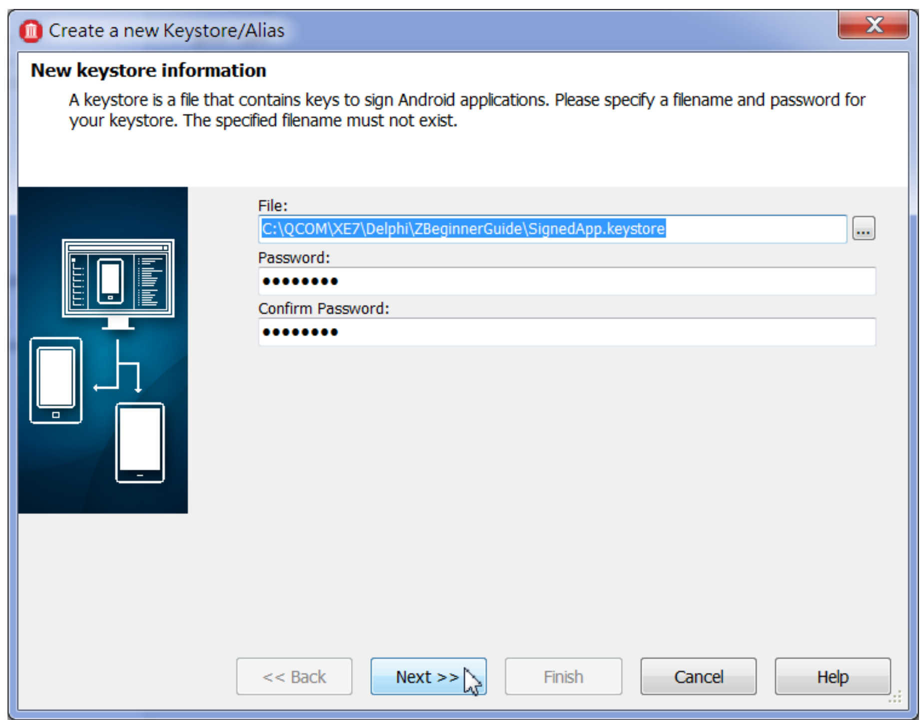
到 Project > Options > Provisioning 頁面填寫必要的資訊

點選 Project | Options 選單，再點選 Provisioning 節點：

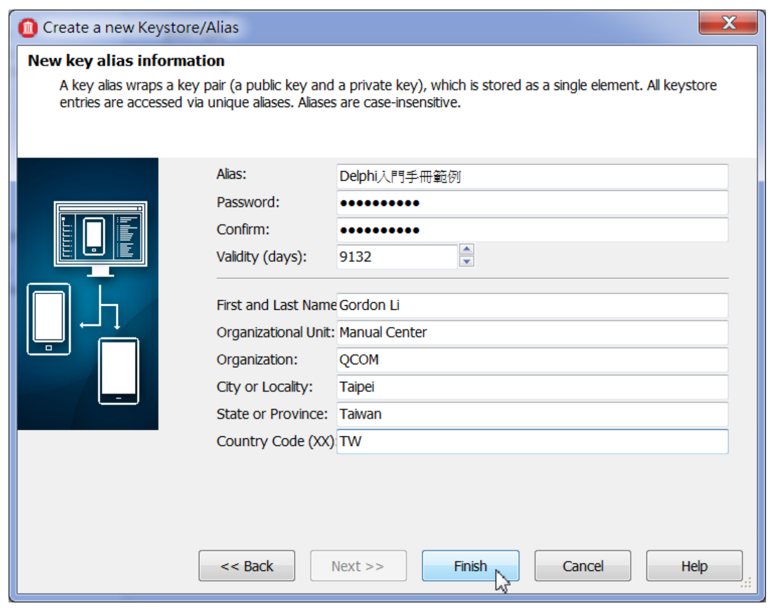


版權所有 請勿翻印

接著點選其中的 New Keystore...按鈕，選擇一個要建立 Keystore 檔的目錄和檔案名並且輸入密碼：

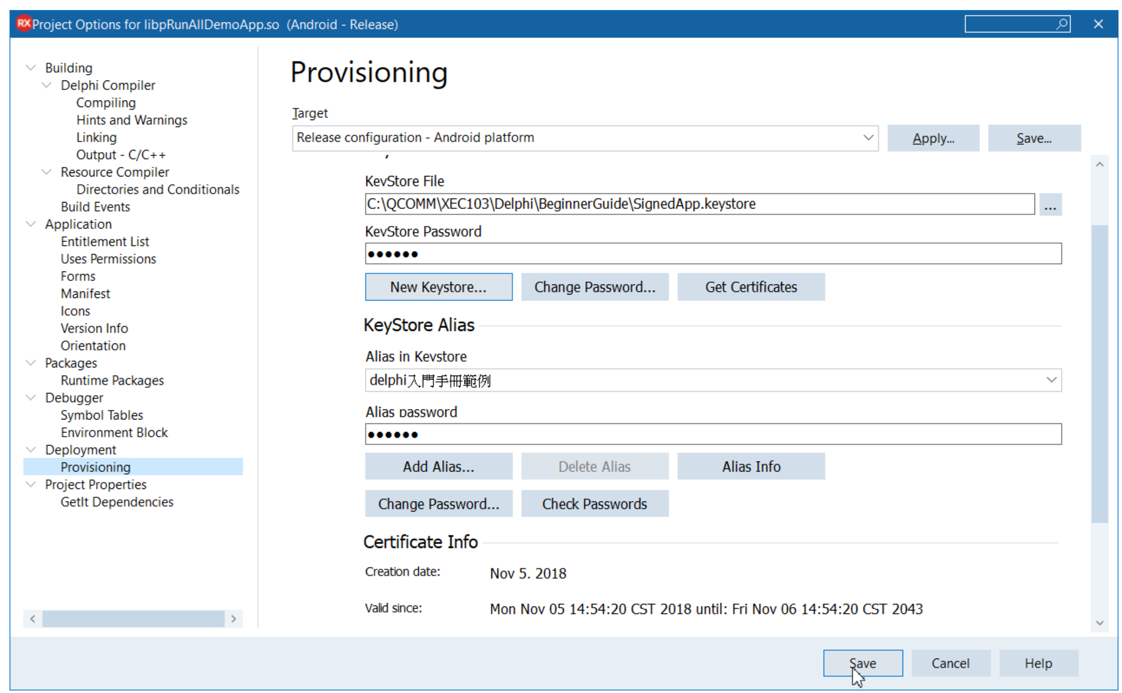


點選 Next 按鈕，在下一個對話下輸入他的別名(Alias)，密碼等資訊，如下所示：



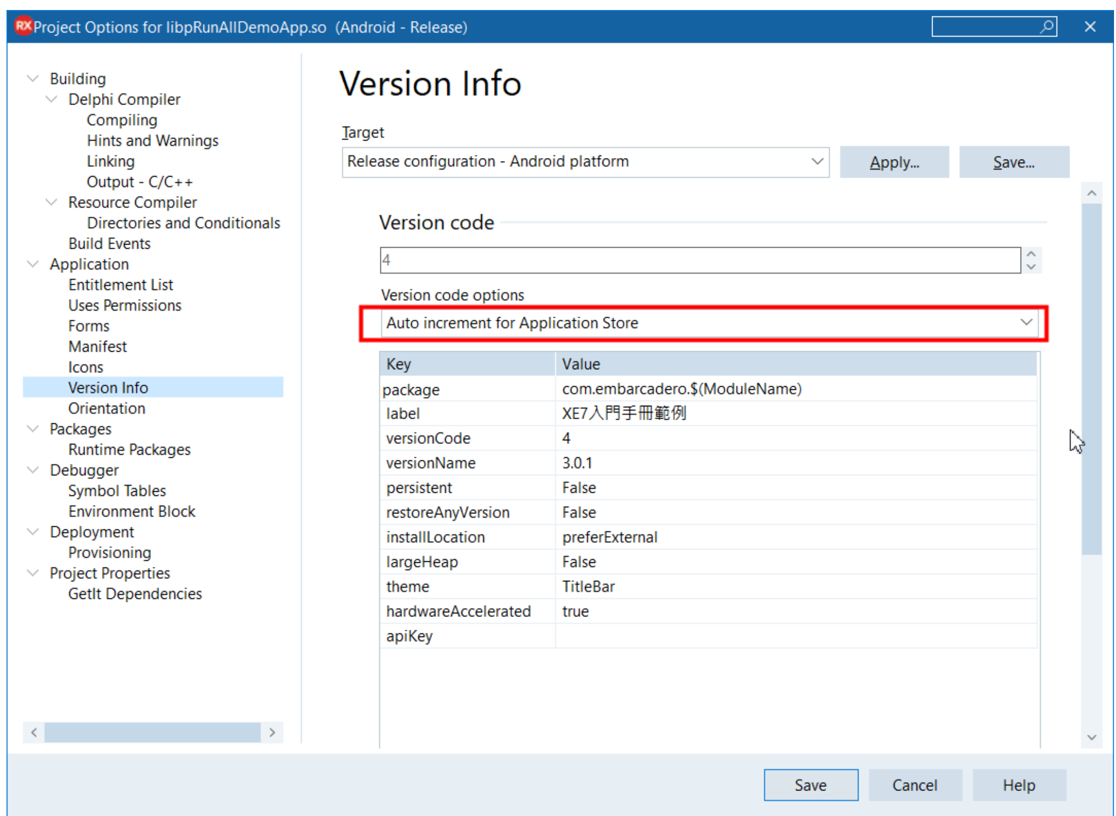
點選 Finish 按鈕之後就可以回到 Provisioning 節點並且看到剛才設定的資訊都已經自動帶入到 Provisioning 的各個欄位中了。剛才建立的 Keystore 檔是這個 App 的 Provisioning 資訊，請像 App 的原始程式一樣妥善保存好。

版權所有 請勿翻印

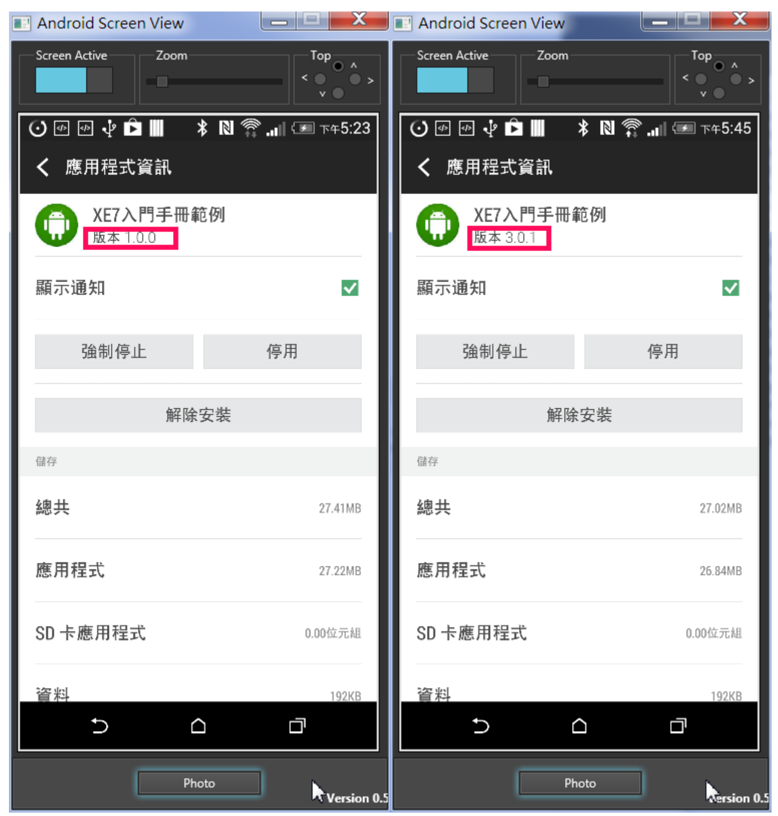


維護 App 版本資訊

最後請設定你的 App 版本資訊，您可以點選 Project|Options 選單，再點選 Version Info 節點中找到您的 App 版本資訊。要讓 IDE 自動維護您的 App 版本資訊,請選擇 Auto increment for Application Store 選項：



那麼您每一次重新 Build 您的項目 Version code 就會自動增加，當然您也可以使用 versionName 特性來顯示您的 App 的版本資訊，例如更改 versionName 特性值和您的 Version code 一致後就可以在手機中看到您的 App 的版本資訊了：



14 RIO 新功能

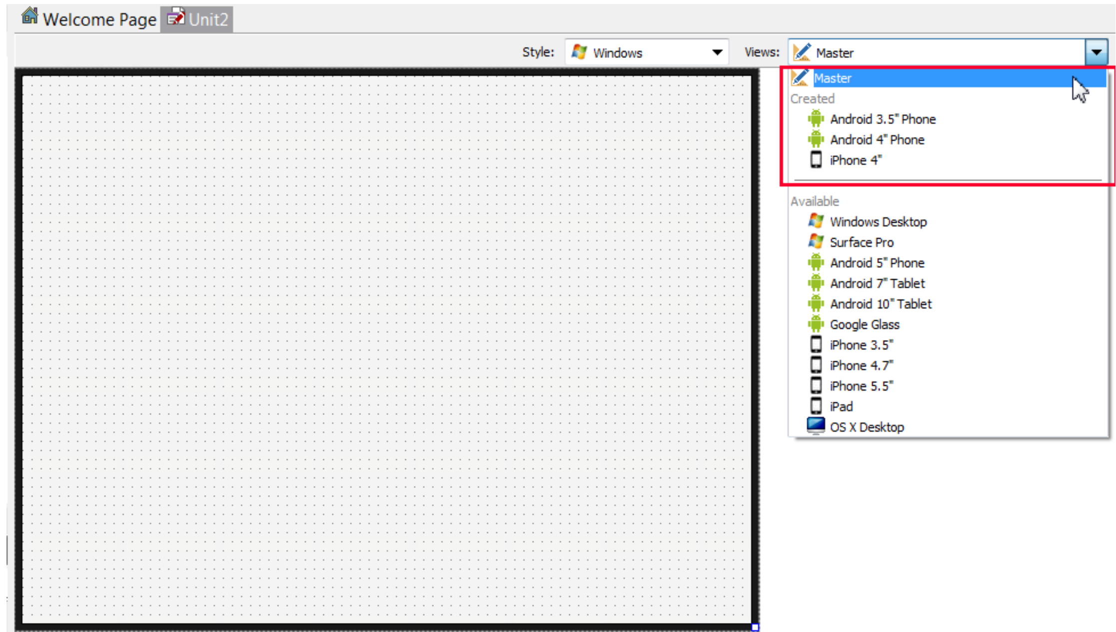
版權所有 請勿翻印

Delphi RIO 中的新功能：MultiView 設計家，版本控制功能 Git 和 DUNITX 測試框架是筆者認為非常重要的 3 大功能。這是因為 Delphi 支持多平臺的開發能力，但如何在多個平臺開發時能夠減少 UI 的設計工作，如何能夠最大化的使用相同的程式碼便成為了開發人員最須重視的問題。而 MultiView，Git 和 DUNITX 正是這 2 個問題的解決方案，因此在本章中將先為讀者介紹這 3 個功能。

14-1 MultiView

MultiView 功能是從 XE7 版本開始出現的，它的目標是讓開發人員節省多平臺開發的時間進而提升生產力。在 RIO 中 MultiView 繼續獲得了強化，RIO 加入了 Multi-Device 功能允許開發人員可同時檢視所有開發平臺的 UI 介面。

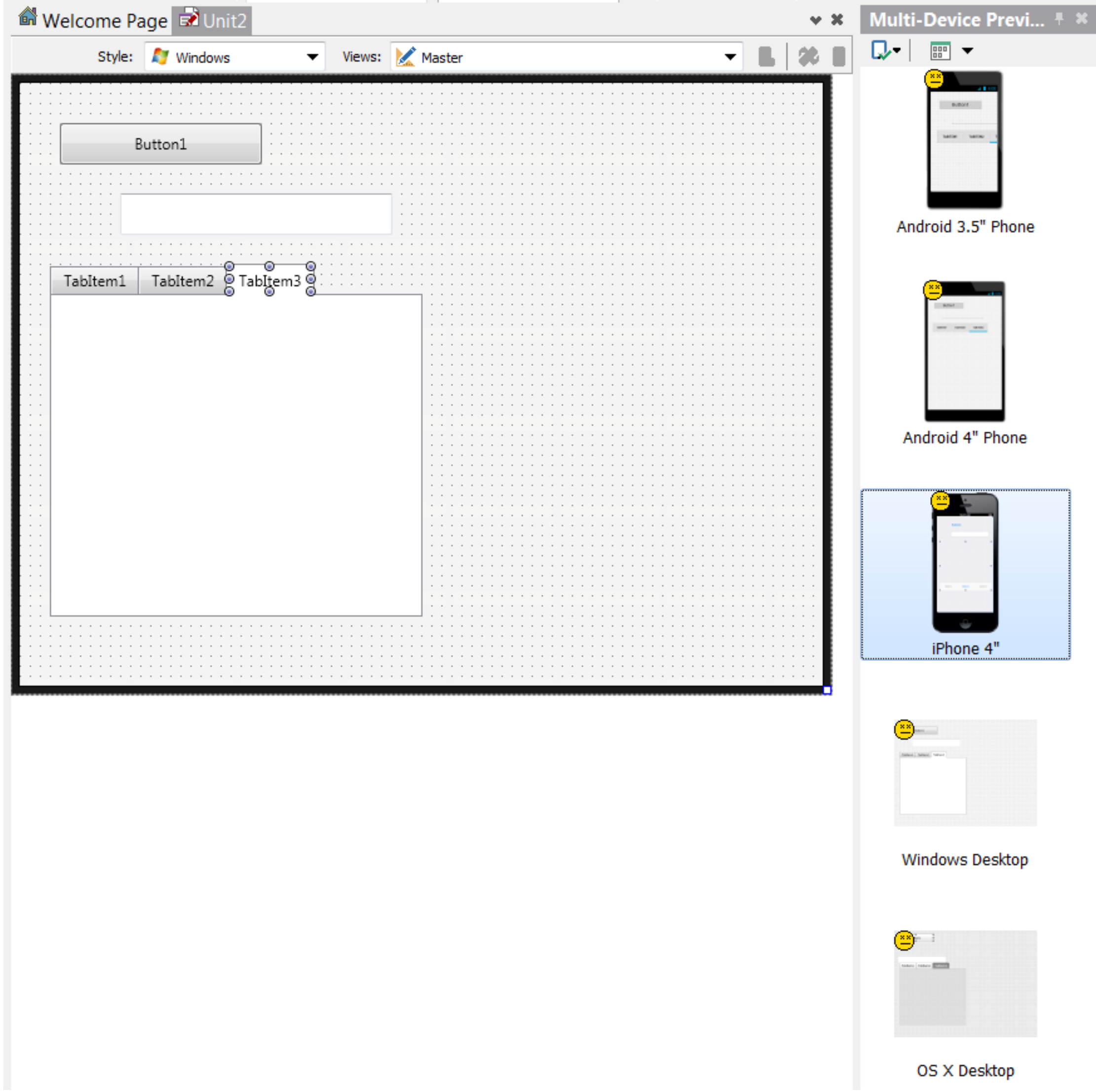
Multi-Device 在使用上非常的簡單，只要在開發專案時開啟 Multi-Device 設計視窗即可。例如下圖是一個 Multi-Device 專案，它同時要開發 Android 3.5 吋，Android 4 吋和 iPhone 吋的設備：



在開始設計時請點選 View | Multi-Device Preview 選單並且在 Master View 中開始放入視覺化元件，就可以看到類似下圖的畫面，View | Multi-Device Preview 會同時顯示在設計 UI 時，每一個開發設備的 UI 會如何顯示。而且只要使用滑鼠按兩下 View | Multi-Device Preview 視窗中特定的設備圖像，IDE 便會切斷到點選的特定設備的視覺化設計介面。

版權所有 請勿翻印

View | Multi-Device Preview 功能在使用上非常的簡單，但卻可提供開發人員非常高的設計 UI 的生產力。



14-2 使用分散式版本控制工具-Git

Delphi XE7 便開始支援分散式版本控制工具 Git，但 XE7 只提供了最基本的 Git 功能，到了 RIO 對於 Git 的支持就比較完整了。本小節的內容將說明如何在 Delphi IDE 中使用 Git 進行版本控制功能。

要在 IDE 中使用 Git，開發人員需要進行下面的步驟：

1. 下載和安裝 Git

2. 在 IDE 中設定 Git

3. 申請帳號

要進行第 1 步驟，讀者可到 https://github.com/下載 Git：

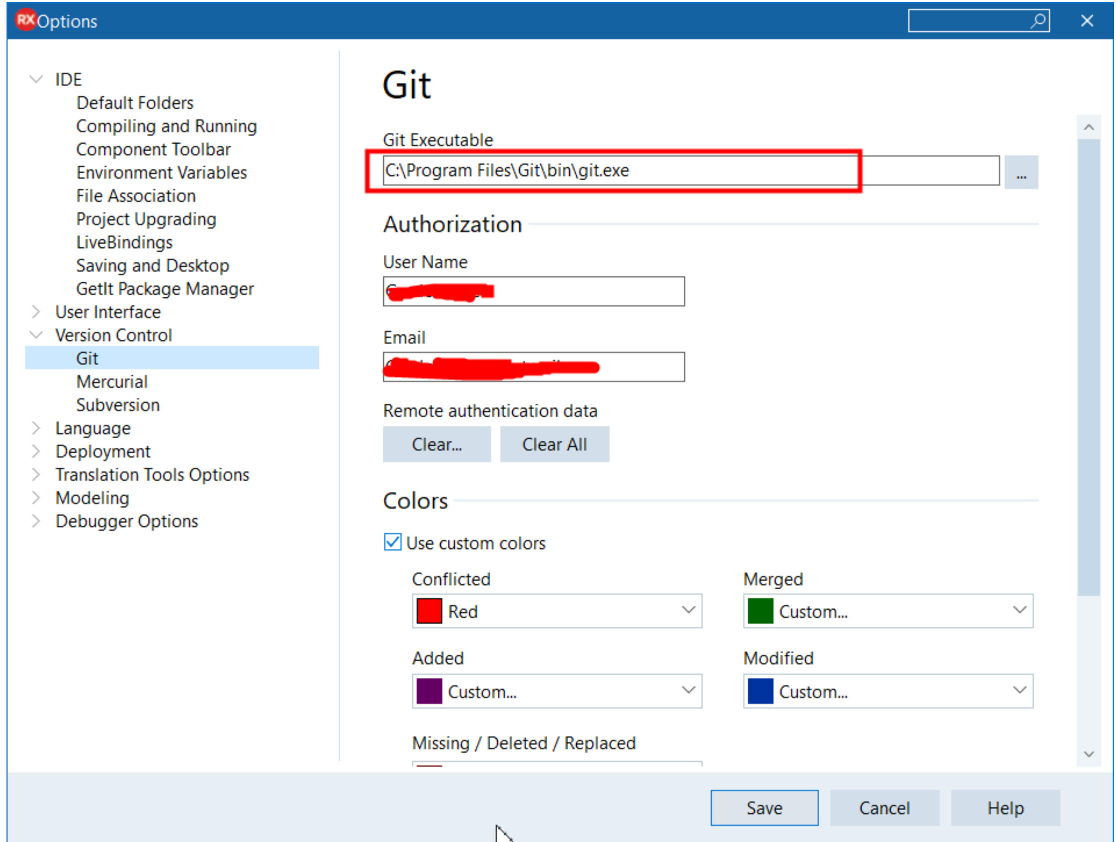
GitHub, Inc. [US] https://github.com

The easiest way to use GitHub on Windows

Download GitHub for Windows

For Windows Vista, 7, 8, & 8.1 · Learn more

下載 Git 之後請安裝它，安裝完之後請到 Delphi IDE 中點選 Tools|Options 選項，在 Version Control|Git 類別中請在 Git Executable 欄位中輸入您安裝的 Git 執行檔位置：



在上面的對話盒中還需要您輸入您申請的 Git 帳號資訊，這是第 3 個步驟。

要使用 Git，開發人員需要先到 GitHub 或是 BitBucket 申請使用空間，

讀者可到

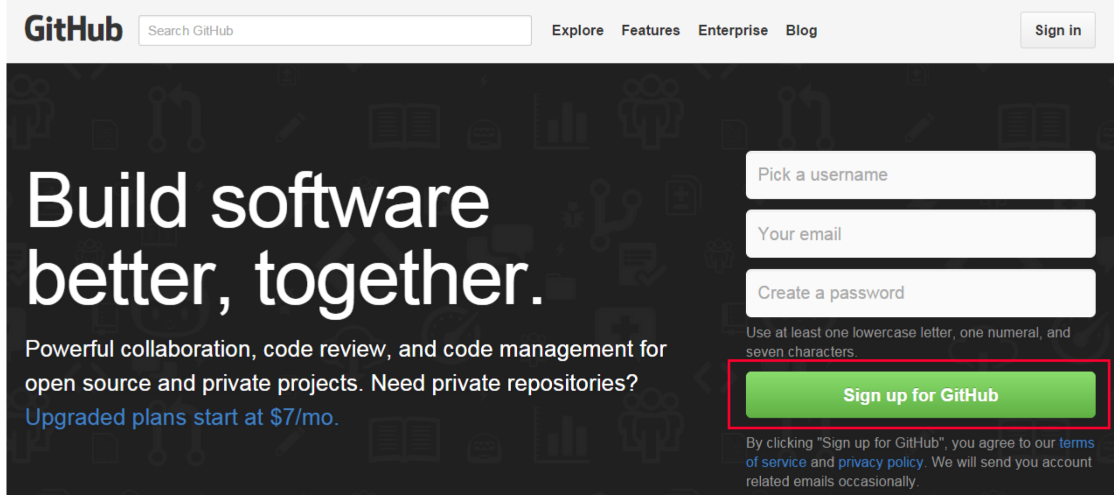
```
https://github.com/
```

申請使用 GitHub，或是到

```
https://bitbucket.org/
```

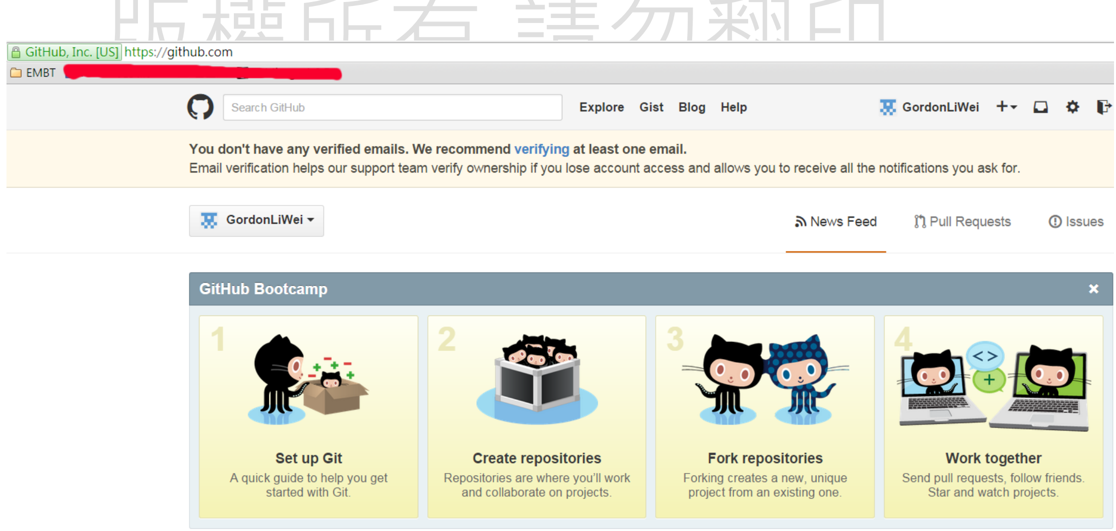
申請使用 BitBucket。在下面的內容中筆者以 GitHub 做為說明。

首先到 https://github.com/點選如下圖的"Sign up for GitHub"並申請帳號：

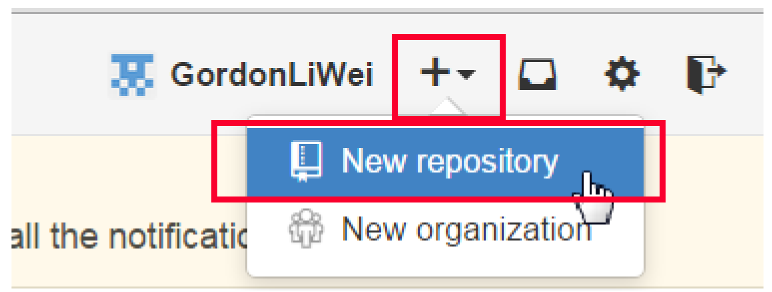


在成功申請帳號並登入之後可看到如下的畫面，不過申請完帳號後請記得回到 IDE 的 Tools | Options | Version Control | Git 類別輸入您的帳號資訊：

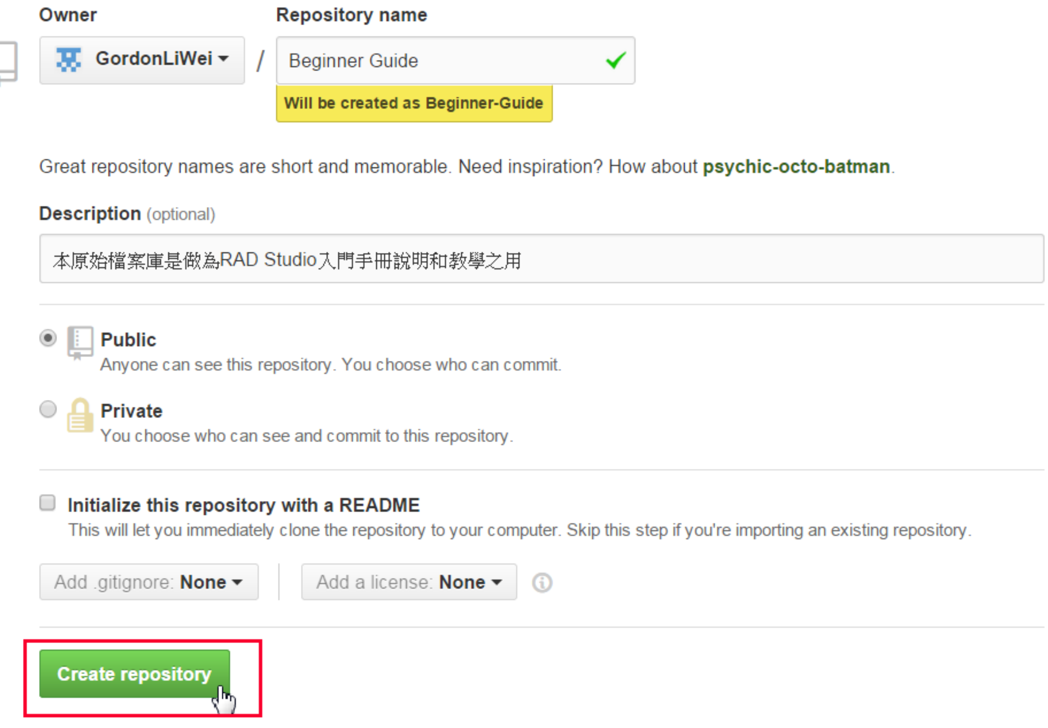
版權所有 請勿翻印



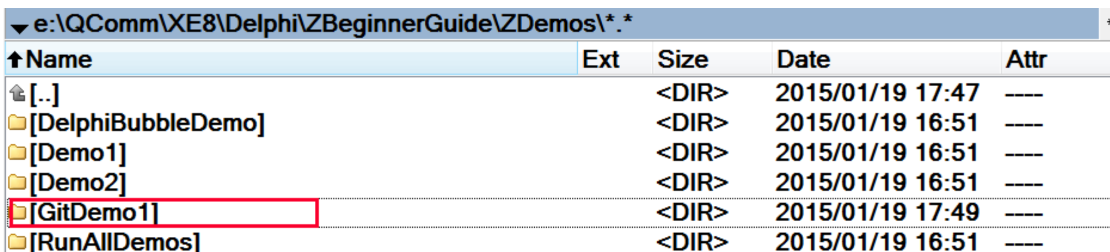
此時可點選您帳號名稱右邊的"+"號以建立一新的程式庫，如下所示：



點選”+”號之後會看到如下的類似畫面，您需要為新的程式庫取一名稱和輸入一簡短的說明之後點選下方的”Crate repository”按鈕真正的建立程式庫：



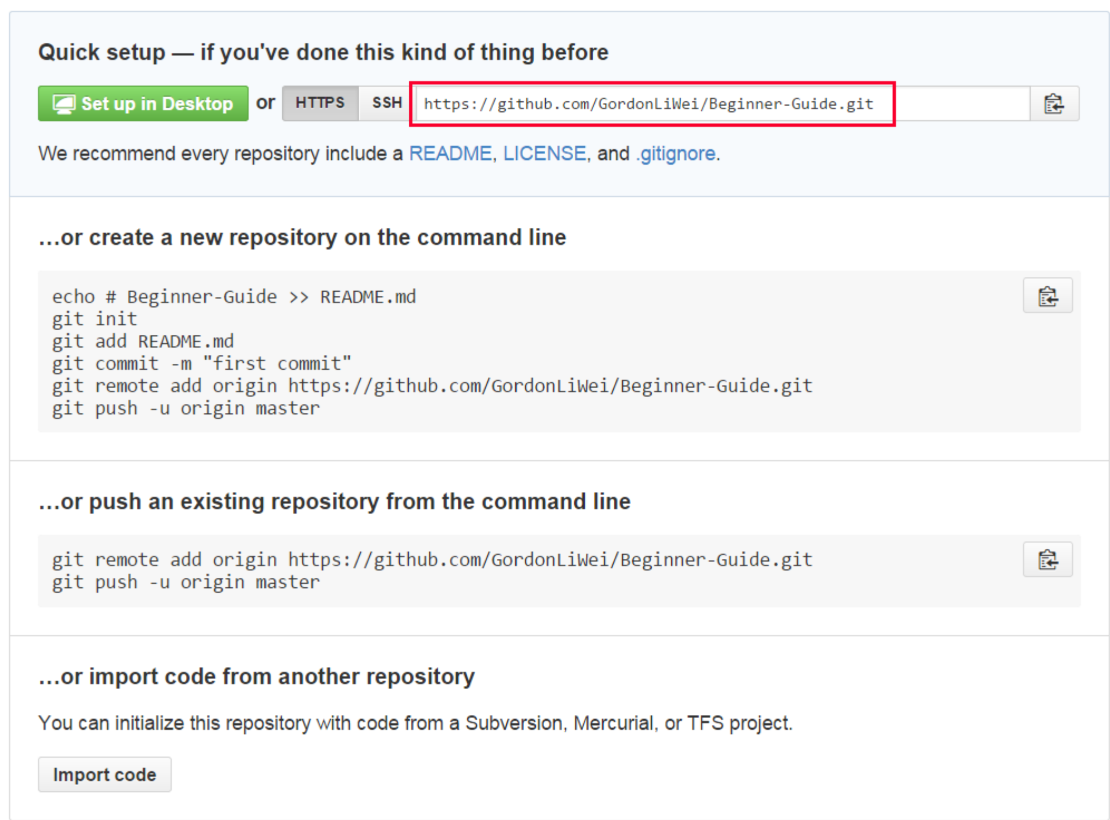
回到你的電腦中，我們現在需要在本機中先建立一個文字檔並簽入到遠端的剛才建立的程式庫中。下圖是筆者在本機的 GitDemo1 目錄中建立一個”讀我.txt”文字檔：



e:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo1*.*

Name	Ext	Size	Date	Attr
[..]		<DIR>	2015/01/19 17:49	----
讀我	txt	0	2015/01/19 17:49	-a--

事實上當您建立了程式庫時 GitHub 便會顯示如下的內容教導您如何把一個檔案簽入到程式庫中：

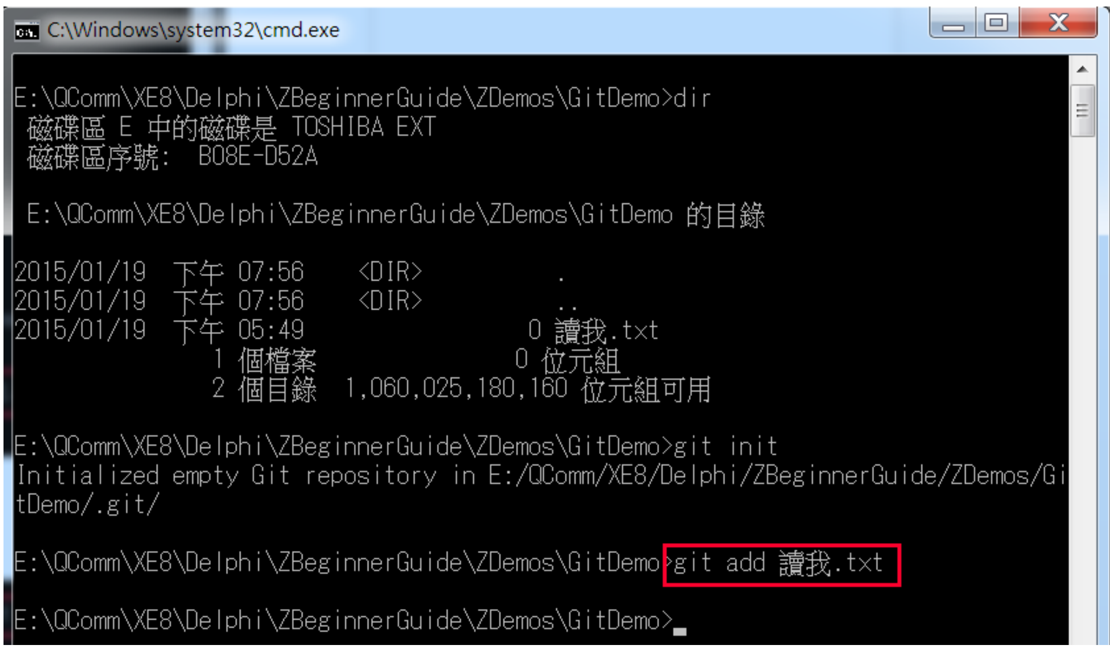


現在就讓我們使用上面的說明把 GitDemo1 目錄中”讀我.txt”文字檔簽入到遠程的 GitHub 中。

開啟一個命令列視窗並且到 GitDemo1 目錄執行”git init”命令，Git 便會在此目錄建立本機程式庫資訊：

```
C:\Windows\system32\cmd.exe

E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo>dir
 磁碟區 E 中的磁碟是 TOSHIBA EXT
 磁碟區序號:  B08E-D52A

 E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo 的目錄

2015/01/19  下午 07:56    <DIR>          .
2015/01/19  下午 07:56    <DIR>          ..
2015/01/19  下午 05:49                 0 讀我.txt
               1 個檔案               0 位元組
               2 個目錄  1,060,025,180,160 位元組可用

E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo>git init
Initialized empty Git repository in E:/QComm/XE8/Delphi/ZBeginnerGuide/ZDemos/Gi
tDemo/.git/

E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo>
```

再執行”git add 讀我.txt”命令把”讀我.txt”文字檔加入到 git 的 stage 檔案串列中：



再執行”git commit –m 第 1 個 Commit”命令簽入此文字檔到本機的程式庫中：

```
E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo>git commit -m "第1個Commit"
[master (root-commit) fc7ffcb] 第1個Commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 "\350\256\200\346\210\221.txt"
```

接下來我們就準備把本機簽入的”讀我.txt”文字檔同步到遠端的 GitHub 的程式庫中，但在這之前您可能需要執行”git config –global user.email 您的 email 地址”讓 Git 知道要簽入到遠端的什麼地方：

```
E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo1>git config --global user.emai
l "gordonliwei@hotmail.com"
```

最後執行"git remote add origin https://github.com/GordonLiWei/您的遠端程式庫名稱”命令連結本機和遠端程式庫，再執行” git push -u origin master”命令把本機的”讀我.txt”文字檔從本機的程式庫真正簽入到遠程的程式庫中，在這一步驟中您可能需要輸入登入資訊：

```
E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo>git remote add origin https://
github.com/GordonLiWei/Beginner-Guide

E:\QComm\XE8\Delphi\ZBeginnerGuide\ZDemos\GitDemo>git push -u origin master
Username for 'https://github.com': GordonLiWei
Password for 'https://GordonLiWei@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 226 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/GordonLiWei/Beginner-Guide
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

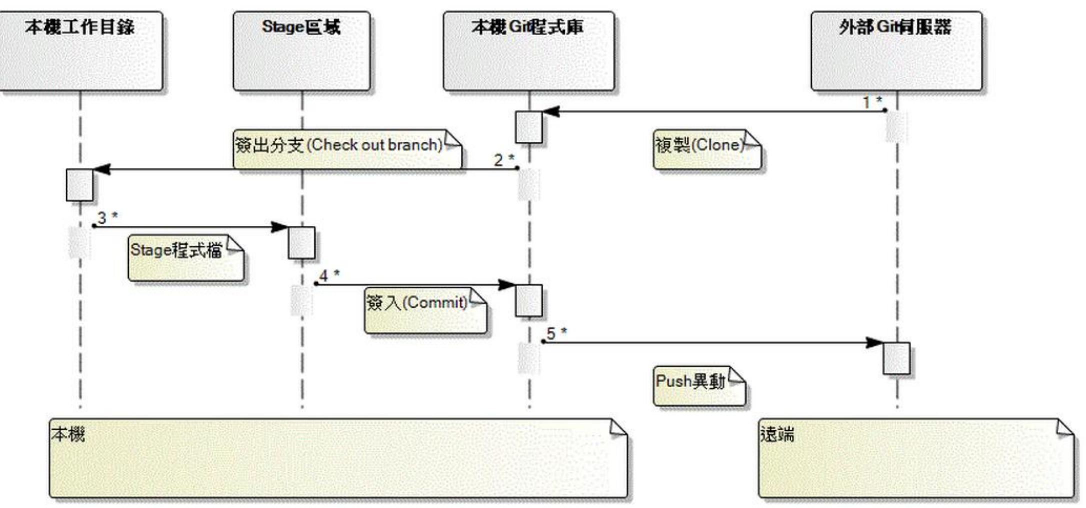
版權所有 請勿翻印

最後回到流覽器並查看遠端 GitHub 的程式庫就可以看到”讀我.txt”文字檔已經出現在其中了：

到了這裡我們就可以開始使用 Delphi IDE 來進行項目的版本控制了，而我們使用的程式庫就是剛才在遠端于 GitHub 中建立的程式庫。但在說明如何在 Delphi IDE 中使用 GitHub，先讓我們說明一下什麼是分散式版本控制，一旦讀者瞭解之後就能明白前面和稍後在 Delphi IDE 中執行的動作是什麼意思。

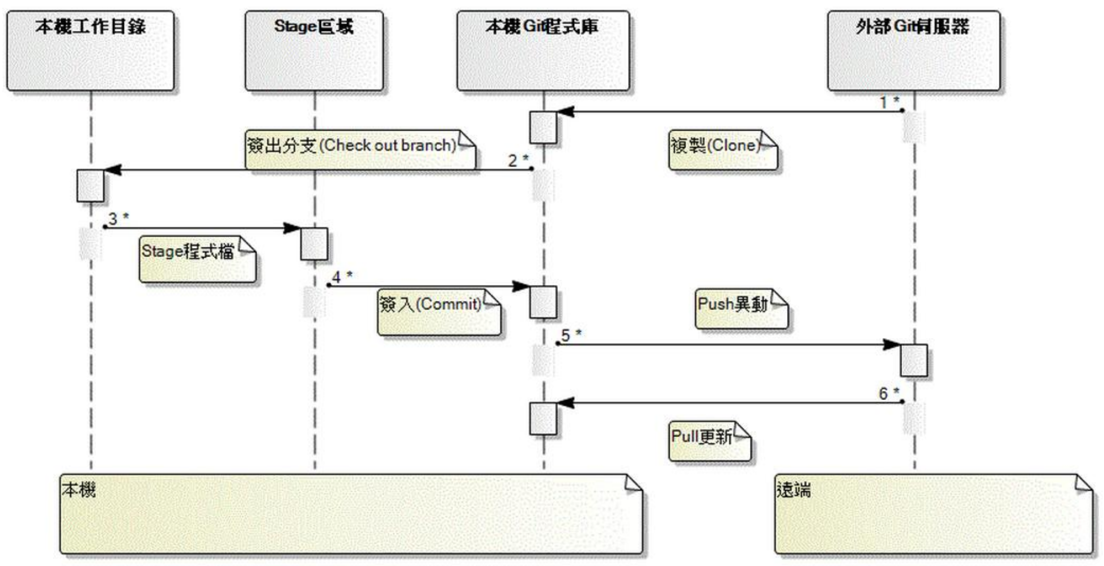
傳統的版本控制工具，例如 StarTeam 和 SubVersion 等都是屬於中央控制的 C/S 架構，每一個用戶端的開發人員從版本控制伺服器簽出程式開發和異動，最後再簽入回版本控制伺服器。但 Git 是屬於分散式的版本控制系統，Git 除了有遠端版本控制伺服器之外(即 GitHub)，Git 也會在本機建立本機程式庫，Git 可同步遠端版本控制伺服器和本機的程式庫。

讓我們使用下面的圖形簡單說明 Git 基本的運作原理 ，在前面我們于 GitHub 上建立的程式庫就類似下圖中的外部 Git 伺服器，一開始我們從外部 Git 伺服器複製程式到本機 Git 程式庫，接著我們簽出分支，進行開發的工作。開發到一段落之後我們可 Stage 開發的程式檔，確定異動之後再簽入異動到本機 Git 程式庫。到此所有的開發，異動和簽出/簽入都是在本機之中，因此可減少外部 Git 伺服器的負荷。當本機的開發到一段落之後我們可以 Push 本機簽入的結果到外部 Git 伺服器，如此一來團隊其他的成員就可以複製/簽入這些開發成果，再繼續進行團隊開發。



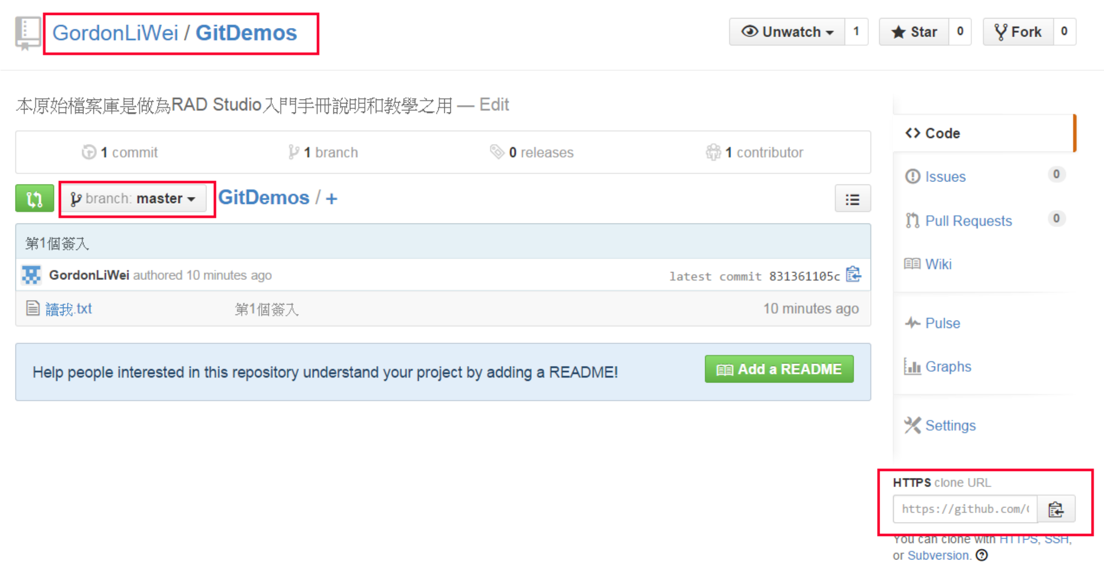
Git 基本運作原理

如果團隊中其他成員開發到一段落那麼外部 Git 伺服器可以主動再把它們 Push 到你的本機 Git 程式庫我們就可以再簽出進行後續開發。當然本機 Git 程式庫也可以主動執行 Pull 命令從外部 Git 伺服器更新本機 Git 程式庫：



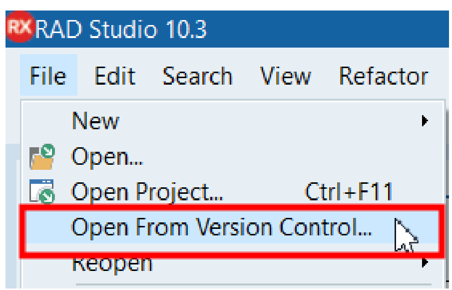
從上面的簡單說明讀者應該可以瞭解為什麼現在 Git 這麼流行，因為 Git 提供了分散式的版本控制能力，又能整合遠端 Git 伺服器和本機 Git 程式庫進行團隊開發。

在具備了 Git 基本的觀念後，我們就可以開發解釋如何在 RIO 中使用 Git，筆者在 GitHub 中建立了一個”GitDemos”程式庫做為上圖中的遠端伺服器：



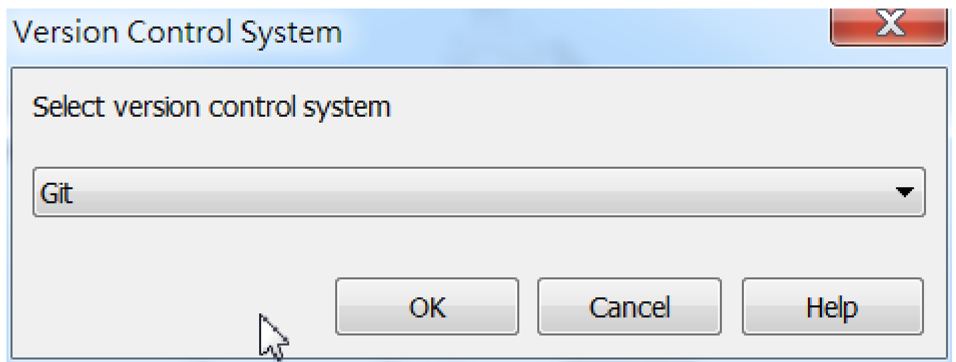
請注意上圖中這個遠端程式庫目前的分支是”Master”，而在上圖右下角的"HTTPS clone URL”處即是指向此遠端程式庫的 URL，稍後在 Delphi IDE 中將使用這個 URL 複製(clone)程式到本機 Git 程式庫，也就是前圖” Git 基本運作原理”中的第 1 步驟。

在 GitHub 的”GitDemos”程式庫頁面中點選拷貝”GitDemos”程式庫 URL：

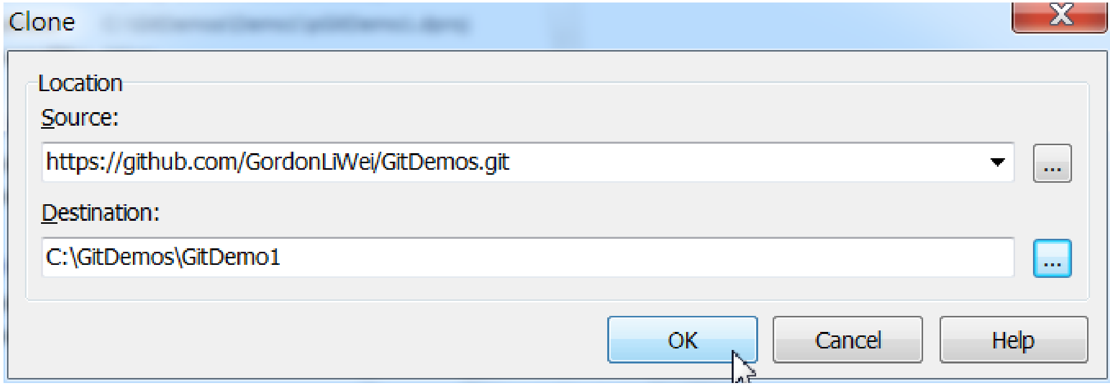


現在回到 Delphi IDE，點選 File | Open From Version Control...選項：

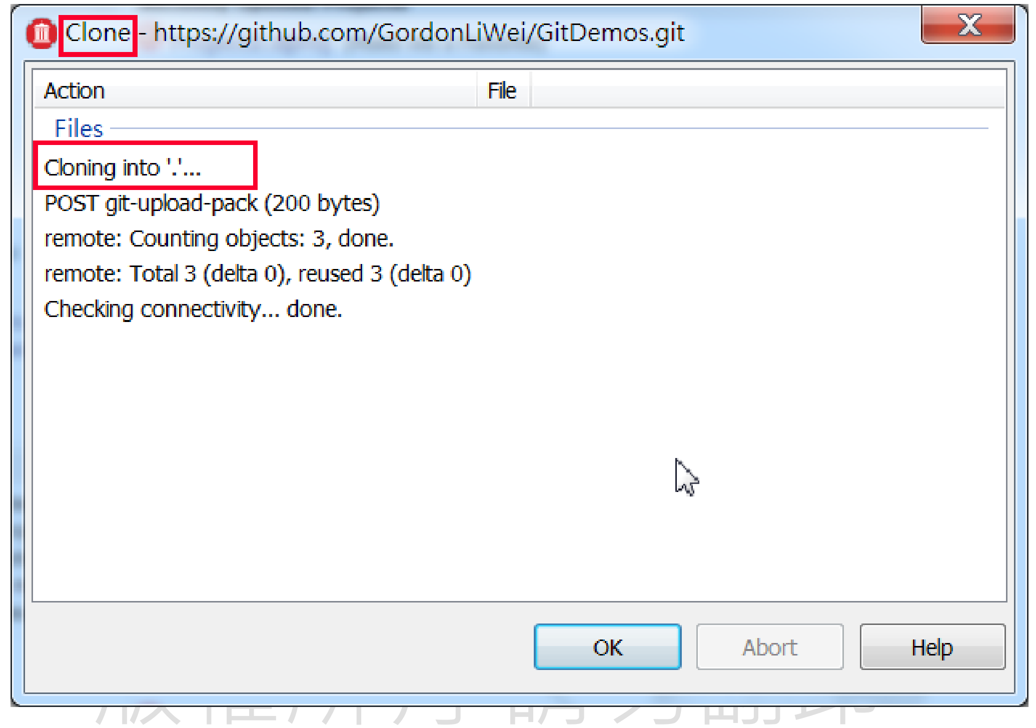
從 Version Control System 對話盒中選擇 Git：



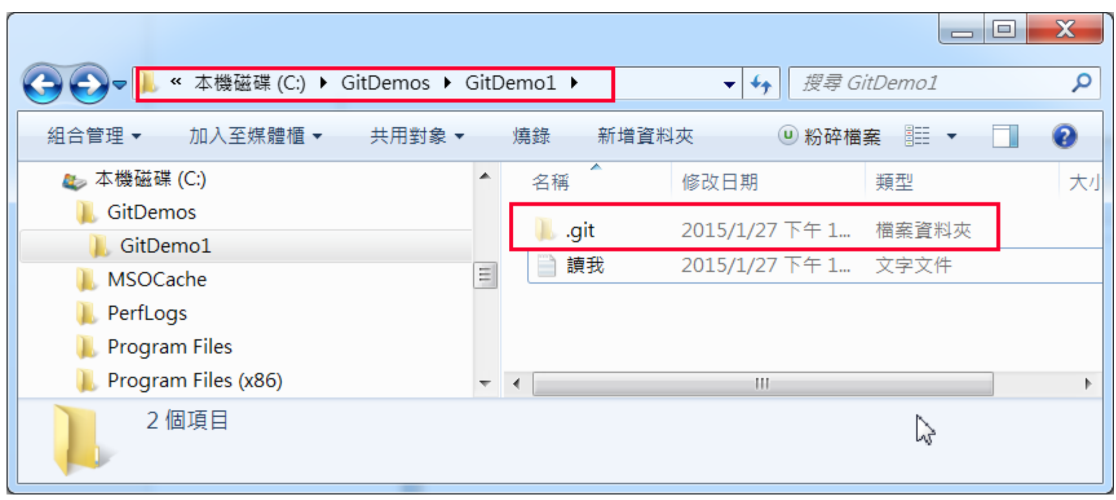
點選上圖的 OK 按鈕後在 Clone 對話盒中的 Source 欄位中貼上前面拷貝的”GitDemos”程式庫 URL，再於 Destination 欄位中輸入複製(clone)到本機 Git 程式庫的目錄：



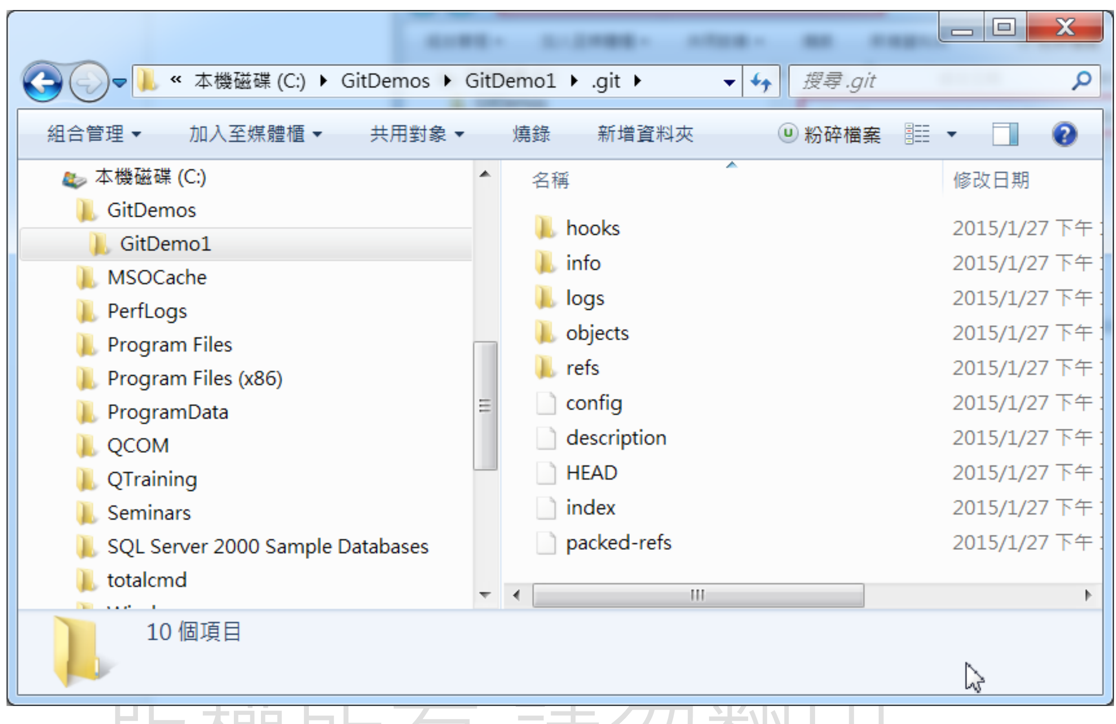
最後點選 OK 按鈕，就可以看到 Delphi IDE 顯示如下的複製對話盒，您可以看到它顯示 Git 正在複製(cloning)遠端的程式庫內容到本機的 c:\GitDemos\GitDemo1 目錄中。這印證了 Delphi IDE 正在執行前圖”Git 基本運作原理”中的第 1 步驟：



點選 OK 按鈕之後到 c:\GitDemos\GitDemo1 目錄中查看，我們可以看到在 c:\GitDemos\GitDemo1 目錄中果然複製了遠端”讀我.txt”而且在目錄中出現了一個 .git 子目錄：

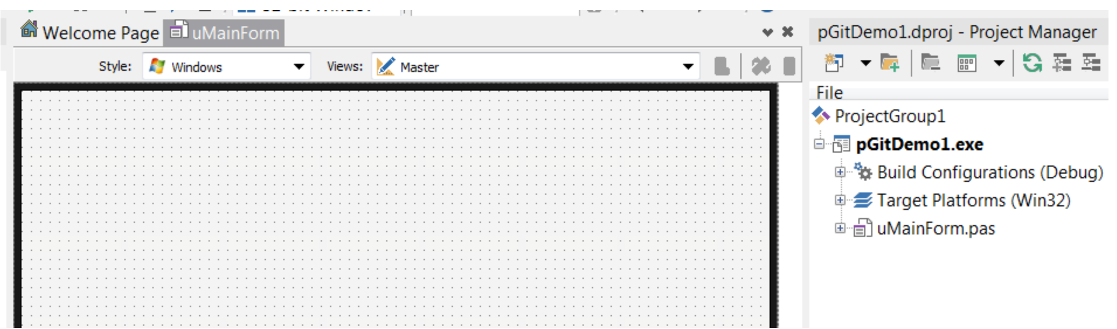


.git 子目錄就是 Git 在 c:\GitDemos\GitDemo1 目錄中建立的本機程式庫，我們如果到.git 子目錄中就可以看到下面的內容，.git 子目錄是由 Git 管理的本機程式庫：

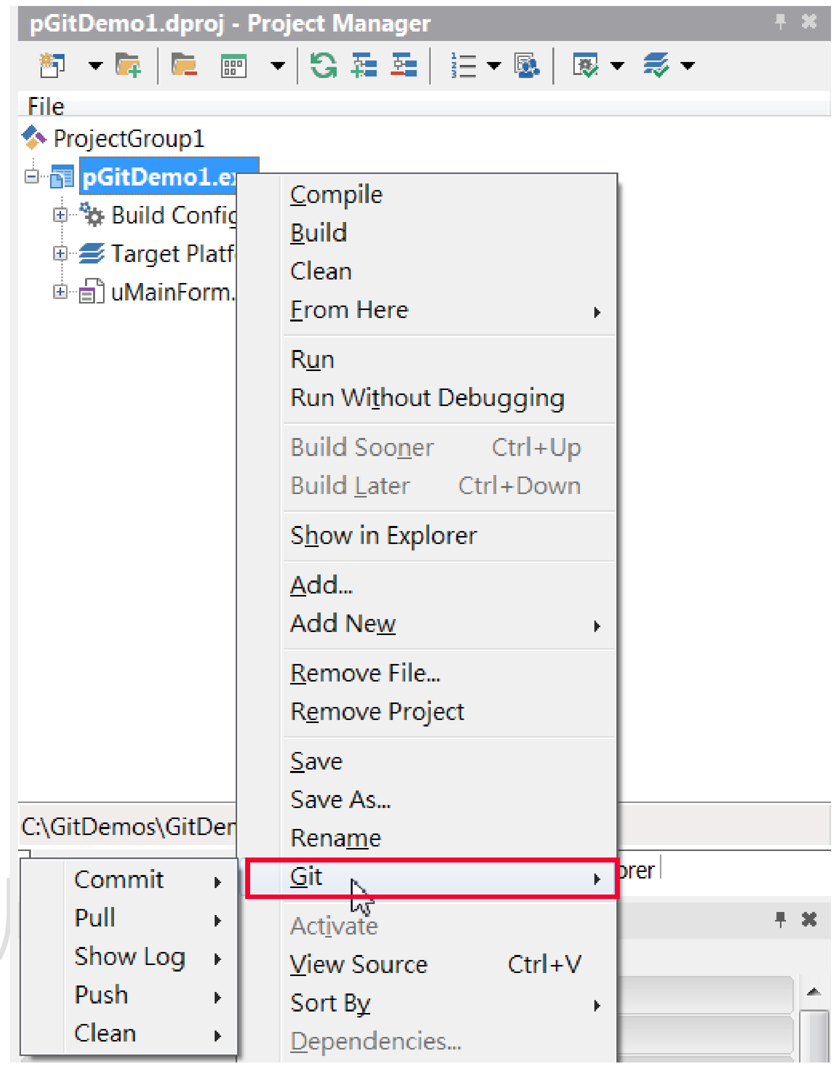


版權所有 請勿翻印

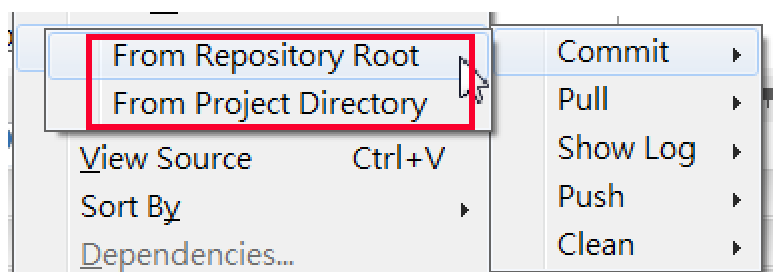
讓我們建立一個 Multi-Device 專案並儲存在 c:\GitDemos\GitDemo1 目錄，如下所示：



接著在專案管理員中點選滑鼠右鍵可以看到有 Git 選項,在其中有數個可執行的 Git 命令：

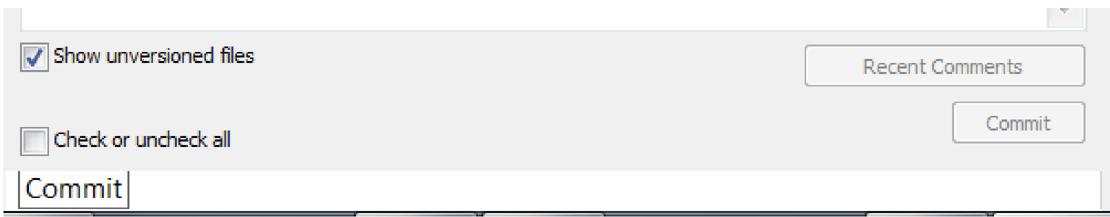


在 Git 選項中有 Commit 子選項，在其中有 2 個命令”From Repository Root”和”From Project Directory”：

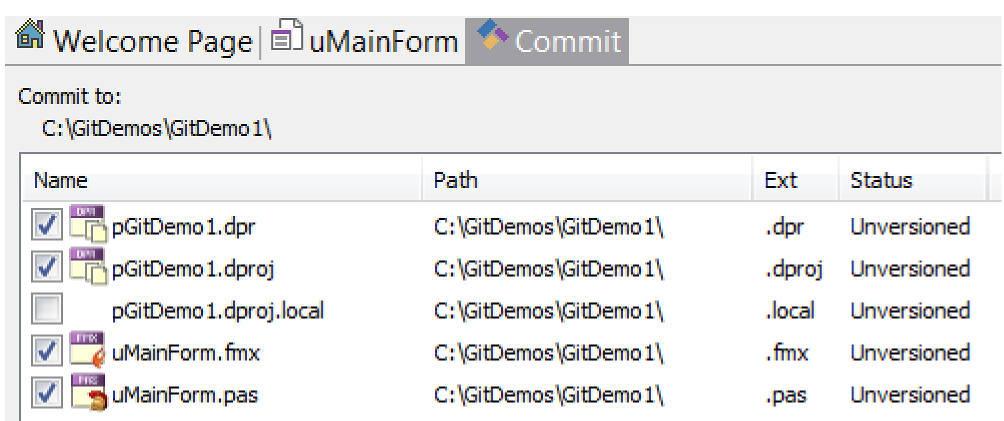


現在讓我們把這個項目簽入到本機的 Git 程式庫中，由於現在本機中本機的 Git 程式庫和 Delphi 項目是在同，目錄中，因此上圖中的”From Repository Root”和”From Project Directory”2 個命令效果是一樣的，所以請讀者點選上圖中的 Git|Commit|From Repository Root 選項，準備把專案簽入到本機的 Git 程式庫。

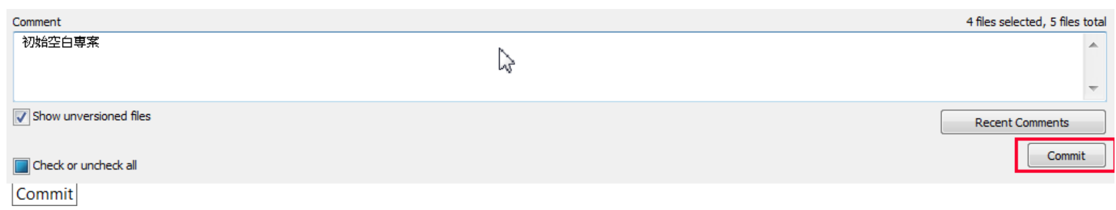
點選 Git | Commit | From Repository Root 選項後 IDE 會顯示 Commit 頁面讓開發人員可以簽入檔案，請先 Commit 頁面下方的 Show unversioned files 以顯示尚未簽入的專案檔案：



此時 Commit 頁面便會顯示專案中所有的檔案，請勾選要簽入的檔案，例如下圖顯示要簽入 4 個檔案，目前每一個檔案的狀態欄位元都是 Unversioned：

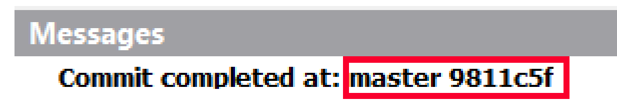


接著在下方的 Comment 欄位輸入簽入說明，此時右下方的”Commit”按鈕會致能，輸入完畢之後請點選”Commit”按鈕執行 Stage 檔案和簽入檔案到本機的 Git 程式庫中。



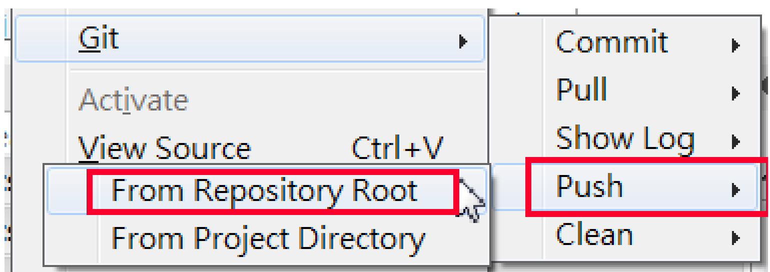
點選”Commit”按鈕就是前圖” Git 基本運作原理”中的第 3 和第 4 步驟。

點選”Commit”按鈕之後 IDE 會執行 Git 的 Commit 命令，成功之後 IDE 的訊息視窗就會顯示一簽入代號如下所示代表簽入成功：

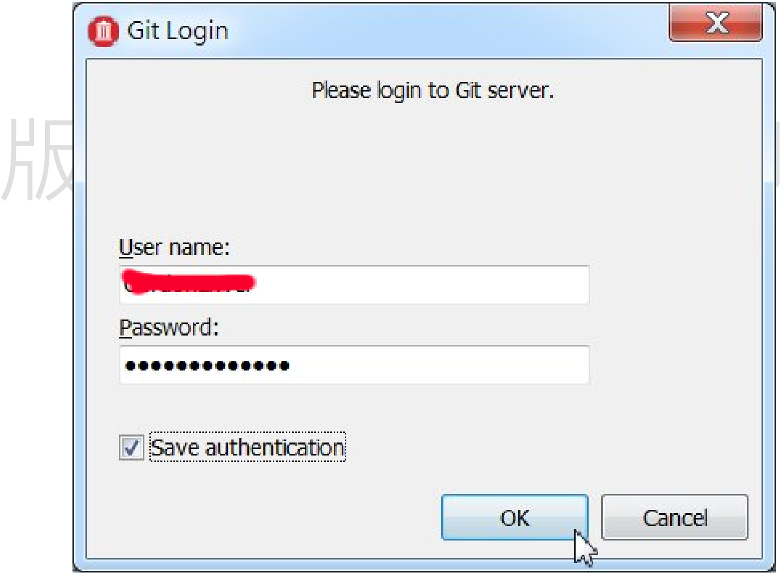


請注意現在我們只是把專案檔案簽入到本機程式庫中，並沒有把項目推播到遠端 Git 伺服器中，因此其他團體成員無法共用現在的開發成果，因此如果現在去 GitHub 的 GitDemos 程式庫我們看不到剛才簽入的專案檔案。

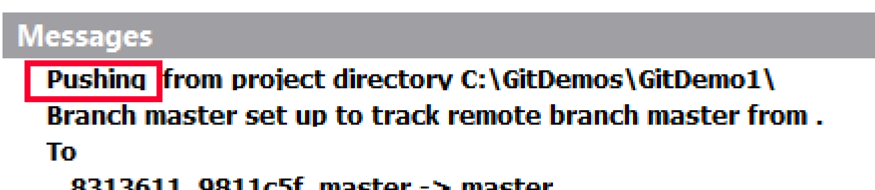
因此現在讓我們把此項目推播到遠端 Git 伺服器中，以便進行團隊開發。請點選 Git | Push | From Repository Root 選項：



由於現在要推播專案到遠端 GitHub 的程式庫，因此 IDE 會顯示如下的對話盒要求輸入您在 GitHub 的帳號資訊：



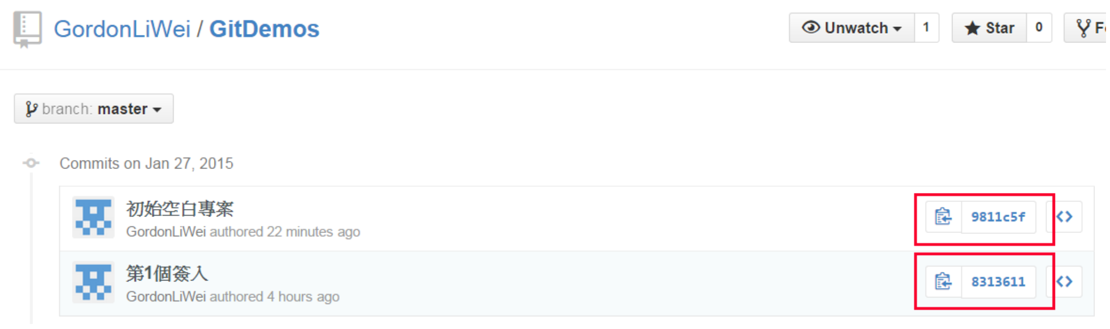
點選 OK 按鈕之後 IDE 便會執行 Git 的 Push 命令，成功之後 IDE 在訊息視窗會顯示推播成功的資訊，從下圖可以看到 IDE 的確是執行 Push 命令，而且會顯示從本機的 master 分支推播到 GitHub 的程式庫中 GitDemos 程式庫的 master 分支。



這個動作就是前圖” Git 基本運作原理”中的第 5 步驟。

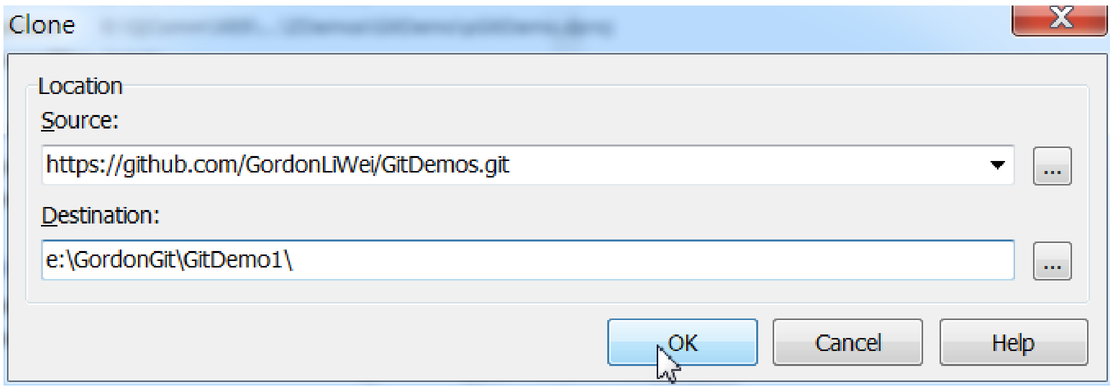
現在回到 GitHub，重新整理 GitDemos 程式庫頁面我們就可以看到類似下面的結果，專案中的檔案果然簽入到遠端的 Git 伺服器了：

請注意上圖中顯示我們有 2 次簽入的動作，第 1 次是簽入”讀我.txt”檔案，第 2 次當然就是剛才簽入項目的動作。如果我們點選上圖中的 2 commits，那麼就可以看到如下的結果：

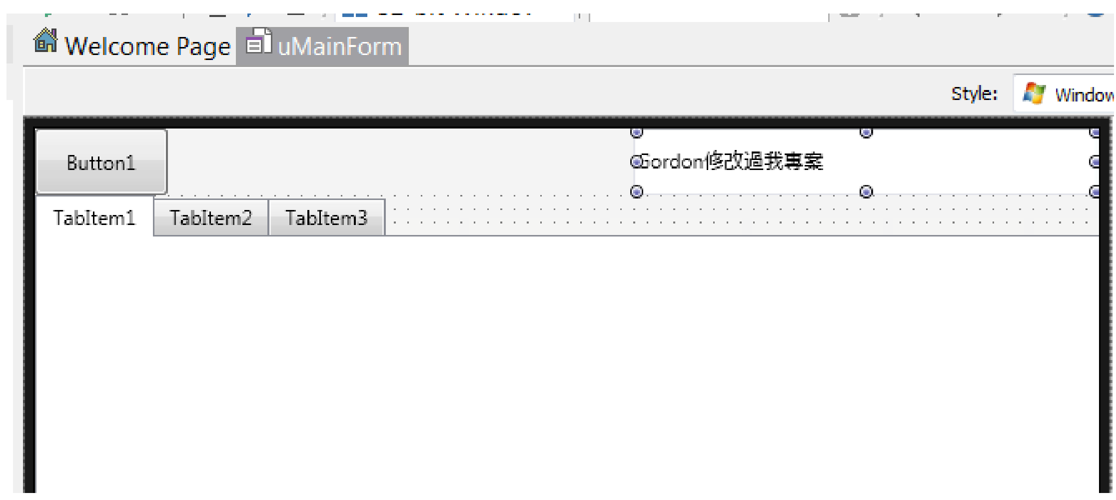


我們可以看到 2 次簽入的批註和每次的簽入代號。

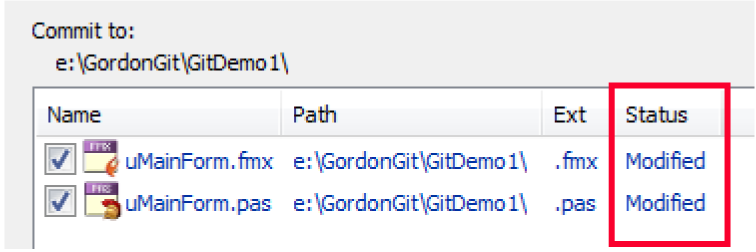
現在假設有一個團隊成員 Gordon 在另一台機器中想使用剛才推播到 GitHub 伺服器中的項目，那麼 Gordon 也可以在 IDE 中點選 File|Open From Version Control...選項從遠端 GitHub 伺服器中複製專案到 Gordon 的本機中，就像前面逆明的流程一樣，



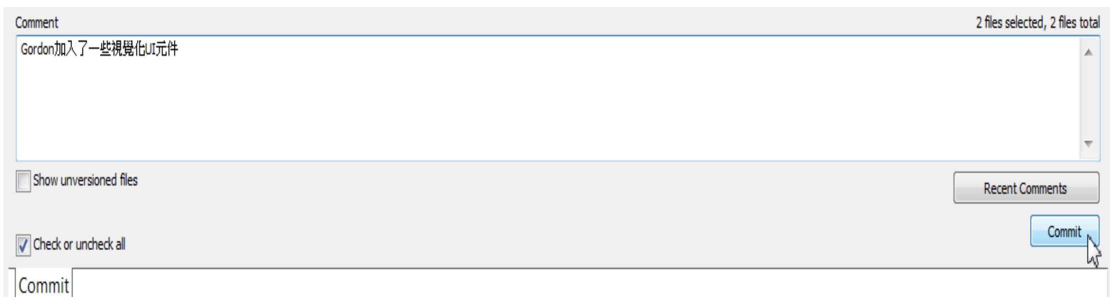
然後 Gordon 在複製到本機的專案中加入如下的 UI 組件：



接著 Gordon 點選 Git | Commit | "From Repository Root"選項就可以在 IDE 中看到下面的 2 個專案檔案被修改過了：

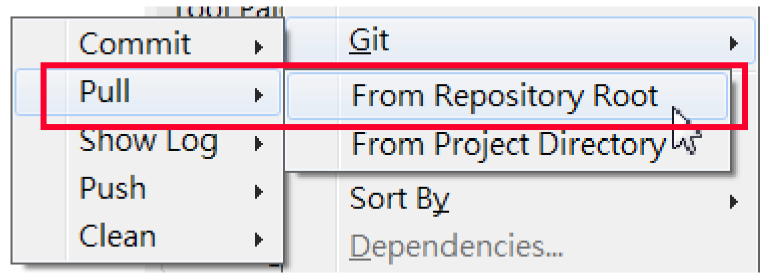


Gordon 輸入新的簽入說明再點選 Commit 按鈕簽入修改的檔案：

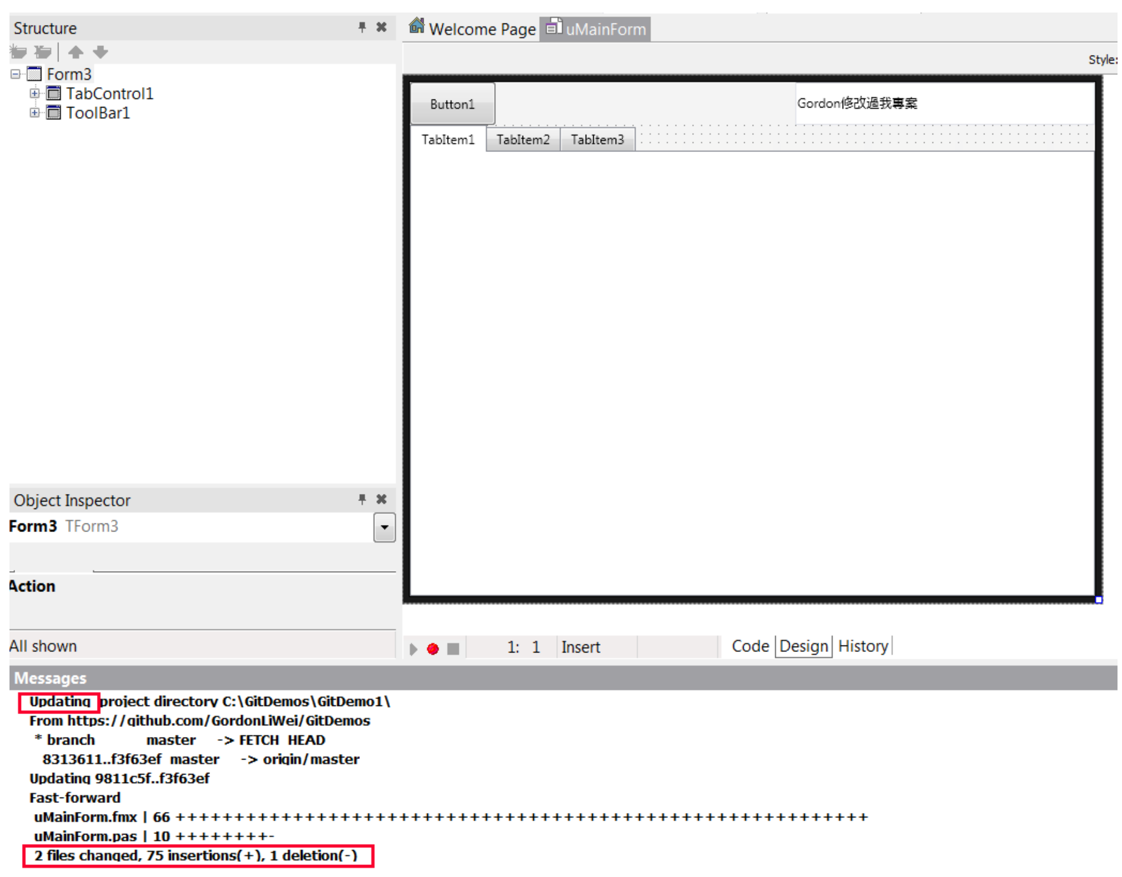


再點選 Git | Push | From Repository Root 選項把修改的專案推播回遠端 Git 伺服器，之後我們就可以看到類似下面的結果，Gordon 修改的項目果然簽回到遠端的 Git 伺服器了：

現在我們再回到原先的機器中，我們想繼續 Gordon 的修改工作，因此請在 IDE 中點選 Git | Pull | From Repository Root 把專案的異動更新回原來的本機中：



數秒後我們就可以在原先的 IDE 中看到 Gordon 修改的專案已經整合到原先機器中的專案了，而我們就可以接著 Gordon 的開發工作繼續往下開發，這就是分散式團隊開發的範例：



請注意在上圖中當我們從遠端 Git 伺服器中取得 Gordon 的異動時，IDE 會執行更新本機中原本專案的內容，合併我們的初始開發內容以及 Gordon 進行的開發異動。

RIO 整合的分散式版本控制工具 Git 可以讓團隊開發非常的方便且有效率，非常值得讀者使用在日常的開發工作中。

14-3 使用 DUNITX 單元測試框架

Delphi 從很早的版本即使用 DUNIT 框架來支援單元測試，到了 RIO 開始支援 DUNITX 框架來支援單元測試，因此從 Delphi RIO 開發同時支援 DUNIT 和 DUNITX。RIO 開始使用 DUNITX 主要是因為：

1. DUNITX 支援跨平臺的單元測試

2. DUNITX 使用 RTTI 的屬性來定義單元測試類別而無需從特定的單元測試類別繼承，因此任何的 Delphi 類別都可以成為測試類別。如此一來 Delphi 測試類別可免於受到單元測試框架的改變而需要修改

DUNITX 單元測試框架現在特別重要的原因就是因為現在 Delphi 可開發多個平臺，因此為了減少需要在多個平臺測試/除錯的成本，跨平臺的單元測試能力就更重要了。

本小節將簡單的說明如何使用 DUNITX 單元測試框架，希望讀者在瞭解之後能夠使用在每日的開發工作之中。

14-3-1 DUNITX 單元測試框架簡介

DUNITX 是一開放原始碼單元測試框架，DUNITX 主要是由 Vincent Parrett 先生開發的，隨後有數位加入的貢獻者。DUNITX 框架從 Delphi 2010 版即開始支持，讀者可以在下面的 URL 下載 DUNITX 框架：

```
https://github.com/VSoftTechnologies/DUnitX
```

到了 Delphi RIO 版正式加入到 Delphi 的 IDE 中。

版權所有 請勿翻印

基本上 DUNITX 比 DUNIT 提供了更多的功能但使用上卻更方便，下表簡列了這 2 個單元測試框架的差異：

功能,	DUnit,	DUnitX
基礎測試類別	TTestCase	無
測試方法	須宣告在 Published	宣告在 Published 或是使用[Test]屬性標注
Fixture Setup 方法	無	使用[SetupFixture] 屬性標注或是使用建構元(Constructor)
Test Setup 方法	覆載基礎測試類別的 Setup 方法	使用 [Setup] 屬性標注
Test TearDown 方法	覆載基礎測試類別的 TearDown 方法	使用 [TearDown] 屬性標注
命名空間	藉由註冊的字串參數	單元名稱
資料驅動測試	無	使用 [TestCase(parameters)] 屬性標注

Asserts	Check(X),	Assert 類別
Asserts on Containers(IEnumerable<T>),	人工撰寫	Assert.Contains*, Assert.DoesNotContain*, Assert.IsEmpty*
使用正規標記法 Asserts	無	Assert.IsMatch (XE2 及以後的版本).
Stack Trace support	Jcl,	Jcl, madExcept 3, madExcept 4, Eurekalog 7
記憶漏失檢查	FastMM4	FastMM4
IoC Container	使用 Spring 或其他框架	內建簡易 IoC container
Console Logging	內建	內建
XML Logging	內建 (own format),	內建

為了讓讀者瞭解 DUNITX 測試框架使用原理，在下一小節中讓我們一步一步的來說明如何撰寫 DUNITX 測試框架測試類別以及上表的意義。

14-3-2 如何成為 DUNITX 測試框架的測試類別

版權所有 請勿翻印

從上面的表格說明我們可以瞭解在 DUNITX 測試框架中：

規則 1 任何的 Delphi 類別都可以成為測試類別

因此下面的範例類別 TMyTestClass 就是一個 DUNITX 測試框架的測試類別：

```
uses
  DUnitX.TestFramework;

Type
  TMyTestClass = class
  end;
```

非常的簡單。

規則 2 撰寫測試方法

有了測試類別之後我們需要有測試方法，在 DUNITX 中只要使用[Test]屬性標注的方就可以成為一個測試方法，因此下面的 TestPushedMessageCount 和 TestUnPushedMessageCount 方法就成為了測試方法。

```
TMyTestClass = class
public
  [Test]
  procedure TestPushedMessageCount;
  [Test]
  procedure TestUnPushedMessageCount;
end;
```

也非常的簡單。

如果你不想使用[Test]屬性也可以直接在把測試方法宣告在 published 部份再於類別之前加入{M+}編譯器指令即可，例如下面的 DefaultTest 也自動成為一個測試方法：

版權所有 請勿翻印

```
{M+}
[TestFixture]
TMyTestClass = class(TObject)
public
  [SetupFixture]
  procedure SetupFixture;
  [TeardownFixture]
  procedure TearDownFixture;
  [Test]
  procedure TestPushedMessageCount;
  [Test]
  procedure TestUnPushedMessageCount;
published
  procedure DefaultTest;
end;
```

規則 3 如何使用 Test Fixture

在許多測試中我們經常需要在測試之前先進行一些設計工作，例如開啟檔案，建立資料庫連線，建立要被測試的物件等。另外在測試完成之後則需要釋放

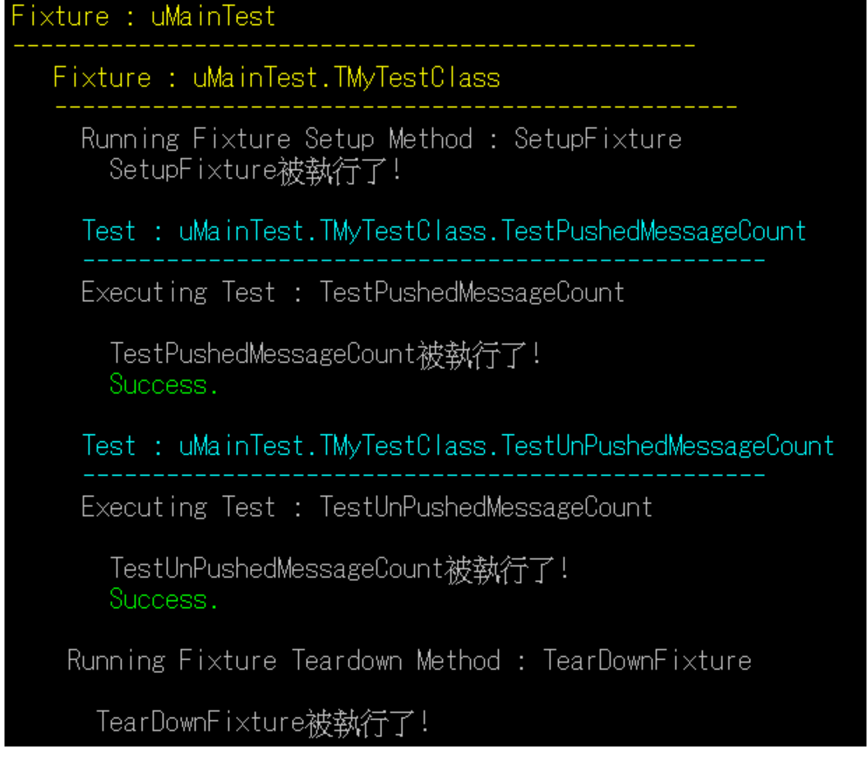
所有的資源。因此在一般測試框架中要執行測試設定的工作都是撰寫在 Setup 方法中，測試完成之後釋放所有資源的工作都是撰寫在 TearDown 方法中。

在 DUNITX 中只要使用[Setup] 屬性標注的方法就是 Setup 方法，DUNITX 在執行測試之前都會先執行使用[Setup] 屬性標注的方法。而使用[TearDown] 屬性標注方法就是 TearDown 方法，DUNITX 在執行完測試方法之後最後會執行使用 [TearDown] 屬性標注的方法。例如下面的 SetupFixture 和 TearDownFixture 方法就分別是 Setup 和 TearDown 方法：

```
TMyTestClass = class
public
  [SetupFixture]
  procedure SetupFixture;
  [TeardownFixture]
  procedure TearDownFixture;
  [Test]
  procedure TestPushedMessageCount;
  [Test]
  procedure TestUnPushedMessageCount;
end;
```

版權所有 請勿翻印

下面就是使用 DUNITX 框架執行 TMyTestClass 的結果：



我們可以從上圖看到 SetupFixture 果然是在所有測試方法之前執行而 TearDownFixture 則是在所有測試方法之後執行。

規則 4 資料驅動測試

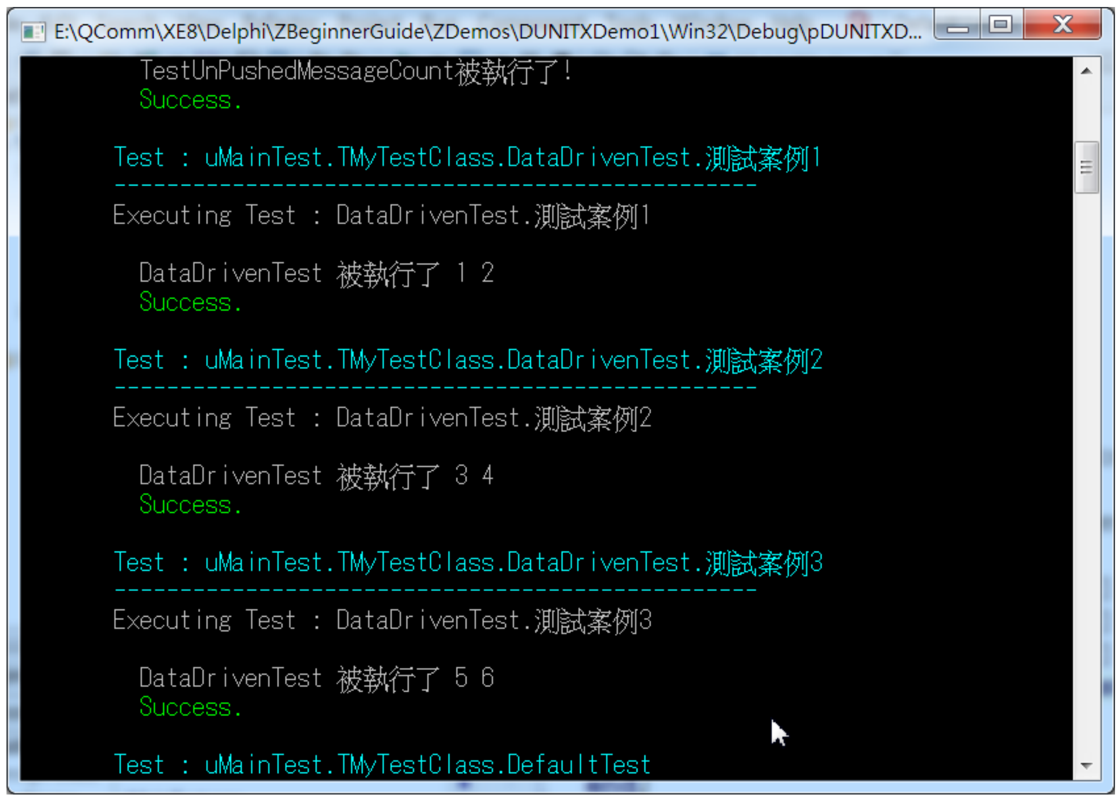
DUNITX 框架比 DUNIT 先進的功能之一就是可使用資料驅動的方式進行測試，開發人員在撰寫測試方法時可以使用[TestCase(parameters)] 屬性標注測試案例以及測試資料，如此一來 DUNITX 框架就會自動執行所有使用[TestCase(parameters)] 屬性標注的測試案例並且把測試數據傳入測試方法。

例如下面的 DataDrivenTest 測試方法之前便使用了[TestCase(parameters)] 屬性標注了 3 個測試案例以及測試資料：

```
[Test]
[TestCase('測試案例 1','1,2')]
[TestCase('測試案例 2','3,4')]
[TestCase('測試案例 3','5,6')]
procedure DataDrivenTest(const iPushedCount : Integer; const
iUnpushedCount : Integer);
```

版權所有 請勿翻印

那麼使用 DUNITX 框架執行 DataDrivenTest 測試方法我們就可以看到如下我結果：



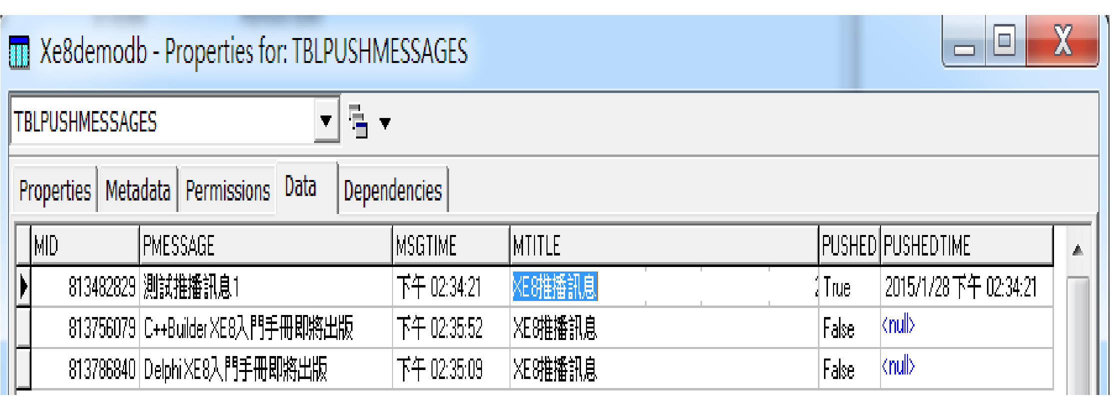
對於每一個標注的測試案例 DUNITX 框架都會自動代入測試資料並執行一次。

DUNITX 框架的資料驅動測試功能在使用上非常的方便。

規則 5 使用 DUnitX.Assert.Assert 類別測試執行結果

使用 DUNITX 框架最主要的目地當然就是測試開發人員撰寫的程式碼是否正確，每當我們實作了一個方法之後就應該撰寫一個測試方法來測試實作方法的正確性，實作第 2 個方法之後再撰寫第 2 個測試方法來測試實作方法的正確性並且不斷的一直執行所有先前的測試方法看看後來加入的實作方法有沒有影響先前的實作程式碼(依照敏捷開發方式您應該反過來，先撰寫測試方法再實作方法)。

讓我們使用一個資料表 TBLPUSHMESSAGES 來說明，一開始 TBLPUSHMESSAGES 中包含了下面的資料：



MID	PMESSAGE	MSGTIME	MTITLE	PUSHED	PUSHEDTIME
813482829	測試推播訊息1	下午 02:34:21	XE8推播訊息	True	2015/1/28 下午 02:34:21
813756079	C++Builder XE8入門手冊即將出版	下午 02:35:52	XE8推播訊息	False	<null>
813786840	Delphi XE8入門手冊即將出版	下午 02:35:09	XE8推播訊息	False	<null>

TBLPUSHMESSAGES 一共有 3 筆資料，其中已經推播的資料筆數是 1，另 2 筆資料是尚未推播的(PUSHED 欄位)。

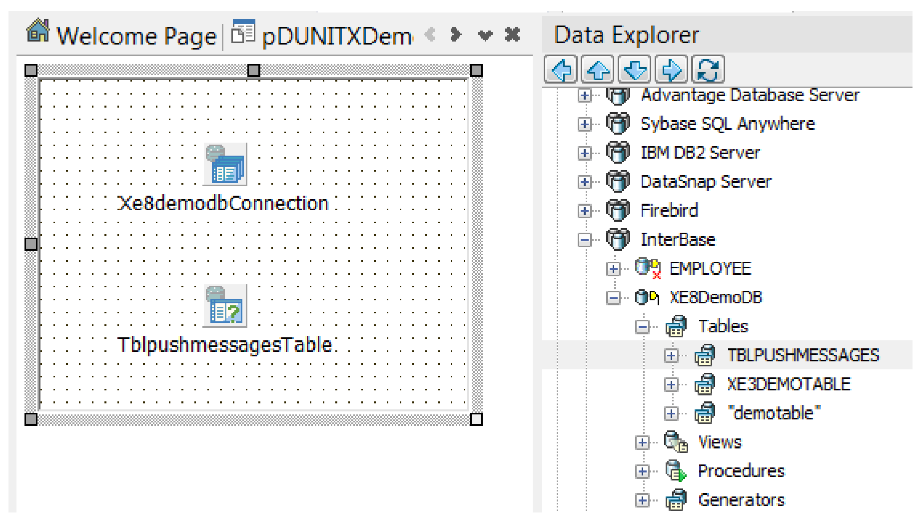
接著讓我們定義一個 Delphi 介面 IPushMessage，它的定義如下：

```
  IPushMessage = interface
  ['{10CBEBED-830F-4007-8674-1013B9698230}']
  procedure PushLatestMessage;
  procedure PushMessage(const sTitle : String; const sMessage :
String);
  procedure PushMessageToDevice(const sRegID : String;const sTitle :
String; const sMessage : String);

  function AddPushMessage(const sTitle : String; const sMessage :
```

```
String) : Integer;
  function DeletePushMessage(const iMID : Integer) : Integer;
  function CheckPushedMessageCount : Integer;
  function CheckUnPushedMessageCount : Integer;
  end;
```

IPushMessage 的 CheckPushedMessageCount 方法可查詢已推播訊息的數目而 CheckUnPushedMessageCount 則可查詢尚未推播訊息的數目。

接著讓我們在前面的範例專案中加入一個資料模組並加入 TFDConnection 和 TFDQuery 元件連結到 TBLPUSHMESSAGES 資料表：



再讓此資料模組實作 IPushMessage 介面：

```
  TdmPushMessage = class(TDataModule, IPushMessage)
    RIOdemodbConnection: TFDConnection;
    TblpushmessagesTable: TFDQuery;
    procedure DataModuleCreate(Sender: TObject);
    procedure DataModuleDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    //IPushMessage 介面
    procedure PushLatestMessage;
    procedure PushMessage(const sTitle : String; const sMessage :
String);
    procedure PushMessageToDevice(const sRegID : String;const
```

```
sTitle : String; const sMessage : String);

    function AddPushMessage(const sTitle : String; const sMessage :
String) : Integer;
    function DeletePushMessage(const iMID : Integer) : Integer;
    function CheckPushedMessageCount : Integer;
    function CheckUnPushedMessageCount : Integer;
  end;
```

目前暫時讓 **CheckPushedMessageCount** 和 **CheckUnPushedMessageCount** 都回傳 0：

```
function TdmPushMessage.CheckPushedMessageCount: Integer;
begin
  Result := 0;
end;

function TdmPushMessage.CheckUnPushedMessageCount: Integer;
begin
  Result := 0;
end;
```

版權所有 請勿翻印

現在回到前面的 **DUNITX** 範例，要測試剛才加入的資料模組我們可以在前面的 **TMyTestClass** 類別中加入 2 個測試方法：

```
    [Test]
    procedure TestPushedMessageCount;
    [Test]
    procedure TestUnPushedMessageCount;
```

接著在 **TMyTestClass** 的 **SetupFixture** 方法中建立 **TdmPushMessage** 物件準備測試它，再於 **TearDownFixture** 中釋放它：

```
procedure TMyTestClass.SetupFixture;
begin
  TDUnitX.CurrentRunner.Status('SetupFixture 被執行了!');
  dmPush := TdmPushMessage.Create(nil);
  TDUnitX.CurrentRunner.Status('TdmPushMessage 被建立了!');
end;
```

```
procedure TMyTestClass.TearDownFixture;
begin
  TDUnitX.CurrentRunner.Status('TearDownFixture 被執行了!');
  FreeAndNil(dmPush);
  TDUnitX.CurrentRunner.Status('TdmPushMessage 被釋放了!');
end;
```

現在我們就可以在 TestPushedMessageCount 和 TestUnPushedMessageCount 測試方法中使用建立的 dmPush 物件來呼叫 CheckPushedMessageCount 和 CheckUnPushedMessageCount 方法，並且用 DUNITX 框架中的 Assert 類別來檢查測試方法的回傳值來否符合我們的預期：

```
procedure TMyTestClass.TestPushedMessageCount;
begin
  TDUnitX.CurrentRunner.Status('TestPushedMessageCount 被執行
了!');
  Assert.AreEqual(0, dmPush.CheckPushedMessageCount);
end;

procedure TMyTestClass.TestUnPushedMessageCount;
begin
  TDUnitX.CurrentRunner.Status('TestUnPushedMessageCount 被執行
了!');
  Assert.AreEqual(0, dmPush.CheckUnPushedMessageCount);
end;
```

版權所有 請勿翻印

DUNITX 框架的 Assert 類別提供了許多的方法可檢查被測試方法的回傳值，例如 AreEqual，AreNotEqual，AreSame，AreNotSame，IsNull 和 IsNotNull 等等，讀者可檢視 DUnitX.TestFramework.Assert 類別的原始程式碼，每個方法的名稱都可說明它的意義。

現在就可以再次執行前面的 DUNITX 範例，下面是執行的結果我們可以看到 TestPushedMessageCount 和 TestUnPushedMessageCount 對目前 CheckPushedMessageCount 和 CheckUnPushedMessageCount 方法的實作都測試成功：

```
--------------------------------------------------
   Fixture : uMainTest.TMyTestClass
   --------------------------------------------------
     Running Fixture Setup Method : SetupFixture
       SetupFixture被執行了!
       TdmPushMessage被建立了!

     Test : uMainTest.TMyTestClass.TestPushedMessageCount
     --------------------------------------------------
     Executing Test : TestPushedMessageCount

       TestPushedMessageCount被執行了!
       Success.

     Test : uMainTest.TMyTestClass.TestUnPushedMessageCount
     --------------------------------------------------
     Executing Test : TestUnPushedMessageCount

       TestUnPushedMessageCount被執行了!
       Success.
```

我們可以這裡開始往下走，現在讓我們開發修改 CheckPushedMessageCount 和 CheckUnPushedMessageCount 方法成如下的程式碼，它們開始真正的要到 TBLPUSHMESSAGES 資料表中檢查已推播和尚未推播的訊息數目，所以它們分別呼叫 GetPushedMessageCount 和 GetUnPushedMessageCount 方法：

版權所有 請勿翻印

```
function TdmPushMessage.CheckPushedMessageCount: Integer;
begin
  Result := GetPushedMessageCount;
end;

function TdmPushMessage.CheckUnPushedMessageCount: Integer;
begin
  Result := GetUnPushedMessageCount;
end;
```

GetPushedMessageCount 和 GetUnPushedMessageCount 方法則是使用 SQL 命令在 TBLPUSHMESSAGES 中查詢資訊：

```
function TdmPushMessage.GetPushedMessageCount: Integer;
begin
  qryGeneral.SQL.Text := 'select count(*) from  TBLPUSHMESSAGES
```

```
where PUSHED = True';
  try
    qryGeneral.Active := True;
    Result := qryGeneral.Fields[0].AsInteger;
  finally
    qryGeneral.Active := False;
  end;
end;

function TdmPushMessage.GetUnPushedMessageCount: Integer;
begin
  qryGeneral.SQL.Text := 'select count(*) from  TBLPUSHMESSAGES
where PUSHED = False';
  try
    qryGeneral.Active := True;
    Result := qryGeneral.Fields[0].AsInteger;
  finally
    qryGeneral.Active := False;
  end;
end;
```

版權所有 請勿翻印

好了，現在我們改變了實作程式碼，因此也就需要再次執行測試方法看看加入的實作程式碼有沒有問題。由於在 TBLPUSHMESSAGES 資料表中已經推播的資料筆數是 1，另 2 筆資料是尚未推播的，因此我們使用 Assert 類別的 AreEqual 方法來檢查 GetPushedMessageCount 和 GetUnPushedMessageCount 方法正不正確：

```
procedure TMyTestClass.TestPushedMessageCount;
begin
  TDUnitX.CurrentRunner.Status('TestPushedMessageCount 被執行
了!');
  Assert.AreEqual(1, dmPush.CheckPushedMessageCount);
end;

procedure TMyTestClass.TestUnPushedMessageCount;
begin
  TDUnitX.CurrentRunner.Status('TestUnPushedMessageCount 被執行
了!');
  Assert.AreEqual(2, dmPush.CheckUnPushedMessageCount);
```

```
end;
```

使用 DUNITX 執行測試方法我們看到如下的結果，TestPushedMessageCount 和 TestPushedMessageCount 方法執行成功沒有錯誤代表我們實作的 GetPushedMessageCount 和 GetUnPushedMessageCount 方法是正確的：

```
    Test : uMainTest.TMyTestClass.TestPushedMessageCount
    ------------------------------------------------
    Executing Test : TestPushedMessageCount

      TestPushedMessageCount被執行了！
      Success.

    Test : uMainTest.TMyTestClass.TestUnPushedMessageCount
    ------------------------------------------------
    Executing Test : TestUnPushedMessageCount

      TestUnPushedMessageCount被執行了！
      Success.

   Running Fixture Teardown Method : TearDownFixture

     TearDownFixture被執行了！
     TdmPushMessage被釋放了！


Done testing.
Tests Found   : 2
Tests Ignored : 0
Tests Passed  : 2
Tests Leaked  : 0
Tests Failed  : 0
Tests Errored : 0
```

再讓我們看看如何使用 Assert 類別的其他方法，讓我們再於 TMyTestClass 類別中加入下面的 2 個測試方法：

```
    [Test]
    procedure TestIsTdmPushMessage;
    [Test]
    procedure TestIsIPushMessage;
```

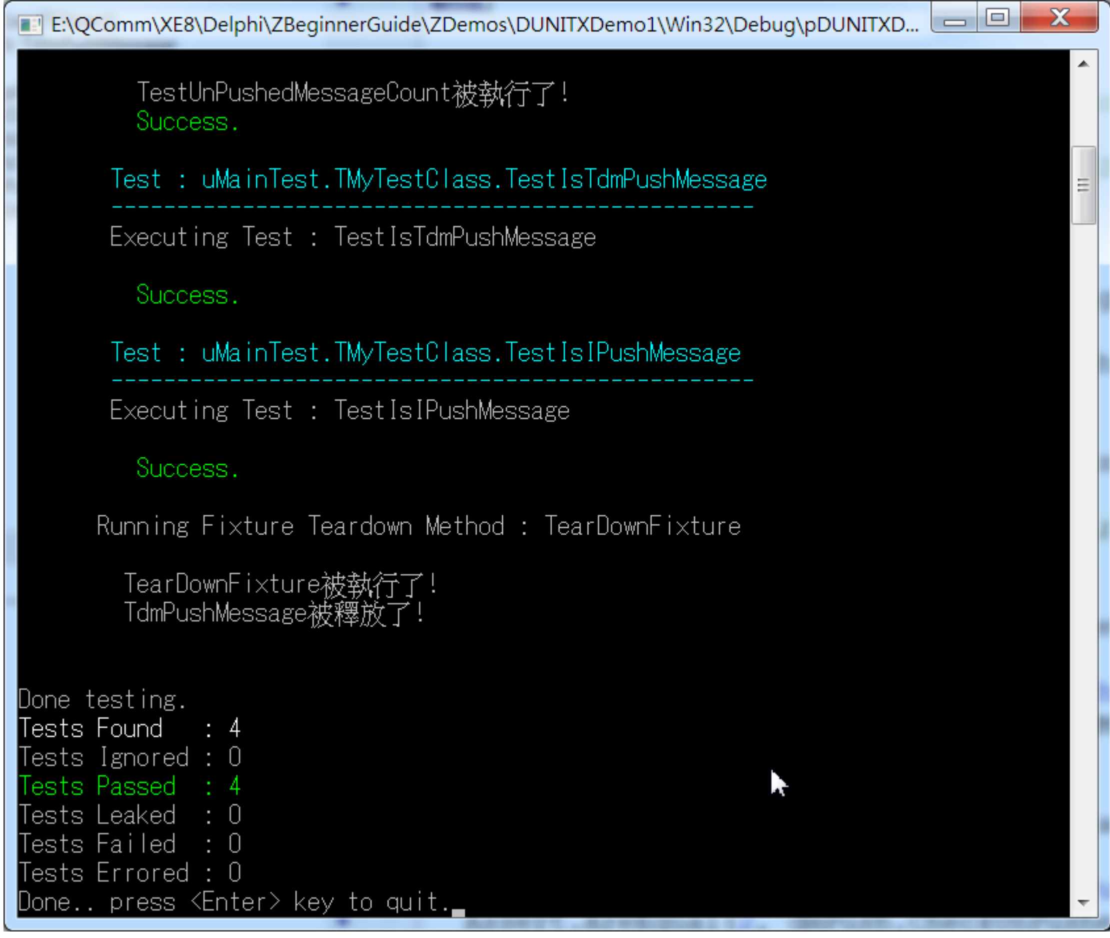
TestIsTdmPushMessage 測試方法要檢查 SetupFixture 方法建立的物件是不是 TdmPushMessage 類別物件，而 TestIsIPushMessage 則要檢查 TdmPushMessage 類別有沒有真的實作 IPushMessage 介面。

TestIsTdmPushMessage 和 TestIsIPushMessage 測試方法實作如下。TestIsTdmPushMessage 呼叫了 Assert 類別的 IsType 泛型方法來測試 dmPush 物件是否是 TdmPushMessage 類別物件。而 TestIsIPushMessage

呼叫了 Assert 類別的 Implements 泛型方法來測試 dmPush 物件是否實作了實作 IPushMessage 介面：

```
procedure TMyTestClass.TestIsIPushMessage;
begin
  Assert.Implements<IPushMessage>(dmPush);
end;

procedure TMyTestClass.TestIsTdmPushMessage;
begin
  Assert.IsType<TdmPushMessage>(dmPush);
end;
```

執行範例 DUNITX 程式可以看到 TestIsIPushMessage 和 TestIsTdmPushMessage 都成功通過測試：



現在讓我們以一個實際的例子來看看如何使用 DUNITX 框架，讓我們撰寫程式碼在 TBLPUSHMESSAGES 資料表中加入一筆新的要推播的資料，但問題是新加入的資料會不會影響我們已經存在程式碼的正確性呢？

因此讓我們先撰寫一個新的測試方法來測試稍後即將撰寫的程式碼的正確性：

```
  [Test]
  procedure TestAddPushMessage;
```

TestAddPushMessage 我實作很簡單，它呼叫 dmPush 物件的 CheckUnPushedMessageCount 測試方法先取得 TBLPUSHMESSAGES 資料表中尚未推播訊息的筆數並儲存在 iBefore 變數中，再呼叫 dmPush 物件的 AddPushMessage 方法實際在 TBLPUSHMESSAGES 資料表中加入一筆新的要推播的資料，最後再次呼叫 dmPush 物件的 CheckUnPushedMessageCount 測試方法先取得最新的尚未推播訊息的筆數並和 iBefore 變數值加 1 比較看看是否一樣以測試新加入資料筆數的正確性：

版權所有 請勿翻印

```
procedure TMyTestClass.TestAddPushMessage;
var
  iResult : Integer;
  iBefore : Integer;
begin
  iBefore := dmPush.CheckUnPushedMessageCount;
  iResult := dmPush.AddPushMessage('DUNTX 測試訊息', 'DUNTX 測試訊息'
+ DateTimeToStr(Now));
  Assert.AreEqual(iBefore + 1, dmPush.CheckUnPushedMessageCount);
end;
```

現在回到範例資料模組中，先實作一個工具方法 DateTimeToSeconds，定可把 TDateTime 的數值轉換成秒數：

```
function DateTimeToSeconds(const ADateTime: TDateTime): Integer;
var
  LTimeStamp: TTimeStamp;
begin
  LTimeStamp := DateTimeToTimeStamp(ADateTime);
  Result := LTimeStamp.Date;
  Result := Abs((Result * SecsPerDay) + LTimeStamp.Time);
```

```
end;
```

接著實作 AddPushMessage 方法，它使用 FireDAC 的 TFDQuery 元件在 TBLPUSHMESSAGES 資料表中加入一筆新的資料並使用呼叫 DateTimeToSeconds 方法的回傳值做為此筆資料的鍵值：

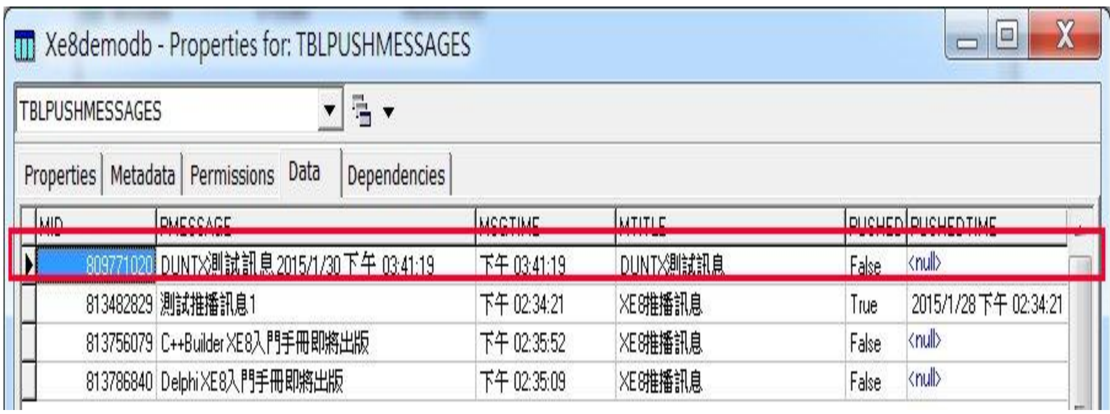
```
function TdmPushMessage.AddPushMessage(const sTitle, sMessage:
String): Integer;
begin
  OpenPushMessageTable;
  TblpushmessagesTable.Insert;
  Result := DateTimeToSeconds;
  TblpushmessagesTable.FieldByName('MID').Value := Result;
  TblpushmessagesTable.FieldByName('PMESSAGE').Value := sMessage;
  TblpushmessagesTable.FieldByName('MTITLE').Value := sTitle;
  TblpushmessagesTable.FieldByName('PUSHED').Value := False;
  TblpushmessagesTable.FieldByName('MSGTIME').Value := NOW;
  TblpushmessagesTable.Post;
end;
```

版權所有 請勿翻印

編譯並執行範例測試程式就可以看到 TestAddPushMessage 測試方法通過測試，這代表新加入 TBLPUSHMESSAGES 資料表的資料應該是正確的，更重要的是新實作的 AddPushMessage 方法也沒有影響到以前程式碼的正確性，因為所有的原先的測試方法也仍然都通過測試：

```
      Test : uMainTest.TMyTestClass.TestAddPushMessage
      ------------------------------------------------
      Executing Test : TestAddPushMessage

        Success.

     Running Fixture Teardown Method : TearDownFixture

       TearDownFixture被執行了!
       TdmPushMessage被釋放了!


Done testing.
Tests Found   : 5
Tests Ignored : 0
Tests Passed  : 5
Tests Leaked  : 0
Tests Failed  : 0
Tests Errored : 0
Done.. press <Enter> key to quit.
```

用 InterBase 工具開啟 TBLPUSHMESSAGES 資料表可以看到如下的結果，資料果然正確加入其中了：

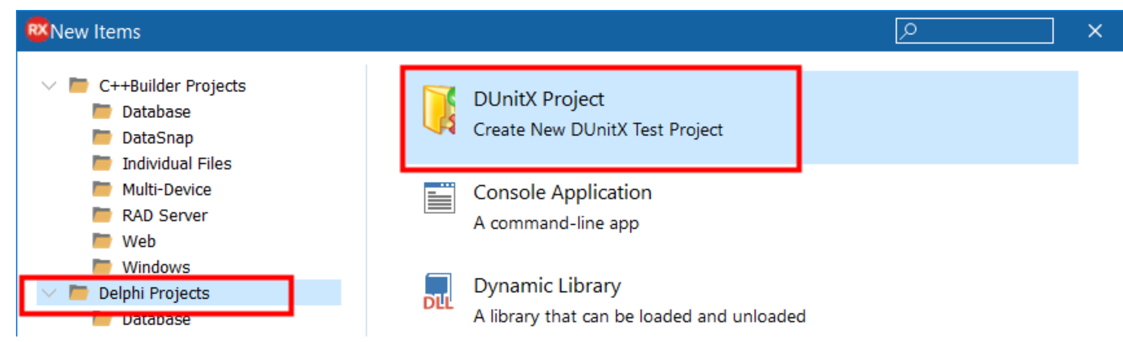


讀者應該對於如何使用 DUNITX 有基本的瞭解了，現在在 Delphi RIO 中我們不需要再去自行下載 DUNITX 框架，因為 Delphi 已經把它整合在 IDE 中了，下面將介紹如何在 Delphi IDE 中使用 DUNITX 來執行上面的範例 DUNITX 單元測試。

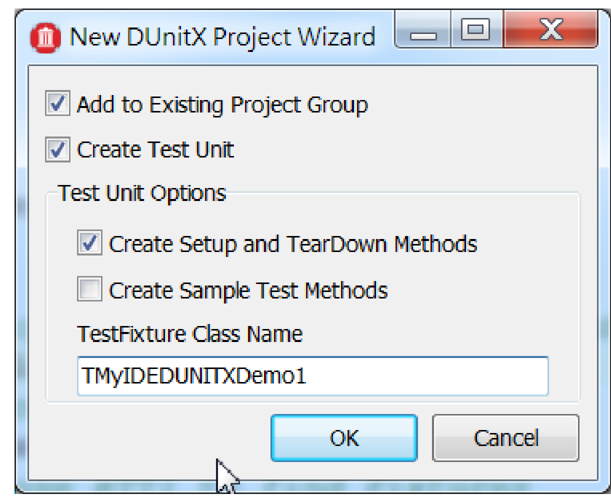
14-3-3 使用 DUNITX 單元測試框架

版權所有 請勿翻印

在 Delphi IDE 中使用 DUNITX 框架非常簡單，只需點選 File | New | Other...選項再於 Delphi Projects | DUnitX 專案中可看到 DUnitX Project 和 DUnitX Unit 圖像。DUnitX Project 圖像代表要建立 DUNITX 測試專案而 DUnitX Unit 圖像代表要建立測試程式式單元，現在讓我們點選 DUnitX Project 圖像建立 DUNITX 測試專案：



接著 IDE 會顯示下面的對話盒詢問您是否要建立測試程式式單元，是否要建立 Setup 和 TearDown 方法，是否要建立簡單的範例測試方法以及為測試類別取一個名稱等資訊：



點選上面的選項和 OK 按鈕後 IDE 便會產生如下的程式碼，這些程式碼現在讀者應該都瞭解它們的意義了：

Welcome Page | pDUNITXDemo1 | Unit3

```
unit Unit3;

interface
uses
  DUnitX.TestFramework;

type

  [TestFixture]
  TMyIDEDUNITXDemo1 = class(TObject)
  public
    [Setup]
    procedure Setup;
    [TearDown]
    procedure TearDown;
  end;

implementation

procedure TMyIDEDUNITXDemo1.Setup;
begin
end;

procedure TMyIDEDUNITXDemo1.TearDown;
begin
end;


initialization
  TDUnitX.RegisterTestFixture(TMyIDEDUNITXDemo1);
end.
```

現在讀者就可以直接在產生的測試程式單元中撰寫測試程式碼了。

14-4 App Tethering

App Tethering 技術允許開發人員使用軟體連結 PC 和各種用戶端的設備，開發人員可藉由 WiFi 或是藍牙連結各種不同的硬體：

App Tethering 技術在發展之初是希望幫助傳統 PC 開發人員能夠把 Windows 應用程式的功能移植到手機中，後來才逐漸發展成可連結各種不同的硬體的軟體技術。

版權所有 請勿翻印

藉由 App Tethering 技術在不同硬體平臺中的軟體可以：

1. 互相遠端控制執行行動命令
2. 傳遞和共用資料
3. 發展點對點的執行控制模式

App Tethering 的功能在 XE6 中即開始出現，一開始 App Tethering 只提供在同一個子網路區域中設備的連結，到了 XE7 App Tethering 可藉由直接提供 IP 位址連結也允許使用藍牙協定連結，到了 RIO App Tethering 提供了更完整的連結功能並且增進了 App Tethering 的執行效率。

由於 App Tethering 允許不同硬體平臺中的軟體不同硬體平臺中的軟體，因此使用者可以把手機 App 的資料及時傳遞給 PC 中的應用程式，或是反之。因此 App Tethering 提供了 2 種方式傳遞資料：

傳遞資料方式	說明
共用資源	使用資源方式共用資料，並提供資料更新的能力

以暫時資源一次傳送資料	從一平臺 App 一次傳遞資料到另一平臺的 App

由於 Delphi 本身提供了數個 App Tethering 的範例供開發人員參考，但如果沒有一些基本的概念的話可能不太容易瞭解，因此在本小節中我們將使用一個範例來說明如何使用 App Tethering 技術，這個範例就是：BDShoppingList。

基本上 App Tethering 使用了類似藍牙的概念，也就是說要使用 App Tethering 功能，開發人員需要使用一個主要功能的元件，再使用另一個 Profile 元件來完成應用程式要提供的功能。因此這個主要功能元件就是 TetheringManager，而 Profile 元件就是 TetheringAppProfile。下面的表格說明了這 2 個元件的功能：

組件	說明
TetheringManager	負責提供最基本的功能，即偵測其他的 TetheringManager 並連結和配對遠端的 TetheringManager 組件
TetheringAppProfile	提供 Tethering 的執行功能，包含執行遠端命令，分享資源和傳送資料等

現在我們就可以開始說明 BDShoppingList 是如何工作的。請在 Delphi IDE 中開啟下面目錄中的

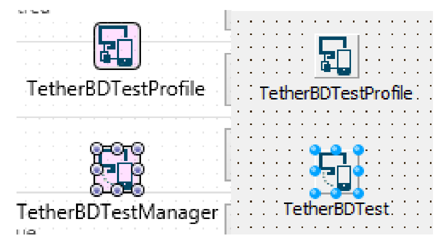
版權所有 請勿翻印

```
C:\Users\Public\Documents\Embarcadero\Studio\20.0\Samples\Object
Pascal\RTL\Tethering\BDShoppingList
```

TetherShopping.groupproj 專案群組。這個項目群組中包含了一個 Windows 端的 VCL 應用程式和一個能在 Android/iOS 平臺中執行的 App。在下面的小節中將說明 Windows 端和 Android/iOS 平臺中的 App 如何藉由 App Tethering 技術共同執行運作。

14-4-1 手機端 TetherDBClient 如何工作

開啟 TetherDBClient 專案的主表單，您會看到下面左方的 TetheringManager 和 TetheringAppProfile 組件，如何此時再開啟 Windows 端的 VCL 應用程式的主表單就會看到到下方右邊同樣的 2 個組件：



使用 App Tethering 技術的 2 方應用程式各自需要這 2 個元件以偵測和連結對方。

TetheringManager 元件需要註冊並使用 TetheringAppProfile 元件以啟動功能，因此在物件檢視器中可以看到上面的 2 個 TetheringAppProfile 元件都設定了它的 Manager 都連結到 TetheringManager 元件：

Object Inspector

TetherBDTestProfile TTetheringAppProfile

Properties | Events

Property	Value
Actions	(TActionCollection)
Enabled	☑ True
Group	TetherBDTestGroup
⊞ LiveBindings Designer	LiveBindings Designer
⊞ Manager	TetherBDTestManager

Object Inspector

TetherBDTestProfile TTetheringAppProfile

Properties | Events

Property	Value
Actions	(TActionCollection)
Enabled	☑ True
Group	TetherBDTestGrou
⊞ LiveBindings Designer	LiveBindings Designe
⊞ Manager	TetherBDTest

版權所有 請勿翻印

設定好上面的元件之後要讓 App Tethering 能夠工作，那麼這 2 方一定要有一方必須執行偵測和連結對方的工作，這個工作在此範例中是由手機端執行的。

偵測和連結

要偵測和連結對方，開發人員可以呼叫 TetheringManager 元件的 AutoConnect 或是 DiscoverManagers 方法：

```
procedure AutoConnect(Timeout: Cardinal; const ATarget: string =
''); overload;
procedure AutoConnect(const ATarget: string = ''); overload;
procedure AutoConnect(Timeout: Cardinal; const ATargetList:
TTetheringTargetHosts); overload;
procedure AutoConnect(const ATargetList: TTetheringTargetHosts);
overload;

procedure DiscoverManagers(Timeout: Cardinal; const ATarget: string
= ''); overload;
procedure DiscoverManagers(const ATarget: string = ''); overload;
```

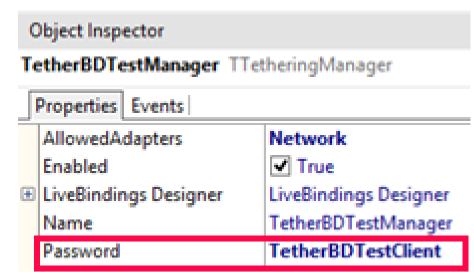
```
procedure DiscoverManagers(Timeout: Cardinal; const ATargetList:
TTetheringTargetHosts); overload;
procedure DiscoverManagers(const ATargetList:
TTetheringTargetHosts); overload;
```

這 2 個方法都可以接受一個超時時間參數，此內定的參數值是 1500ms，在這 2 個方法于超時時間參數值到達之後就會觸發 OnEndAutoConnect 或 OnEndManagersDiscovery 事件，在這 2 個事件處理函式中您會收到一列可十 2 結的遠端 TetheringManager 元件，您可以選擇其中您的 App 的連結化物件。因此在 TetherDBClient 專案的中您會看到此範例使 AutoConnect 方法偵測並連結對方共在 OnEndAutoConnect 事件中檢查偵測到的遠端 TetheringManager 元件：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  tmCheckConnection.Enabled := true;
  TetherBDTestManager.AutoConnect;
end;

procedure TForm1.TetherBDTestManagerEndAutoConnect(Sender:
TObject);
begin
  CheckRemoteProfiles;
end;
```

版權所有 請勿翻印

您也可以使用密碼來保護合法的連結：

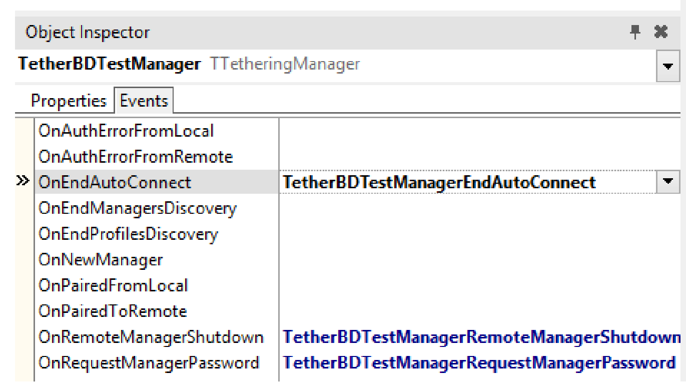


當使用密碼時在連結遠端 TetheringManager 元件時遠端 TetheringManager 元件會觸發 OnRequestManagerPassword 事件要求提供登入密碼，因此 TetherDBClient 專案在它的 OnRequestManagerPassword 事件中提供如下的登錄連結密碼：

```
procedure TForm1.TetherBDTestManagerRequestManagerPassword(
```

```
  const Sender: TObject; const RemoteIdentifier: string; var
Password: string);
begin
  Password := 'TetherBDTest';
end;
```

而如果使用的密碼錯誤就會觸發 OnAuthErrorFromRemote 事件。



如果成功登錄並連結，那麼接著會觸發 OnPairedToRemote 事件最後再觸發 OnEndProfilesDiscovery 或 OnEndAutoConnect 事件。

版權所有 請勿翻印

而在遠端一方，例如下面的 Windows 端 TetherDatabase 項目中，如果 TetherDBClient 使用錯誤的密碼連結就會觸發 OnAuthErrorFromLocal 事件，但如果密碼正確就會觸發 OnPairedFromLocal 事件。

瞭解了 TetherDBClient 端如何工作之後再讓我們說明 Windows 端。

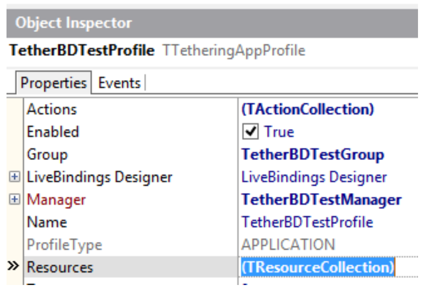
14-4-2 Windows 端 TetherDatabase 如何工作

Windows 端 TetherDatabase 專案提供了手機端 TetherDBClient App 一個遠端命令讓手機端可執行在遠端 Windows 的 TetherDatabase 應用程式中的功能，TetherDatabase 專案也提供了手機端 TetherDBClient App 一個共用的資源讓手機端 TetherDBClient App 可擷取遠端 Windows 的 TetherDatabase 應用程式的執行結果。因此 Windows 端 TetherDatabase 項目提供了：

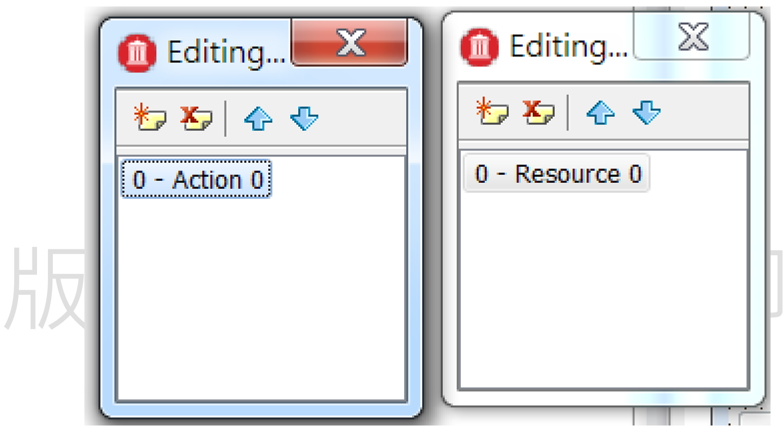
1. 一個輸出執行命令

2. 一個共用資源

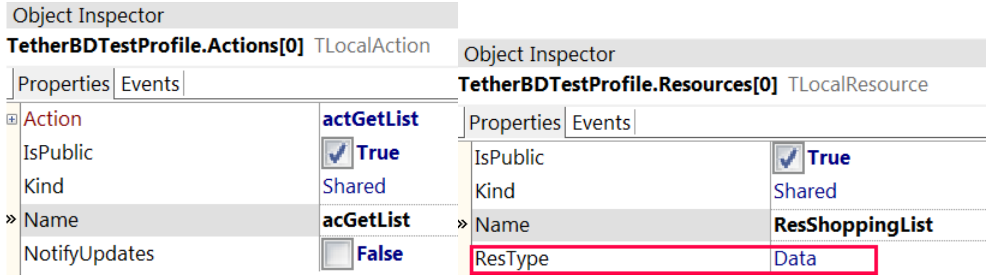
要提供輸出執行命令和共用資源，開發人員可以藉由 TetheringAppProfile 元件的 Actions 和 Resources 特性，例如在 TetherDatabase 專案中的 TetherBDTestProfile 元件：



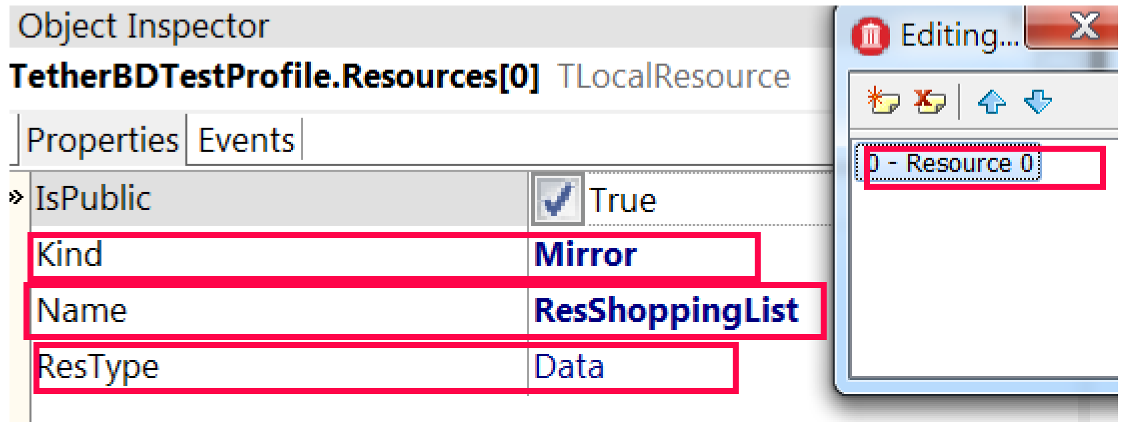
在它的 Actions 和 Resources 特性中提供了一個命令物件 TAction 和一個共用資源物件 TLocalResource：



命令物件在被遠端手機 App 呼叫執行之後就會把執行結果以共用資源方式讓雙方可存取，因此共用資源物件 TLocalResource 的 ResType 的型態是設定為 Data，Kind 特性設定為”Shared”，而且共用資源是命名為”ResShoppingList”：

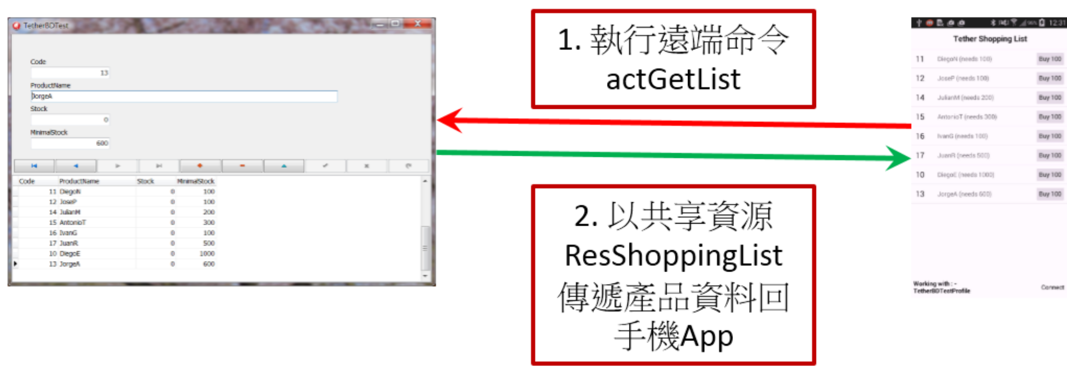


因此在手機端 TetherDBClient 專案中的 TetherBDTestProfile 元件其 Resources 特性中也要加入一個命名為”ResShoppingList”的共用資源物件但其 Kind 特性設定為”Mirror”：



14-4-3 Windows 端和手機如何共同工作

現在就可以說明 Windows 端和手機如何使用 App Tethering 技術共同工作了，請先在 Windows 中執行 TetherDatabase 專案再於手機中執行 TetherDBClient 專案，點選 TetherDBClient 專案右下方的 Connect 按鈕就可以看到如下的執行結果：



手機中的 App 可以藉由 App Tethering 技術看到需要購買那麼產品，為什麼？這是因為…

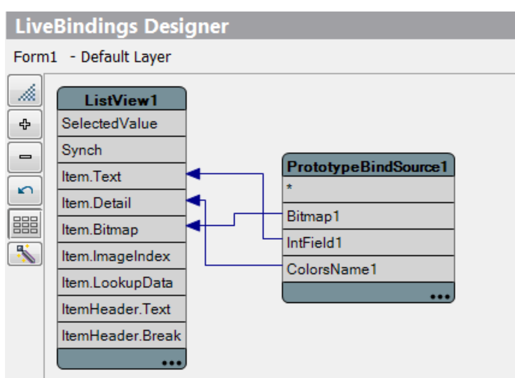
TetherDBClient 專案端

在手機 TetherDBClient 專案端的 Connect 按鈕被點選後就會去執行遠端的命令 actGetList：

```
if not FIsConnected then
  actGetList.Execute;
```

在遠端(Windows 端)執行了此命令之後就會把執行結果儲存在共用資源中，接著 App Tethering 就會觸發 TetherDBClient 專案端的 OnResourceReceived

事件，TetherDBClient 專案端再使用 Live Binding 技術把產品資料顯示在手機中。



TetherDatabase 專案端

在 Windows 端的 TetherDatabase 專案的 actGetList 命令被手機端呼叫後就會執行並把執行結果存放在共用資源中等待手機端去存取執行結果：

```
  if LShoppingList.Count > 0 then
    TetherBDTestProfile.Resources.Items[0].Value :=
LShoppingList.DelimitedText
  else
    TetherBDTestProfile.Resources.Items[0].Value := 'NONE';
```

版權所有 請勿翻印

接著手機端可以決定要採購庫存不足的產品，一但使用者在手機端點選採購特定的產品後，就會執生下面的流程：

TetherDBClient 專案端

在手機 TetherDBClient 專案端的 OnListView1ButtonClick 事件中藉由呼叫 SendString 方法把要採購的產品同稱傳遞給 Windows 端：

```
procedure TForm1.ListView1ButtonClick(const Sender: TObject;
  const AItem: TListViewItem; const AObject:
TListItemSimpleControl);
begin

TetherBDTestProfile.SendString(TetherBDTestManager.RemoteProfile
s.Items[0], 'Buy item', AItem.Text);
end;
```

TetherDatabase 專案端

此時 Windows 端的 TetherDatabase 專案就會觸發 OnResourceReceived 事件，接著就執行採購和更新資料的工作：

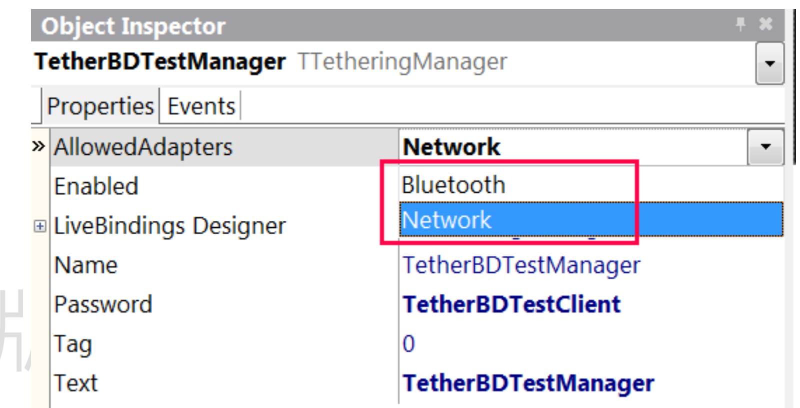
版權所有 請勿翻印

```
procedure TForm2.TetherBDTestProfileResourceReceived(const
Sender: TObject;
  const AResource: TRemoteResource);
var
  PID: Integer;
begin
  if AResource.ResType = TRemoteResourceType.Data then
  begin
    PID := StrToInt(AResource.Value.AsString);
    CDSProducts.First;
    while not CDSProducts.Eof do
    begin
      if CDSProductsCode.Value = PID then
      begin
        CDSProducts.Edit;
        CDSProductsStock.Value := CDSProductsStock.Value + 100;
        CDSProducts.Post;
        Break;
      end;
      CDSProducts.Next
```

```
    end;
  end;
end;
```

瞭解了 BDShoppingList 專案群組如何工作之後您就應該掌握了 App Tethering 技術的基本觀念了，您可以再檢視其他 App Tethering 的範例專案學習如何傳遞圖形資料或是串列流的資料。

在 XE7 之後 App Tethering 不但可以使用 Wi-Fi 連結，也可以使用改用藍牙連結，開發人員只需要設定 TTetheringManager 元件的”AllowedAdapters”特性即可，設定 AllowedAdapters 特性值為 Network 代表使用 Wi-Fi，設定 Bluetooth 代表使用藍牙連結：



另外 XE7 之後也允許開發人員直接連結特定的 IP 位址，例如在呼叫 TTetheringManager 元件的 AutoConnect 方法時直接使用一個 IP 位址參數即可：

```
TetherBDTestManager.AutoConnect(2000,'192.168.0.27');
```

14-5 藍牙開發

Delphi 從 XE7 開始便支援開發傳統藍牙和低耗電藍牙 BLE 的功能，並且提供了 TBlueToothLE 元件封裝低耗電藍牙 BLE 的功能，但對於傳統藍芽則只提供類別支援並沒有提供封裝傳統藍牙的元件。但到了 RIO Delphi 終於同時提供了 TBlueTooth 和 TBlueToothLE 這 2 個元件來封裝傳統藍牙和低耗電藍牙 BLE 功能。

要開發藍牙應用程式您必須瞭解不同平臺對於藍牙技術的支援差異，下面的表格列出了 4 個平臺對傳統藍牙和低耗電藍牙 BLE 的支持程度：

	傳統藍牙	低耗電藍牙BLE
Android	✔	✔(Android 4.3(含)以上版本)
iOS	✖	✔(iPhone 4s+ and iPad2+)
Windows	✔	Windows 8+
Mac	✔	✔

在本小節中我們將說明如何使用 TBlueTooth 元件來開發藍牙 App。

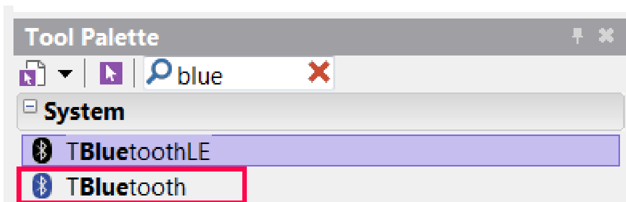
使用 TBlueTooth 元件

開發藍牙功能的 App 基本上開發人員要執行下列的步驟：

1. 搜尋附近的藍牙設備
2. 搜尋藍牙設備提供的服務
3. 配對藍牙設備
4. 建立配對藍牙設備之間的通訊管道
5. 傳遞資料

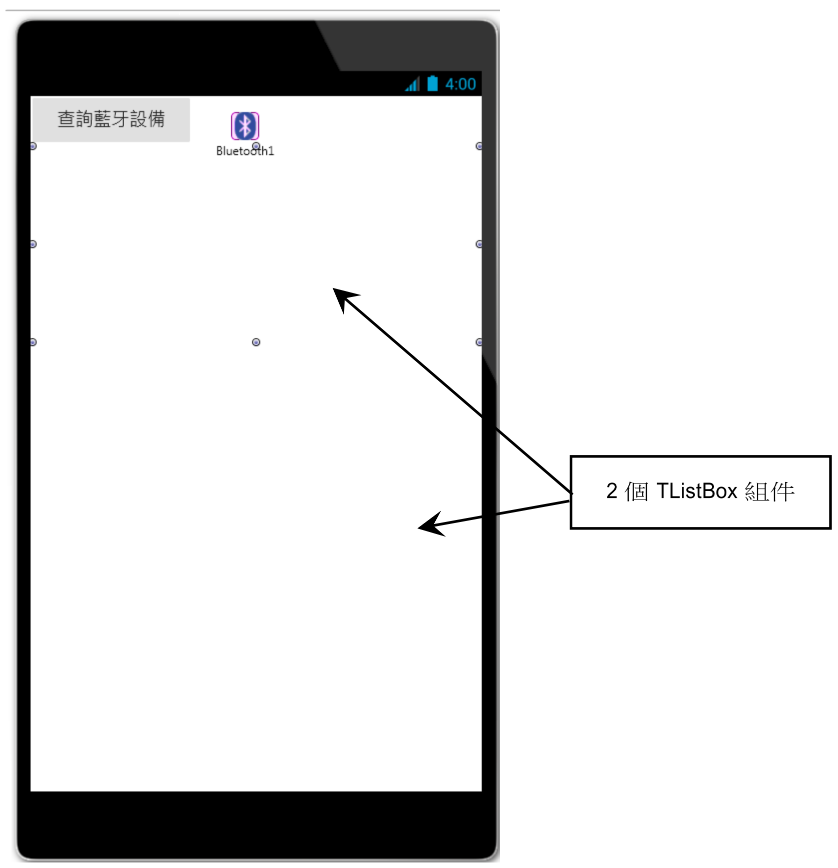
版權所有 請勿翻印

有了下面 TBlueTooth/TBlueToothLE 元件之後這些工作就非常簡單了：



現在就讓我開發一個簡單的範例藍牙 App 來說明如何完成上面的開發步驟。

首先建立一個 Multi-Device 專案並在主表單中加入如下的元件：



版權所有 請勿翻印

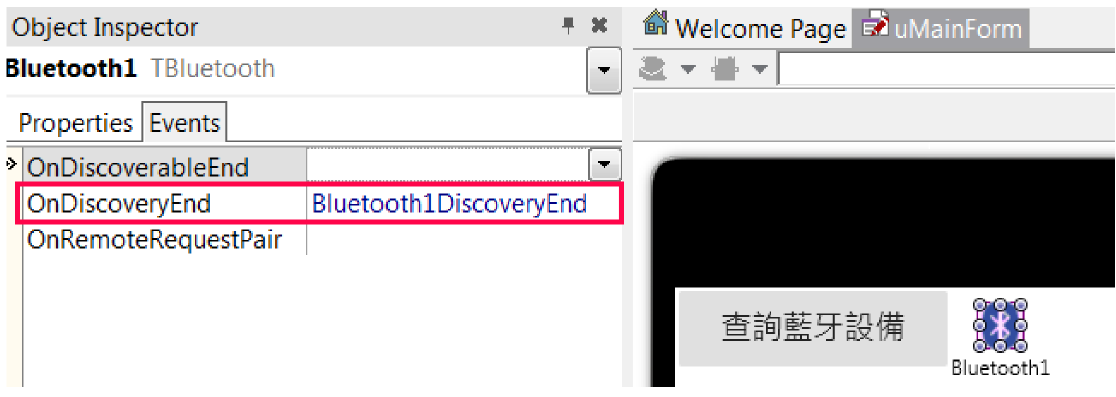
要查詢藍牙設備很簡單，只要呼叫 TBlueTooth 元件的 DiscoverDevices 方法即可，DiscoverDevices 接收一個搜尋藍牙設備時間的參數：

```
procedure DiscoverDevices(ATimeout: Integer);
```

因此要實作主表單”查詢藍牙設備”按鈕的功能只需要使用下面的程式碼讓範例 App 使用 10 秒的時間查詢藍牙設備：

```
procedure TForm3.Button1Click(Sender: TObject);
begin
  Bluetooth1.DiscoverDevices(10000);
end;
```

在 TBlueTooth 組件查詢完畢之後它會觸發 OnDiscoveryEnd 事件：



在 OnDiscoveryEnd 事件會傳入所有找到的藍牙設備參數 ADeviceList，它的型態是 TBluetoothDeviceList。TBluetoothDeviceList 的 Items 特性是 TBluetoothDevice 類別物件，每一個 TBluetoothDevice 類別物件代表一個搜尋到的藍牙設備。TBluetoothDevice 類別物件即可提供藍牙設備的名稱，位址等資訊，而且我們也可以使用它來獲得此藍牙設備提供的服務資訊。

因此在 OnDiscoveryEnd 事件中我們即可以取出每一個 TBluetoothDevice 類別物件並顯示此它的名稱到主表單的 TListBox 中：

版權所有 請勿翻印

```
procedure TForm3.Bluetooth1DiscoveryEnd(const Sender: TObject;
  const ADeviceList: TBluetoothDeviceList);
var
  iCount: Integer;
begin
  lbBTDevices.Items.Clear;
  for iCount := 0 to ADeviceList.Count - 1 do
  begin
    lbBTDevices.Items.Add(ADeviceList.Items[iCount].DeviceName);
  end;
  FDiscoverDevices := ADeviceList;
end;
```

現在可以準備執行範例 App 來看看它是否能搜尋藍牙設備，但在編譯和執行之前必須先開啟範例 App 存取藍牙的許可權。請點選 Project|Options...選項並如下圖勾選 Bluetooth 和 Bluetooth admin 許可權：

下面是執行開發到現在的範例 App 畫面，我們可以看到範例 App 是可以找到附近的藍牙設備：

版權所有 請勿翻印

現在再讓我們實作在找到藍牙設備之後如果在 TListBox 中點選這個藍牙設備就可以找到它提供的服務資訊，這可以在 TListBox 的 OnItemClick 事件中先找到被點選的藍牙設備，然後呼叫 DisplayDeviceServices 方法：

```
procedure TForm3.lbBTDevicesItemClick(const Sender:
TCustomListBox;
  const Item: TListBoxItem);
begin
  DisplayDeviceServices(FDiscoverDevices.Items[Item.Index]);
end;
```

DisplayDeviceServices 方法藉由呼叫 TBlueTooth 元件的 GetServices 方法並傳入被點選的藍牙設備做為參數：

```
    function GetServices(const ADevice: TBluetoothDevice):
TBluetoothServiceList;
```

GetServices 方法會回傳藍牙設備提供的所有服務，在回傳的 TBluetoothServiceList 對象中每一個藍牙設備提供的服務都以一個 TBluetoothService 物件代表，因此在 DisplayDeviceServices 方法中我們就

可以取出每一個 TBluetoothService 物件並顯示此服務名稱到主表單的第 2 個 TListBox 中：

```
procedure TForm3.DisplayDeviceServices(aDevice :
TBluetoothDevice);
var
  sl: TBluetoothServiceList;
  iCount: Integer;
begin
  BTDeviceServices.Items.Clear;
  sl := Bluetooth1.GetServices(aDevice);
  for iCount := 0 to sl.Count - 1 do
  begin
    BTDeviceServices.Items.Add(sl.Items[iCount].Name);
  end;
```

再執行範例 App 並點選找到的藍牙設備就可以如下圖看到每個藍牙設備都提供了不同的服務：

接下來讓我們實作第 2 個步驟”配對藍牙設備”。

要實作配對藍牙設備非常簡單，只需要呼叫 TBlueTooth 元件的 Pair 方法即可，而 UnPair 方法則是解除配對：

```
function Pair(const ADevice: TBluetoothDevice): Boolean;
function UnPair(const ADevice: TBluetoothDevice): Boolean;
```

Pair 方法也是接受一個代表要配對的藍牙設備的 TBluetoothDevice 物件。因此讓我們在主表單中加入一個配對按鈕，當使用者在主表單的 TListBox 中選擇了一個搜尋到的藍牙設備後就可以點選配對按鈕來進行藍牙設備之間的配對工作。

下面是配對按鈕的 OnClick 實作程式碼，它呼叫了 Pair 方法並且把點選的 TBluetoothDevice 物件傳入做為參數。如果 Pair 方法執行成功就會回傳 True 值，那麼我們就把成功配對的藍牙設備名稱加入到成功配對的 TComboBox 組件中：

```
procedure TForm3.btnPairClick(Sender: TObject);
begin
  if
(Bluetooth1.Pair(FDiscoverDevices.Items[lbBTDevices.ItemIndex]))
then
  begin

cbPairDevices.Items.Add(FDiscoverDevices.Items[lbBTDevices.ItemI
ndex].DeviceName);
    cbPairDevices.ItemIndex :=
cbPairDevices.Items.IndexOf(FDiscoverDevices.Items[lbBTDevices.I
temIndex].DeviceName);
  end;
end;
```

版權所有 請勿翻印

再次執行範例程式碼，在下面的畫面中可以看到我們要在範例 App 執行的 Samsung S4 中配對 HTC M8 手機：

點選配對按鈕之後可以看到 Samsung S4 顯示的配對要求：

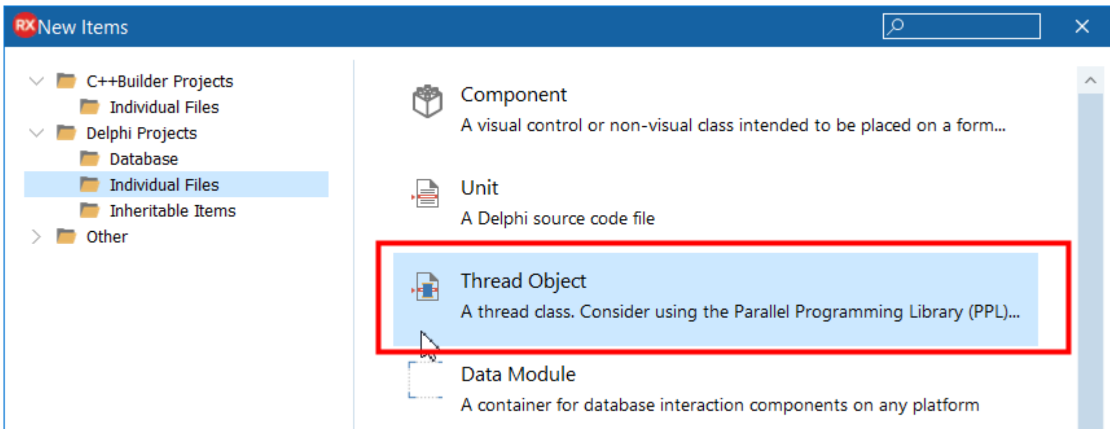
按下確定之後可以看到成功和 HTC M8 配對成功了：

完成了配對步驟之後我們就可以開始下一個步驟”建立配對藍牙設備之間的通訊管道”，一旦通訊管道建立成功之後就可以在藍牙設備之間傳遞資料了。

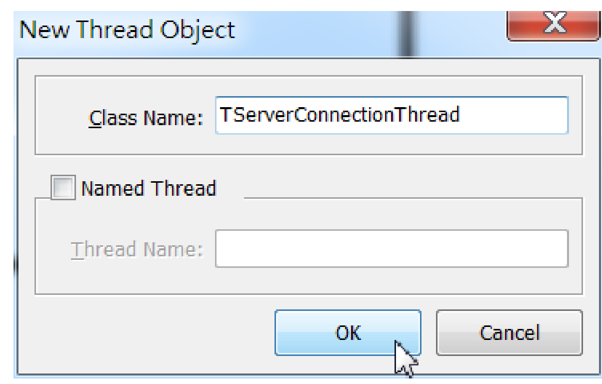
版權所有 請勿翻印

要建立通訊管道並且傳遞資料我們需要使用一個獨立的執行緒，如此一來才不會影響主執行緒的執行。先讓我們在主表單中加入 3 個按鈕，1 可輸入資料的 **TEdit** 元件和一個 **TMemo** 元件，如下所示：

接著在專案中建立一個 Thread Object：



取名為 TServerConnectionThread 並且在主表單中使用它：



要在配對的藍牙設備之間通訊，開發人員需要使用下面的步驟：

1. 一方的藍牙設備必須以伺服端建立通訊管道

2. 另一方的藍牙設備必須以用戶端建立通訊管道

3. 伺服端的藍牙設備先建立 TBluetoothServerSocket 物件，並使用它監聽用戶端的連結請求，一旦伺服端藍牙設備接受連結便可取得 TBluetoothSocket 物件。伺服端取得 TBluetoothSocket 物件之後就可以使用它讀取用戶端傳送來的資料

4. 用戶端藍牙設備建立 TBluetoothSocket 物件並使用它傳遞資料給伺服端的藍牙設備

版權所有 請勿翻印

瞭解了如何建立通訊管道和傳遞資料的原理之後我們就可以開始實作了，首先讓我們實作伺服端。

在伺服端中我們要使用前面建立的 TServerConnectionThread 物件中於一個獨立的執行緒來監聽用戶端的連結請求並讀取用戶端傳送來的資料。因此在主表單的”建立通訊管道’按鈕的 OnClick 事件處理函式中呼叫了 CreateBTTextService 方法來堤行這些工作：

```
procedure TfmMainForm.btnCreateTextServiceClick(Sender: TObject);
begin
  CreateBTTextService;
end;
```

CreateBTTextService 方法主要是建立 TServerConnectionThread 物件並藉由 TBluetoothServerSocket 物件取得可讀取資料的 TBluetoothSocket 物件。要取得 TBluetoothServerSocket 物件我們可以藉由 TBlueTooth 元件的 CurrentAdapter 特性先得 TBluetoothAdapter 物件，再呼叫

TBluetoothAdapter 物件的 CreateServerSocket 方法取得 TBluetoothServerSocket 物件：

```
function CreateServerSocket(const AName: string; const AUUID:
TGUID; Secure: Boolean): TBluetoothServerSocket;
```

CreateServerSocket 方法接受 3 個參數，第 1 個是此通訊管道的名稱，第 2 個參數是代表此通訊管道的 GUID 值，最後一個參數代表是否要建立一個安全的通訊管道。為了傳遞給 CreateServerSocket 方法第 1 和第 2 個參數，範例 App 定義了下面的常數(在 IDE 中 GUID 值可使用 ctrl-shift-g 產生)：

```
Const
  ServiceName = '範例藍牙傳遞資料服務';
  ServiceGUI = '{9827B0D2-66FF-4B30-A3C3-59F38ED59B61}';
```

下面就是 CreateBTTextService 方法的實作程式碼，004 行建立 TServerConnectionThread 物件，005 行建立 TBluetoothServerSocket 物件並指定給 TServerConnectionThread 物件的 ServerSocket 特性以便 TServerConnectionThread 物件使用來接受用戶端的連絡請求，最後在 006 行啟示執行 TServerConnectionThread 物件代表的獨立執行緒：

版權所有 請勿翻印

```
001   procedure TfmMainForm.CreateBTTextService;
002   begin
003     try
004       scThread := TServerConnectionThread.Create(True);
005       scThread.ServerSocket :=
Bluetooth1.CurrentAdapter.CreateServerSocket(ServiceName,
StringToGUID(ServiceGUI), False);
006       scThread.Start;
007       mmMessages.Lines.Add(' - 成功建立服務 :
"'+ServiceName+'"');
008       mmMessages.GoToTextEnd;
009     except
010       on E : Exception do
011       begin
012         mmMessages.Lines.Add(E.Message);
013         mmMessages.GoToTextEnd;
014       end;
015     end;
```

```
016   end;
```

TServerConnectionThread 是從 TThread 繼承下來的執行緒類別，它的功能是在一個獨立的執行緒中建立 TBluetoothSocket 物件準備接受用戶端藍牙的連結並接收資料。

因此下面即是 TServerConnectionThread 類別的宣告，它的 FSocket 即是使用來接受用戶端藍牙連結並接收資料的 TBluetoothSocket 物件變數，而 FData 則是 FSocket 從用戶端接收的資料：

```
  TServerConnectionThread = class(TThread)
  private
    { Private declarations }
    FServerSocket: TBluetoothServerSocket;
    FSocket: TBluetoothSocket;
    FData: TBytes;
  protected
    procedure Execute; override;
  public
    { Public declarations }
    constructor Create(ACreateSuspended: Boolean);
    destructor Destroy; override;

    property ServerSocket : TBluetoothServerSocket Read
FServerSocket Write FServerSocket;
  end;
```

當 TServerConnectionThread 物件在前面 CreateBTTextService 方法的第 6 行啟動之後就會執行它的 Execute 方法，在 010 行先接受用戶端藍牙連結以取得 TBluetoothSocket 物件，如果連結成功就在 016 行呼叫 ReveiveData 方法接受用戶端藍牙傳遞來的資料，接著在 018 行藉由 Synchronize 方法把接受來的資料顯示在主表單的 UI 中，由於傳遞來的資料型態是 TBytes，因此我們需要藉由 TEncoding.UTF8.GetString 方法把 TBytes 轉成字串型態：

```
001   procedure TServerConnectionThread.Execute;
002   var
003     ASocket: TBluetoothSocket;
004     Msg: string;
005   begin
```

版權所有 請勿翻印

```
006     while not Terminated do
007       try
008         ASocket := nil;
009         while not Terminated and (ASocket = nil) do
010           ASocket := FServerSocket.Accept(100);
011         if(ASocket <> nil) then
012         begin
013           FSocket := ASocket;
014           while not Terminated do
015           begin
016             FData := ASocket.ReveiveData;
017             if length(FData) > 0 then
018               Synchronize(procedure
019                 begin
020
fmMainForm.PostBTMessage(TEncoding.UTF8.GetString(FData));
021                 end);
022             sleep(100);
023           end;
024         end;
025       except
026         on E : Exception do
027         begin
028           Msg := E.Message;
029           Synchronize(procedure
030             begin
031               fmMainForm.PostBTMessage('伺服端連結已關閉 : ' +
Msg);
032             end);
033         end;
034       end;
035   end;
```

版權所有 請勿翻印

由於 **TServerConnectionThread** 類別中擁有 2 個建立的物件 **FSocket** 和 **FServerSocket**，因此在它的解構元中必備釋放這 2 個物件：

```
destructor TServerConnectionThread.Destroy;
begin
```

```
  FSocket.Free;
  FServerSocket.Free;
  inherited;
end;
```

另外在主表單的"移除通訊管道" 按鈕中也需要釋放我們建立的物件和資源，因此它的 OnClick 事件呼叫 FreeBTTextService 方法：

```
procedure TfmMainForm.btnRemoveTextServiceClick(Sender: TObject);
begin
  FreeBTTextService;
end;
```

FreeBTTextService 方法先判斷是否建立過 TServerConnectionThread 物件接受資料，如果有的話就先停止它的執行再釋放它：

```
procedure TfmMainForm.FreeBTTextService;
begin
  if (scThread <> nil) then
  begin
    scThread.Terminate;
    scThread.WaitFor;
    FreeAndNil(scThread);
    mmMessages.Lines.Add(' - 連結服務已移除 -');
    mmMessages.GoToTextEnd;
  end;
end;
```

版權所有 請勿翻印

現在我們已完成藍牙伺服端的開發工作，再來需要完成用戶端的開發工作，用戶端需要建立一個 TBluetoothSocket 物件要求連結，在伺服端接受之後就可以使用這個 TBluetoothSocket 物件傳送資料給伺服端。

因此在主表單的"傳送文字"按鈕中我們需要先找到配對需要傳遞資料的伺服端藍牙，再建立用戶端的 TBluetoothSocket 物件以傳遞資料給伺服端藍牙。因此在"傳送文字"按鈕的 OnClick 事件中先呼叫 FillPairDeviceInfo 方法找到配對藍牙設備，再呼叫 SendDataToServer 方法開始傳遞資料：

```
procedure TfmMainForm.btnSendTextClick(Sender: TObject);
begin
  FillPairDeviceInfo;
```

```
  SendDataToServer;
end;
```

下面是 SendDataToServer 方法的主程序代碼部份，SendDataToServer 方法先呼叫 CreateClientSocket 方法建立用戶端 TBluetoothSocket 物件，再呼叫 SendData 方法傳遞資料：

```
  try
    CreateClientSocket;
    SendData;
  except
    on E : Exception do
    begin
      PostBTMessage(E.Message);
      FreeAndNil(FClientSocket);
    end;
```

CreateClientSocket 方法先於 007 行找到使用者選擇要傳遞資料的配對藍牙設備，再呼叫代表配對藍牙設備的 pairedDevice 物件的 CreateClientSocket 方法建立用戶端的 TBluetoothSocket 物件，請注意 CreateClientSocket 方法的第 1 個參數是伺服端和用戶端使用來建立連結的相同 GUID 值。

版權所有 請勿翻印

在建立了用戶端 TBluetoothSocket 物件之後就可以於 012 行呼叫它的 Connect 方法連結伺服端藍牙設備：

```
001     procedure CreateClientSocket;
002     var
003       pairedDevice: TBluetoothDevice;
004     begin
005       if (FClientSocket = Nil) then
006       begin
007         pairedDevice :=
GetThePairedDevice(cbPairDevices.Items[cbPairDevices.ItemIndex])
;
008         PostBTMessage(GetServiceName(pairedDevice,
ServiceGUI));
009         FClientSocket :=
pairedDevice.CreateClientSocket(StringToGUID(ServiceGUI), False);
010         if (FClientSocket <> Nil) then
```

```
011         begin
012           FClientSocket.Connect;
013           PostBTMessage('藍牙已連結');
014         end
015         else
016          PostBTMessage('發生錯誤, 藍牙連結超時!');
017       end;
018     end;
```

上面的 **FClientSocket** 變數當然是宣告為 **TBluetoothSocket** 型態的物件變數：

```
    FClientSocket: TBluetoothSocket;
```

最後的 **SendData** 方法則非常的簡單，它使用剛才建立的 **FClientSocket** 物件呼叫它的 **SendData** 方法傳遞資料，由於 **SendData** 方法需要傳遞 **TBytes** 型態的資料，因此在 **005** 行需要藉由 **TEncoding.UTF8.GetBytes** 方法把主表單中 **TEdit** 元件中的文字字串型態資料先轉換成 **TBytes** 型態的資料：

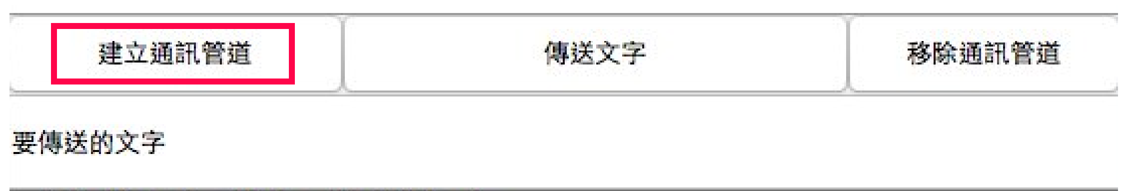
版權所有 請勿翻印

```
001     procedure SendData;
002     var
003       ToSend: TBytes;
004     begin
005       ToSend := TEncoding.UTF8.GetBytes(edtText.Text);
006       FClientSocket.SendData(ToSend);
007       PostBTMessage('訊息已傳送');
008     end;
```

到這裡我們也完成了用戶端的開發工作，接下來我們就可以試著執行範例程式來看看它是否能成功的工作，在下面我們就使用 **Mac** 做為藍牙伺服端，它執行 **OSX Yosemite 10.10.2** 版，另外再使用 **HTC M8** 做為藍牙用戶端，**M8** 執行了 **Android 4.4** 以上的版本。

首先部署此範例程式到 **OSX** 中執行，先如下圖點選”查詢藍牙設備”按鈕找到 **M8** 手機，再點選 **M8** 然後再點選”配對”按鈕以配對 **M8**：

最後再點選”建立通訊管道”按鈕建立伺服端 TBluetoothSocketh 物件等待稍後用戶端連結：



接著在用戶端 M8 手機中執行範例 App，找到配對的 Mac 機器，再點選”傳送文字”按鈕建立用戶端 TBluetoothSocketh 物件，連結伺服端並傳遞資料給伺服端：

版權所有 請勿翻印

下面的 2 個畫面就是伺服端和用戶端的執行結果，我們可以看到 M8 手機中的資料果然藉由藍牙傳遞到 Mac 機器中了。

版權所有 請勿翻印

14-6 10.3 版 Delphi 程式語言新功能

Delphi 10.3 版的編譯器開始進入一個新的階段，除了是因為低層使用的 Clang 已經支持 Clang 3.3 並且準備在將近的未來支持 Clang 3.9 外，10.3 版也開始為久未更動的 Delphi 程式語言加入新的功能。

10.3 版新語言功能主要是加入了 inline 變數宣告和型態推論，在下面的小節將分別說明如何使用這些新的語言功能。

14-6-1 inline 變數宣告

Delphi 的根基是 Pascal 程式語言，而 Pascal 在本質上是一強型(Strong Type) 程式語言，因此在 Pascal 中所有變數在使用之前都必須先行宣告其資料型態。但由於現行的程式語言是趨向自由型態，因此 Delphi 10.3 版的編譯器放寬了強型要求，允許程式師選擇使用自由型態和自由風格。

因此 Delphi 10.3 版中的 inline 變數宣告的意思是指程式師不再需要在一個程式或是函式的開頭 var 部份就先宣告所有要使用的區域變數，而是在程式碼中需要使用區域變數時再就地宣告就可以使用，而不必回到程式或是函式的開頭 var 部份補行宣告。這樣不但比較方便，而且比較直覺，因為一般來說程式師很難在撰寫程式或是函式時就能先在開頭 var 部份想到所有要使用的變數。

下面讓我們使用一些範例來說明什麼是 **inline** 變數宣告以及它的用法。

例如如果程式師撰寫了下面的程式碼在 **for...loop** 中需要一個 **iCount** 的整數迴圈變數，在以前的版本中需要回到 **var** 部分再加入 **iCount** 的宣告:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  wYear, wMonth, wDate, wHour, wMinute, wSecond, wMS : Word;
begin
  DecodeDate(Now, wYear, wMonth, wDate);
  DecodeTime(Now, wHour, wMinute, wSecond, wMS);

  for iCount := 1 to 100 do
  begin

  end;
end;
```

但在 10.3 版後就可以"就地"宣告使用不用回到 **var** 部分，因此 10.3 版後就可以使用下面的語法:

版權所有 請勿翻印

```
  DecodeDate(Now, wYear, wMonth, wDate);
  DecodeTime(Now, wHour, wMinute, wSecond, wMS);

  for var iCount : Integer := 1 to 100 do
  begin

  end;
```

再假設現在在 **for...loop** 中又需要另一個變數把 **wYear, wMonth, wDate, wHour, wMinute, wSecond, wMS** 等變數值加總在 **for...loop** 迴圈中使用，那麼就可以使用如下的語法:

```
  for var iCount : Integer := 1 to 100 do
  begin
    var iSum : Integer := wYear + wMonth + wDate + wHour + wMinute
+ wSecond + wMS;
    var iMod : Integer := iSum mod 100;
    …;
```

```
end;
```

如此就地宣告的 iCount，iSum 和 iMod 等變數就稱為 inline 變數宣告。

使用 inline 變數宣告時程式師必須注意它的有效範圍，一個使用 inline 宣告的變數其有效範圍是包含它的程式區塊中，一旦執行權離開了包含它的程式區塊，那麼使用 inline 宣告的變數便不可中被存取。

iCount, iSum 和 iMod 的有效範圍是 for...loop 迴圈的程式區塊

例如

```
procedure TForm1.Button1Click(Sender: TObject);
var
  wYear, wMonth, wDate, wHour, wMinute, wSecond, wMS : Word;
begin
  DecodeDate(Now, wYear, wMonth, wDate);
  DecodeTime(Now, wHour, wMinute, wSecond, wMS);

  for var iCount : Integer := 1 to 100 do
  begin
    var iSum : Integer := wYear + wMonth + wDate + wHour + wMinute + wSecond + wMS;
    var iMod : Integer := iSum mod 100;
    mmDemo.Lines.Add(iMod.ToString);
  end;
end;
```

版權所有 請勿翻印

因此如果我們在 **for...loop 迴圈**之外再存取 iSum 這個 inline 宣告變數的話就會得到如下的編譯錯誤:

```
Messages
[dcc32 Error] uMainForm.pas(41): E2003 Undeclared identifier: 'iSum'
[dcc32 Error] uMainForm.pas(42): E2029 ')' expected but identifier 'ToString' found
[dcc32 Fatal Error] pDelphiLanguageDemo.dpr(5): F2063 Could not compile used unit 'uMainForm.pas'
```

編譯器編譯出錯說找不到 iSum 的宣告，這就是因為出了 **for...loop 迴圈**之外 iSum 的宣告定義就消失了。這個 inline 變數宣告的特性是和把 iSum 宣告在程式開頭的 var 部分有效範圍是不同的。

14-6-2 型態推論

所謂型態推論功能是指 10.3 版之後的編譯器能夠從程式碼的有效範圍中主動得知 inline 宣告的變數的資料型態，如此一來程式師就不用自己撰寫 inline 宣告的變數的資料型態，而節省了撰寫的時間。

根據此定義，那麼我們可以進一步簡化上面的程式碼如下:

```
for var iCount := 1 to 100 do
```

```
  begin
    var iSum := wYear + wMonth + wDate + wHour + wMinute + wSecond
+ wMS;
    var iMod := iSum mod 100;
    …;
  end;
```

由於 for...loop 迴圈的迴圈值一定是整數，因此編譯器可得知 iCount 的資料型態一定整數型態，因此程式師不用再撰寫宣告 iCount 的資料型態。

同樣的 iSum 是把 Word 型態的數值相加，因此編譯器自動以 Word 型態處理 iSum，而 mod 運算元是整數值因此編譯器可得知 iMod 的資料型態一定整數型態。

16-4-3 其他語言新功能

10.3 版之後的編譯器除了可以使 inline 宣告變數，也可以直接在 inline 宣告變數之後立即指定數值給它，例如：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  var iValue := 123456;
  var iArray := [1, 2, 3, 4, 5, 6];

  mmDemo.Lines.Add(iValue.ToString);
  mmDemo.Lines.Add(iArray[1].ToString);
```

版權所有 請勿翻印

當然，除了變數之外，也可以對常數進行 inline 宣告變數和簡單的運算，例如下面顯示的程式碼:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  var iValue := 123456;
  var iArray := [1, 2, 3, 4, 5, 6];

  mmDemo.Lines.Add(iValue.ToString);
  mmDemo.Lines.Add(iArray[1].ToString);

  const IDAYSECONDS = 365 * 24 * 60 * 60;
  var iSeconds := IDAYSECONDS mod 1000;
```

```
  mmDemo.Lines.Add(iSeconds.ToString);
end;
```

加入這些新的語言新功能主要是方便程式師更快的撰寫程式碼。

15 呼叫 Android 系統功能

在 Android 平臺 Delphi 產生的 App 是原生機械碼而不是 Java 的虛擬程式碼，不過 Android 的功能大多是用 Java 撰寫的因此在一些應用場合我們仍然需要在 Delphi 中呼叫 Java 的程式碼或是 API。在前面的，章節中已經有 2 個範例說明如何在 Delphi 中呼叫 Java 的功能，在本小節中將做比較完整的說明並使用數個範例來展示。

開發人員在使用 Delphi 呼叫 Java 的 API 或是程式碼時需要瞭解一些基本的知識才可以順利的完成呼叫工作。這些知識有些是使用在呼叫 Java 的 API 需要的，有的則是在呼叫 Delphi 宣未封裝的 API 時需要的，下面的表格說明了這些最重要的知識：

需要瞭解的知識	說明
JObjectClass 介面	Delphi 使用來封裝 Java API 類別的內容，例如類別常數(class constant)，類別方法等
JObject 介面	Delphi 使用來封裝 Java API 物件的內容，例如物件方法等
TJavaGenericImport 類別	Delphi 使用來合成 JObjectClass 和 JObject 介面成為 Delphi 類別，如此一來開發人員可建立此類別物來呼叫 Java API。此類別是泛型類似因此可使用封裝任何的 JObjectClass 和 JObject 介面
JavaSignature 屬性	使用來指定 JObject 介面封裝的 Java 類別
init 方法	TJavaGenericImport 類別的建構元(constructor)，由於 TJavaGenericImport 類別是封裝 Java 類別，因此要真正建立 JVM 中的 Java 物件，Delphi 程式碼要呼叫此 init 方法
cdecl 呼叫方式宣告	在使用 JObject 介面封裝的 Java API 時，一定要使用 cdecl 的呼叫方式

版權所有 請勿翻印

15-1 呼叫 Java 類別程式碼

首先讓我們使用一個簡單的 Java 類別來展示如何讓 Delphi 可以呼叫 Java 程式碼，在這個討論的過程中您將學習到數個重要的技術來幫助您瞭解如何能夠不直接使用 JNI 方式而讓 C/C++可以呼叫 Java。

下面是一個非常簡單的 Java 類別，讓我們用 Delphi 寫一個 Android App 來呼叫其中的 setIntValue() 方法設定 RTWrapClassTest1 類別物件的 storedintvalue 數值，再呼叫其中的 getIntValue ()方法取出這個數值看看從 Delphi 設定 Java 數值是否能成功：

```
package com.xe5test.myjavaclasastest;

import android.util.Log;

public class RTWrapClassTest1 {
    int storedintvalue;

    public int getIntValue {
        Log.d("classtesting","getIntValue called" );
        return storedintvalue + 3;
    }

    public void setIntValue(int newvalue) {
        Log.d("classtesting", "setIntValue called");
        storedintvalue = newvalue;
    }

    public void stunt(String s, int n) {
        Log.d("classtesting", "stunt called");

        for (int x = 10; x < 12; x = x + 1)
            Log.d("classtesting", s + ' ' + n);
    }
}
```

版權所有 請勿翻印

這個 Java 類別被執編譯並封裝在 XE5ClassTesting1.jar 中，現在我們就可以準備來呼叫了。

在 RAD Studio 中我們可以使用如下的步驟來讓 Delphi 直接呼叫 Java 程式碼：

1. 使用 Java2OP 工具把 Java 類別宣告轉成 Delphi 類別宣告

2. 在 Delphi 專案中 use 上面步驟產生的 Delphi 類別宣告，編譯並直接連結到最後的 App 中

3. 使用 Delphi 專案管理員在 App 中加入 XE5ClassTesting1.jar 檔並部署

在下面的內容中我們將一一的說明如何完成每一個步驟。

步驟 1 把 Java 類別宣告轉成 Delphi 類別宣告

首先您可以使用 RAD Studio 內附的 Java2OP 工具把 Java Jar 檔中的 Java 類別轉成 Delphi 類別宣告，例如您可以使用如下的命令把 mylib.jar 中的 Java 類別轉成 Delphi 類別宣告：

```
Java2OP.exe -jar mylib.jar
```

版權所有 請勿翻印

因此在此步驟中您只需要使用下面的指令：

```
Java2OP.exe -jar XE5ClassTesting1.jar
```

就可以產生類似如下的 Delphi 類別宣告：

```
unit com.xe5.ClassTesting1.RTClassTesting1;

interface

uses
  AndroidAPI.JNIBridge,
  Androidapi.JNI.JavaTypes;

type
  JRTClassTesting1 = interface;

  JRTClassTesting1Class = interface(JObjectClass)
    ['{8EF97555-32BC-4262-B17E-7A5157944E97}']
    function getIntValue : Integer; cdecl;
```

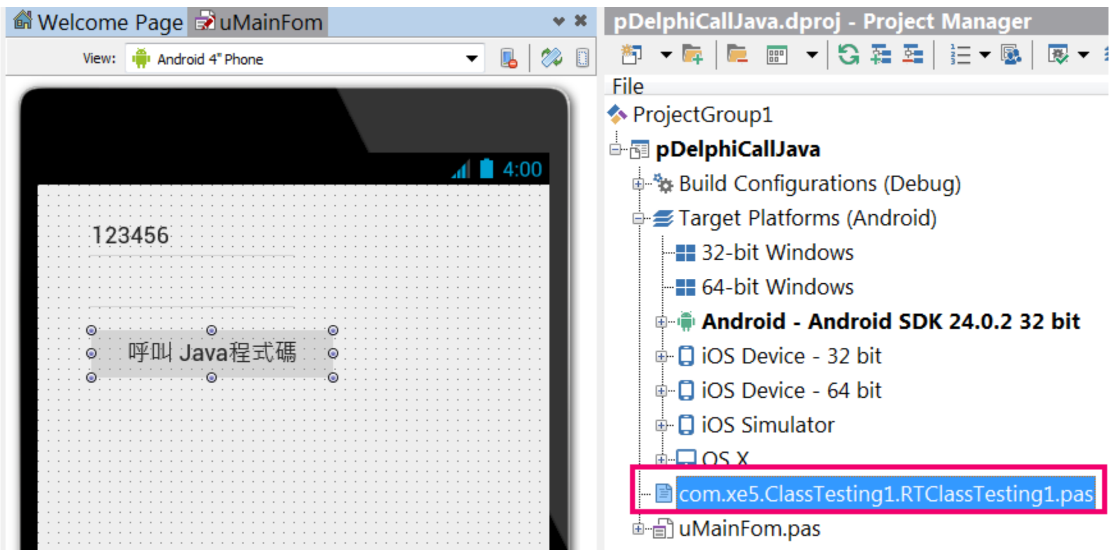
```
// ()I A: $1
    function init : JRTClassTesting1; cdecl;
// ()V A: $1
    procedure setIntValue(newvalue : Integer) ; cdecl;
// (I)V A: $1
    procedure stunt(s : JString; n : Integer) ; cdecl;
// (Ljava/lang/String;I)V A: $1
  end;

  [JavaSignature('com/xe5/ClassTesting1/RTClassTesting1')]
  JRTClassTesting1 = interface(JObject)
    ['{983668BC-BD74-4142-8A1F-3DDA43F3E29F}']
    function getIntValue : Integer; cdecl;
// ()I A: $1
    procedure setIntValue(newvalue : Integer) ; cdecl;
// (I)V A: $1
    procedure stunt(s : JString; n : Integer) ; cdecl;
// (Ljava/lang/String;I)V A: $1
  end;

  TJRTClassTesting1 =
class(TJavaGenericImport<JRTClassTesting1Class,
JRTClassTesting1>)
  end;

implementation

end.
```

版權所有 請勿翻印

讓我們把這個 Delphi 類別宣告儲存為 com.xe5.ClassTesting1.RTClassTesting1.pas。

步驟 2 編譯並直接連結到最後的 App 中

現在在 Delphi IDE 中建立一個 Multi-Device Application 項目如下，並在專案中加入上一步驟產生的 com.xe5.ClassTesting1.RTClassTesting1.pas 檔案：



接著在主表單程式碼中只需要 uses 在步驟 1 com.xe5.ClassTesting1.RTClassTesting1.pas：

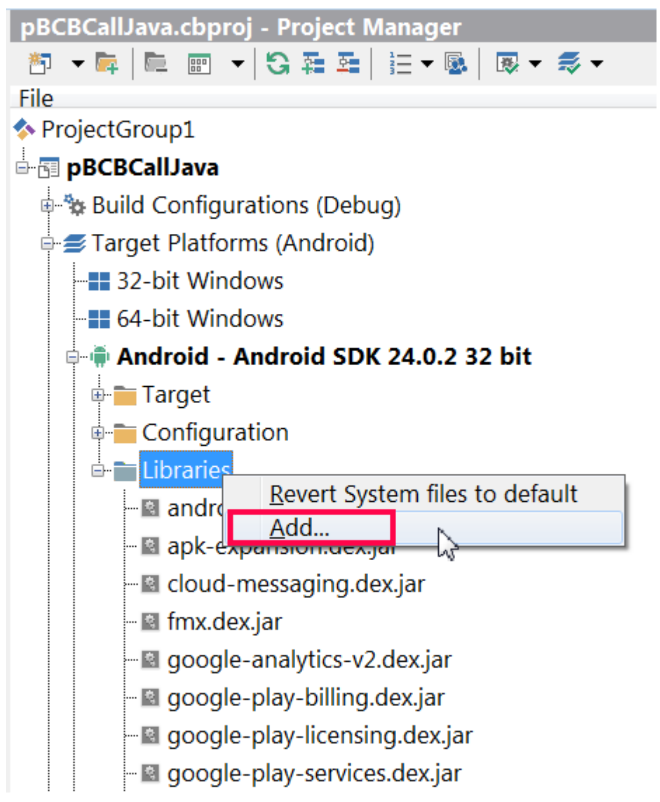
```
implementation

{$R *.fmx}
{$R *.NmXhdpiPh.fmx ANDROID}

uses com.xe5.ClassTesting1.RTClassTesting1;
…
```

版權所有 請勿翻印

就可以了。

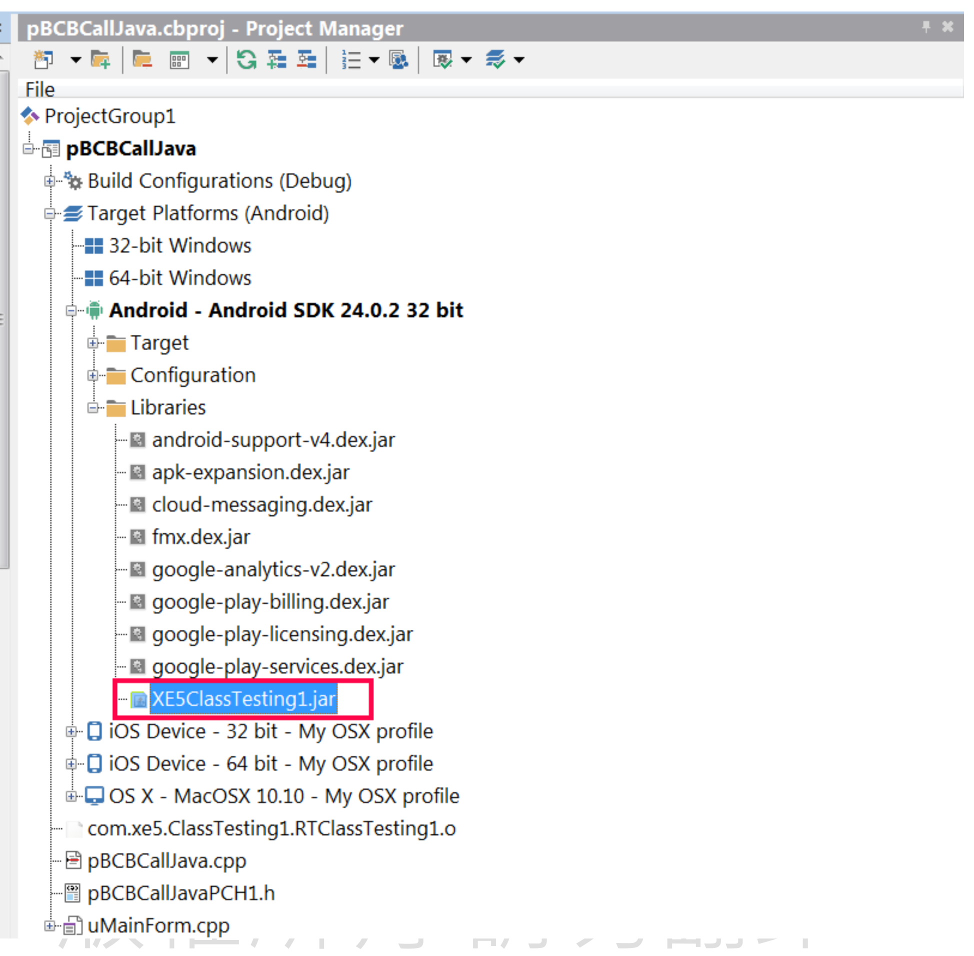
步驟 3 加入 Java 的.jar 檔並部署

RIO 提供了可讓程式師在 Delphi 使用的在 IDE 中 Java 及函式館中加入客制化的 jar 檔以便讓 C/C++程式碼可呼叫其中的 Java 程式碼。由於現在我們要呼叫 XE5ClassTesting1.jar 中的 Java 程式碼，因此請到 IDE 的專案管理員中展開 Android 平臺，再展開 Libraries 節點，右擊滑鼠再從突顯選單中點選 Add...選項並選擇加入 XE5ClassTesting1.jar：



下圖就是加入 XE5ClassTesting1.jar 的結果：

版權所有 請勿翻印



在加入了 XE5ClassTesting1.jar 之後 IDE 便會把它封裝到 classes.dex 並準備部署到 Android 平臺中。

完成了上面的步驟後要讓 Delphi 呼叫 Java 就簡單了，對於 Delphi 來說就像是呼叫一般的 Delphi 程式碼一樣。我們只需要使用如下的程式碼即可：

```
001   procedure TfmMainForm.Button1Click(Sender: TObject);
002   var
003     aj : JRTClassTesting1;
004   begin
005     aj := TJRTClassTesting1.Create;
006     try
007       aj.setIntValue(Edit1.Text.ToInteger());
008       Edit2.Text := aj.getIntValue.ToString();
009     finally
010       aj := Nil;
011     end;
```

```
012    end;
```

在 005 行我們藉由呼叫 TJRTClassTesting1.Create() 建立 JRTClassTesting11 型態的物件"aj"，而這也等於建立了 Java 的 RTWrapClassTest1 類別物件。接著在 007 和 008 行就可以藉由 aj 直接呼叫 RTWrapClassTest1 類別物件的 setIntValue()和 getIntValue()方法了，就像直接呼叫由 Delphi 語言寫的程式碼一樣。

最後編譯並部署到 Android 手機中執行這個範例 App，從下面的執行畫面可以看到我們成功的使用 Delphi 呼叫了由 Java 撰寫的程式碼。

版權 印

瞭解了如何使用 Delphi 程式語言呼叫 Java 程式碼後現在就讓我們使用一個範例來說明如何使用上面的知識來封裝一個 Java 類別 Toast，在完成封裝 Toast 之後讀者就可以完全瞭解上表的意義了。

15-2 呼叫 Delphi 尚未封裝的 Java 類別

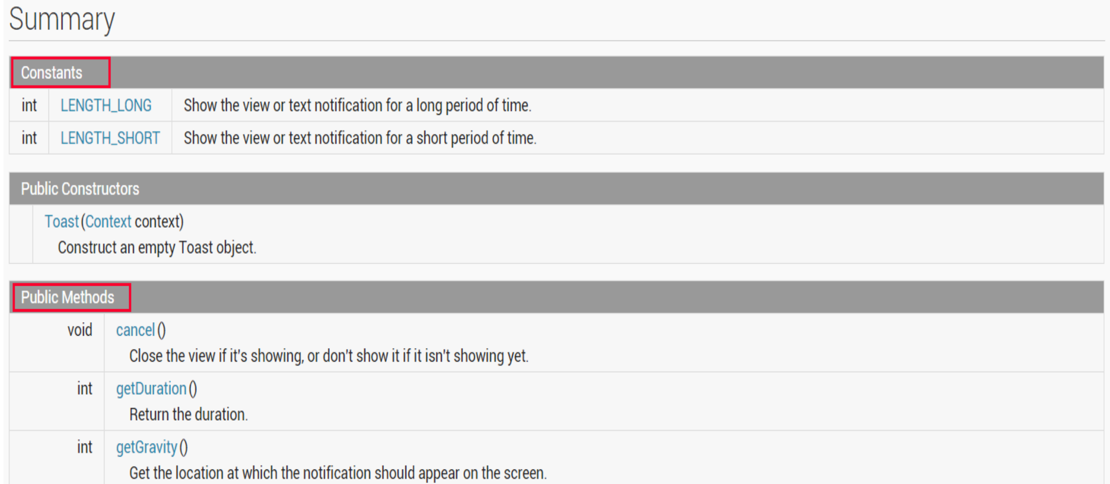
Toast 是一個非常簡單的 Java 類別，它可以在 UI 中顯示一個訊息，類似 Delphi 的 ShowMessage 對話盒。Toast 的資訊可以在下面的 URL 中查到：

```
http://developer.android.com/reference/android/widget/Toast.html
```

它是宣告在 android.widget 中，它完整的類別宣告是：

```
android.widget.Toast
```

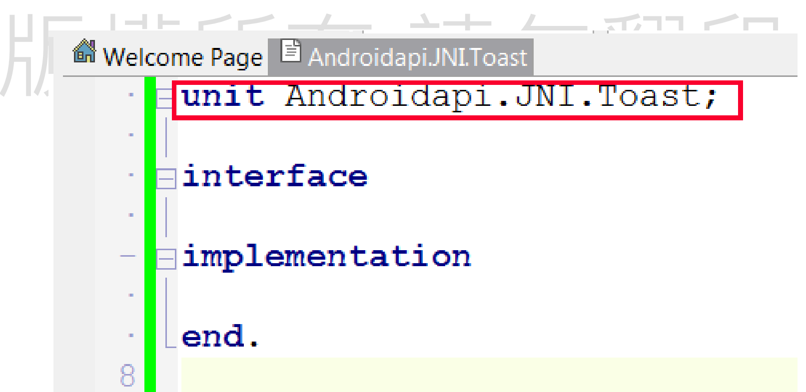
在 Toast.html 中我們可以看到 Toast 如下的宣告，請注意在其中它定義了 2 個類別常數 LENGTH_LONG 和 LENGTH_SHORT，根據上表的說明它們應該宣告在 JObjectClass 介面，而下圖中的 Java 建構元必須用上表的 init 方法宣告，至於它的公開方法 cancel()，getDuration()等根據上表的說明它們應該宣告在 JObject 介面中：



但是 Toast 在公開方法中有 2 個覆載類別方法 makeText，根據上表的說明它們也應該宣告在 JObjectClass 介面中：

Public Methods	
void	cancel() Close the view if it's showing, or don't show it if it isn't showing yet.
int	getDuration() Return the duration.
int	getGravity() Get the location at which the notification should appear on the screen.
float	getHorizontalMargin() Return the horizontal margin.
float	getVerticalMargin() Return the vertical margin.
View	getView() Return the view.
int	getXOffset() Return the X offset in pixels to apply to the gravity's location.
int	getYOffset() Return the Y offset in pixels to apply to the gravity's location.
static Toast	makeText(Context context, int resId, int duration) Make a standard toast that just contains a text view with the text from a resource.
static Toast	makeText(Context context, CharSequence text, int duration) Make a standard toast that just contains a text view.

瞭解上面的說明之後我們就可以到 Delphi IDE 中建立一個程式單元並把它儲存為 Androidapi.JNI.Toast.pas，它的程式單元名稱就會是 Androidapi.JNI.Toast：



接著在程式單元的 uses 子句中加入：

```
Androidapi.JNIBridge,
Androidapi.JNI.JavaTypes,
Androidapi.JNI.GraphicsContentViewText;
```

再宣告 JToast 介面從 JObject 繼承下來，JToastClass 從 JObjectClass 繼承下來，並於介面宣告中按下 Ctrl-Shift-G 產生介面 GUID 值，如下所示：

```
unit Androidapi.JNI.Toast;

interface

uses
  Androidapi.JNIBridge,
  Androidapi.JNI.JavaTypes,
  Androidapi.JNI.GraphicsContentViewText;

type

  JToast = interface;

  JToastClass = interface(JObjectClass)
  ['{F6BBFD92-FA74-4DCD-9AFD-9209656A7034}']


  end;

  [JavaSignature('android/widget/Toast')]
  JToast = interface(JObject)
  ['{CDF541F2-DBB5-4BF9-AC2E-011DB478DCEB}']

  end;

implementation

end.
```

使用 Ctrl-Shift-G 產生 GUID

版權所有 請勿翻印

使用 JObjectClass 介面封裝 Toast 類別內容

接著就可以使用 JObjectClass 介面來封裝 Toast 類別範圍的內容如下：

```
JToastClass = interface(JObjectClass)
['{F6BBFD92-FA74-4DCD-9AFD-9209656A7034}']
  {Property methods}
  function _GetLENGTH_LONG: Integer; cdecl;
  function _GetLENGTH_SHORT: Integer; cdecl;
  {Methods}
  function init(context: JContext): JToast; cdecl; overload;
  function makeText(context: JContext; text: JCharSequence;
duration: Integer): JToast; cdecl; overload;
  function makeText(context: JContext; resId: Integer; duration:
Integer): JToast; cdecl; overload;
  {Properties}
  property LENGTH_LONG: Integer read _GetLENGTH_LONG;
  property LENGTH_SHORT: Integer read _GetLENGTH_SHORT;
```

```
end;
```

請注意上面的

1. 每一個方法都使用 cdecl 呼叫方式宣告
2. makeText 是宣告為覆載類別方法
3. Java Toast 類別的類別常數是宣告在 JToastClass 中

上面的每一點都是遵照前表的說明。

使用 JObject 介面封裝 Toast 物件內容

接下來再使用 JObject 介面封裝 Toast 物件範圍內容：

```
[JavaSignature('android/widget/Toast')]
JToast = interface(JObject)
['{CDF541F2-DBB5-4BF9-AC2E-011DB478DCEB}']
  {Methods}
  procedure cancel; cdecl;
  function getDuration: Integer; cdecl;
  function getGravity: Integer; cdecl;
  function getHorizontalMargin: Single; cdecl;
  function getVerticalMargin: Single; cdecl;
  function getView: JView; cdecl;
  function getXOffset: Integer; cdecl;
  function getYOffset: Integer; cdecl;
  procedure setDuration(value: Integer); cdecl;
  procedure setGravity(gravity, xOffset, yOffset: Integer); cdecl;
  procedure setMargin(horizontalMargin, verticalMargin: Single);
cdecl;
  procedure setText(s: JCharSequence); cdecl;
  procedure setView(view: JView); cdecl;
  procedure show; cdecl;
end;
```

版權所有 請勿翻印

請注意上面的

1. 使用 JavaSignature 屬性宣告此介面是封裝 android.widget.Toast 類別，請注意 JavaSignature 屬性需要使用”/”代替”.”，所以在 JavaSignature 屬性中是用 android/widget/Toast

2. 每一個方法都使用 cdecl 呼叫方式宣告

上面的每一點也都是遵照前表的說明。

使用 TJavaGenericImport 類別封裝完整 Java 類別

最後我們使用 TJavaGenericImport 封裝 JToastClass 和 JToast 介面成為一個 Delphi 類別：

```
TJToast = class(TJavaGenericImport<JToastClass, JToast>) end;
```

使用 TJToast 類別

現在 TJToast 類別已經封裝完成，我們最後再於 Androidapi.JNI.Toast 程式單元中宣告一個 TToastTime 來代表顯示訊息的時間長短：

版權所有 請勿翻印

```
type
  TToastTime = (LongToast, ShortToast);
```

再撰寫一個公共方法 ShowToast：

```
001   procedure ShowToast(const Msg: string; Duration: TToastTime);
002   var
003     ToastLength: Integer;
004   begin
005     if Duration = ShortToast then
006       ToastLength := TJToast.JavaClass.LENGTH_SHORT
007     else
008       ToastLength := TJToast.JavaClass.LENGTH_LONG;
009     CallInUiThread(procedure
010     begin
011       TJToast.JavaClass.makeText(SharedActivityContext,
StrToJCharSequence(msg),
012         ToastLength).show
013     end);
014   end;
```

ShowToast 方法使用了數個重要的技巧，首先由於顯示訊息是要在 FireMonkey 的 UI 執行緒中執行，因此上面的 011 行呼叫了 CallInUiThread 要求 Java 的 Toast 類別物件要在 FireMonkey 的 UI 執行緒中執行。

CallInUiThread 提供了數個覆載方法可讓開發人員呼叫：

```
procedure CallInUIThread(AMethod: TMethodCallback); overload;
procedure CallInUIThread(AMethod: TCallBack); overload;
procedure CallInUIThreadAndWaitFinishing(AMethod:
TMethodCallback); overload;
procedure CallInUIThreadAndWaitFinishing(AMethod: TCallBack);
overload;
```

第 2 個技巧是 makeText 類別方法的第 1 個參數 JContext 可由 Delphi 在 Androidapi.Helpers 程式單元中的公共方法取得：

```
function SharedActivityContext: JContext;
begin
  Result := SharedActivity;
end;
```

版權所有 請勿翻印

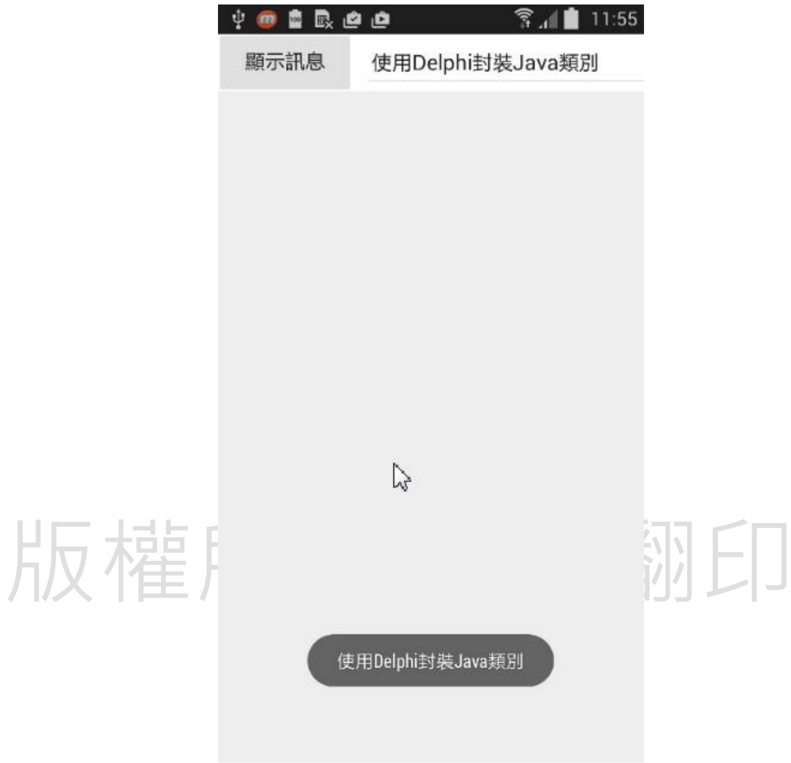
而是 makeText 類別方法的第 2 個參數的型態是 JCharSequence，因此我們可藉由呼叫 StrToJCharSequence 把 Delphi 的字串型態轉成 JCharSequence。同樣的在 Androidapi.Helpers 程式單元中 Delphi 提供了 Delphi 的字串型態和 JCharSequence 互轉的函式：

```
function StrToJCharSequence(const ASource: string): JCharSequence;
begin
  Result := TJCharSequence.Wrap((StringToJString(ASource) as
ILocalObject).GetObjectID);
end;

function JCharSequenceToStr(const ASource: JCharSequence): string;
begin
  Result := JStringToString(ASource.toString);
end;
```

完成了上面的程式碼後我們就可以使用下面簡單的程式碼來使用 Android 的 android.widget.Toast 物件來顯示訊息了：

```
procedure TForm3.Button1Click(Sender: TObject);
begin
  ShowToast(edtMessage.Text, TToastTime.LongToast);
end;
```

下面就是在 Delphi 原生 App 中成功呼叫 android.widget.Toast 物件顯示訊息的結果：



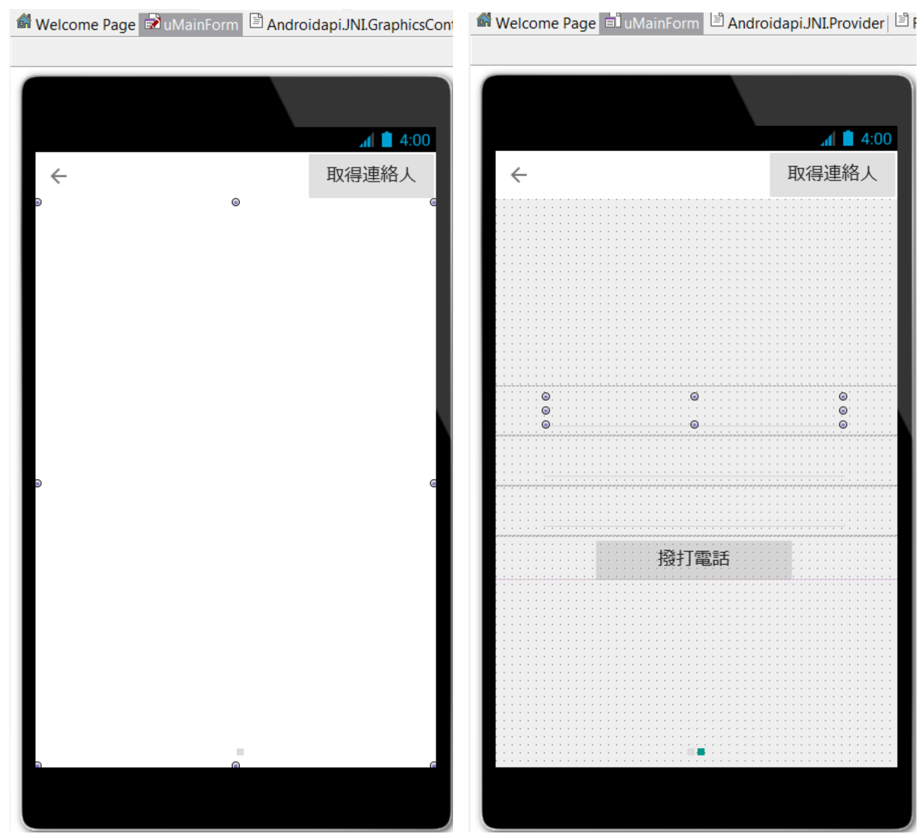
版權 翻印

15-3　呼叫 Java API 存取 Android 連絡人資訊

上面的範例是說明如何使用 Delphi 呼叫尚未封裝的 Android API 的方法，在本小節中再讓我們說明如何使用 Delphi 呼叫已封裝的 Android API 來存取 Android 上的資訊。

在許多的 App 應用中經常會需要存取手機中連絡人的資訊，但目前 Delphi 尚未封裝連絡人資訊為元件和類別讓開發人員使用，不過這也正好讓我們使用這個需求做為說明如何在 Delphi 中呼叫 Java API 的範例。

首先建立一個 Multi-Device 專案並設計如下的 UI，在 TPageControl 的第 1 個頁面將顯示連絡人姓名和電話，點選任何連絡人就會在第 2 個頁面顯示 Email 的資訊：



版權所有 請勿翻印

因此我們需要使用 Delphi 程式碼存取連絡人的姓名，電話和 Email，讓我們宣告 TContactPerson 類別以儲存每一個連絡人的資訊：

```
type
  TContactPerson = class
  private
    FID : String;
    FName : String;
    FPhone : String;
    FEMail : String;
  public
    property ID : String read FID write FID;
    property Name : String read FName write FName;
    property Phone : String read FPhone write FPhone;
    property EMail : String read FEMail write FEMail;
  end;
```

在 TContactPerson 類別中的 ID 欄位將儲存使用來查詢 Android 連絡人的查詢鍵值，使用這個查詢鍵值才能夠查詢到連絡人的延伸資訊，例如電話號碼和 Email 等，我們馬上就會說明如何能取得此查詢鍵值。

現在讓我們開發先說明如何取得連絡人的查詢鍵值和顯示名稱資訊，再根據查詢鍵值取得其他需要的資訊。在主表單的”取得連絡人”按鈕中呼叫 GetContactsInfo 方法：

```
procedure TfmMainForm.btnGetContactsClick(Sender: TObject);
begin
  GetContactsInfo;
end;
```

GetContactsInfo 方法會呼叫 GetContentResolver，QueryContactInfo 和 DisplayContactInfo3 個方法，它們分別取得查詢連絡人的 JCursor 介面，實際進行查詢資訊的工作以及顯示查詢的結果：

```
procedure TfmMainForm.GetContactsInfo;
begin
  GetContentResolver;
  QueryContactInfo;
  DisplayContactInfo;
end;
```

版權所有 請勿翻印

要查詢 Android 的連絡人資訊我們需要呼叫 Android API 的 android.content.ContentResolver 的 query 方法：

```
function query(uri: Jnet_Uri; projection:
TJavaObjectArray<JString>; selection: JString; selectionArgs:
TJavaObjectArray<JString>; sortOrder: JString): JCursor; cdecl;
overload;
```

第 1 個參數 uri 代表要查詢的內容，第 2 個參數 projection 類似要取得的資訊欄位，第 3 個參數 selection 類似 SQL 命令的 where 條件，第 4 個參數 selectionArgs 類似 SQL 命令的動態參數值，selectionArgs 的內容會動態帶入第 3 個參數 selection 中，最後一個參數 sortOrder 代表查詢出來的資料的排序方式。

但要怎麼取得 ContentResolver 以呼叫 query 方法呢？

GetContentResolver 方法藉由存取 Delphi 定義的 SharedActivityContext 公共方法取得 JCursor 介面再呼叫 JCursor 介面中

的 getContentResolver 方法即可取得代表 android.content.ContentResolver 的介面 JContentResolver。

```
procedure TfmMainForm.GetContentResolver;
begin
  jcr := SharedActivityContext.getContentResolver;
end;
```

上面的 jcr 是宣告在表單類別 private 部份的 JContentResolver 介面變數：

```
 private
   { Private declarations }
   jc : JCursor;
   cjc : JCursor;
   jcr : JContentResolver;
   contacts : TObjectList<TContactPerson>;
```

取得了 JContentResolver 介面之後就可以呼叫它的 Query 方法查詢連絡人資訊了，根據下面 URL 的 Android API 的說明

版權所有 請勿翻印

```
http://developer.android.com/reference/android/provider/Contacts
Contract.ContactsColumns.html
```

我們可以看到下面的資訊，其中的 DISPLAY_NAME 正是我們要查詢的連絡人名稱，而 LOOKUP_KEY 正是查詢鍵值：

Summary

Constants		
String	CONTACT_LAST_UPDATED_TIMESTAMP	Timestamp (milliseconds since epoch) of when this contact was last updated.
String	DISPLAY_NAME	The display name for the contact.
String	HAS_PHONE_NUMBER	An indicator of whether this contact has at least one phone number.
String	IN_DEFAULT_DIRECTORY	Flag that reflects whether the contact exists inside the default directory.
String	IN_VISIBLE_GROUP	Flag that reflects the GROUP_VISIBLE state of any ContactsContract.CommonDataKinds.GroupMembership for this contact.
String	IS_USER_PROFILE	Flag that reflects whether this contact represents the user's personal profile entry.
String	LOOKUP_KEY	An opaque value that contains hints on how to find the contact if its row id changed as a result of a sync or aggregation.
String	NAME_RAW_CONTACT_ID	Reference to the row in the RawContacts table holding the contact name.
String	PHOTO_FILE_ID	Photo file ID of the full-size photo.
String	PHOTO_ID	Reference to the row in the data table holding the photo.
String	PHOTO_THUMBNAIL_URI	A URI that can be used to retrieve a thumbnail of the contact's photo.
String	PHOTO_URI	A URI that can be used to retrieve the contact's full-size photo.

點選上面 2 個欄位可以看到代表這 2 個欄位的常數值是”display_name”和”lookup”：

public static final String DISPLAY_NAME

The display name for the contact.

Type: TEXT

Constant Value "display_name"

public static final String LOOKUP_KEY

An opaque value that contains hints on how to find the contact if its row id changed as a result of a sync or aggregation.

Constant Value "lookup"

查詢 Android 連絡人的資訊有一個規則，那就是先要查詢到您要查詢資料的欄位索引值，再根據此欄位索引值來查詢真正的欄位內容資訊。例如如果我們要查詢連絡人名稱，那要先根據上面的”display_name”常數查詢它的欄位索引值，再根據此欄位索引值查詢到真正的連絡人名稱。

版權所有 請勿翻印

瞭解了這些基本的觀念後就可以準備呼叫 JContentResolver 介面的 Query 方法查詢連絡人資訊了，但我們要傳入正確的參數給 Query 方法。

Query 方法第 1 個參數是表示要查詢什麼內容，根據 Android API 檔連絡人是定義在 ContactsContract.Contacts 中，而 Delphi 使用 TJContactsContract_Contacts 定義了這個類別：

```
  [JavaSignature('android/provider/ContactsContract$Contacts')]
  JContactsContract_Contacts = interface(JObject)
    ['{F174D654-2BBC-4854-BB3A-23F403CE38D8}']
  end;
  TJContactsContract_Contacts =
class(TJavaGenericImport<JContactsContract_ContactsClass,
JContactsContract_Contacts>) end;
```

在 JContactsContract_ContactsClass 中的 CONTENT_URI 特性正是 Query 方法需要的第 1 個參數值，代表我們要查詢連絡人資訊：

```
  {class} property CONTENT_URI: Jnet_Uri read _GetCONTENT_URI;
```

所以在下面的 QueryContactInfo 方法中我們使用 jcr 呼叫它的 Query 方法，由於現在我們要取得所有連絡人資訊，因此 Query 方法的 2，3，4 參數都使用 Nil。Query 方法最後一個參數我們藉由呼叫 StringToJString 把 Delphi 的字串轉成 Java 字串要求所有連絡人資訊以名稱排序：

```
procedure TfmMainForm.QueryContactInfo;
var
  aContact : TContactPerson;
begin
  jc :=
jcr.query(TJContactsContract_Contacts.JavaClass.CONTENT_URI, Nil,
Nil, Nil, StringToJString('display_name ASC'));
  jc.moveToFirst();
  while (jc.moveToNext()) do
  begin
    aContact := TContactPerson.Create;
    aContact.ID := GetContactID;
    aContact.Name := GetContactName;
    aContact.Phone := GetContactPhone(aContact);
    aContact.EMail := GetContactEMail(aContact);
    contacts.Add(aContact);
  end;
end;
```

版權所有 請勿翻印

Query 方法成功執行之後會回傳 JCursor 介面：

```
  [JavaSignature('android/database/Cursor')]
  JCursor = interface(JCloseable)
```

我們就可以使用 JCursor 介面中的方法再存取我們需要的資訊值，由於 Query 方法回傳的類似一個資料表，因此我們先呼叫 JCursor 介面中的 **moveToFirst()**方法定位到第 1 筆資料，再進入迴圈藉由呼叫 **moveToNext()**方法依序的存取每一筆資料。

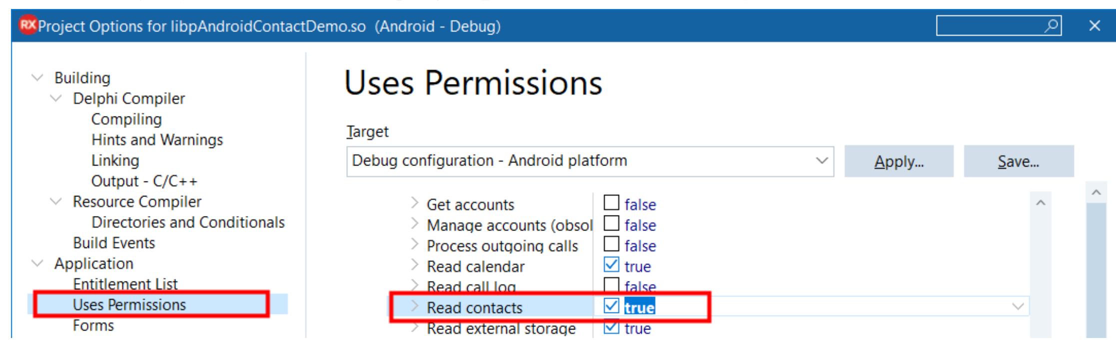
GetContactID，GetContactName 方法就是使用回傳的 JCursor 介面變數 jc 來實際查詢我們需要的查詢鍵值和連絡人名稱。從下面的程式碼我們可以很清楚的看到它使用的規則，先使用 jc 的 getColumnIndex 方法以欄位常數取得欄

位索引值，接著再次使用 jc 的 getString 方法藉由欄位索引值來取得欄位內容值，最後呼叫 JStringToString 把 Java 字串轉回 Delphi 字串：

```
function TfmMainForm.GetContactID: String;
begin
  Result :=
JStringToString( jc.getString(jc.getColumnIndex(StringToJString(
'lookup'))) );
end;

function TfmMainForm.GetContactName: String;
begin
  Result :=
JStringToString( jc.getString(jc.getColumnIndex(StringToJString(
'display_name'))) );
end;
```

在試著執行此範例 App 之前您必須開啟 App 讀取連絡資訊的許可權，請點選 Tools|Options...選單，在 User Permissions 項目中如下勾選設定 Read contacts 為 True：

版權所有 請勿翻印



完成上面的程式碼和許可權設定後執行此範例程式就可以看到類似如下的結果，果然可以存取到 Android 中連絡人的資訊了：

那麼要如何再查詢到電話和 Email 等其他的資訊呢？這就需要使用前面取得的查詢鍵值了，因為這些額外的資料 Android 是定義在 ContactsContract.CommonDataKinds.Email，ContactsContract.CommonDataKinds.Phone 和 ContactsContract.CommonDataKinds.Photo 等類別中。例如下圖是 ContactsContract.CommonDataKinds.Email 類別的定義，我們可以看到它的 ADDRESS 一欄正是我們需要的連絡人 EMAIL：

版權所有 請勿翻印

public static final class

Summ

ContactsContract.CommonDataKinds.Email

extends Object
implements ContactsContract.DataColumnsWithJoins ContactsContract.CommonDataKinds.CommonColumns

java.lang.Object
↳android.provider.ContactsContract.CommonDataKinds.Email

Class Overview

A data kind representing an email address.

You can use all columns defined for `ContactsContract.Data` as well as the following aliases.

Column aliases

Type	Alias	Data column	
String	ADDRESS	DATA1	Email address itself.

所以要查詢特定連絡人的 Email ,我們必須使用前面那個特定連絡人的查詢鍵值到 ContactsContract.CommonDataKinds.Email 類別中查詢，這類似資料庫的 Foreign Key 的概念。在 Delphi 中 ContactsContract.CommonDataKinds.Email 類別已經由 TJCommonDataKinds_Email 定義了：

版權所有 請勿翻印

```
[JavaSignature('android/provider/ContactsContract$CommonDataKinds$Email')]
  JCommonDataKinds_Email = interface(JObject)
    ['{E1AC9554-07BA-4399-8907-B92FA714B486}']
  end;
  TJCommonDataKinds_Email =
class(TJavaGenericImport<JCommonDataKinds_EmailClass,
JCommonDataKinds_Email>) end;
```

因此在下面的 GetContactEMail 方法中 009~010 我們使用 TJavaObjectArray<JString> 泛型類別建立要查詢的 Email 位址欄位，012~014 建立查詢準則，013 代表要查詢 Email 資訊內容，014 則帶入先前已取得的此連絡人的查詢鍵值。016 行同樣取得代表查詢結果的 JCursor 介面變數 emailCursor。018 行判斷如果查詢到此連絡人的 Email 資訊就在 021 行取得他的 Email 位址：

```
001    function TfmMainForm.GetContactEMail(aContact :
TContactPerson): String;
002    var
003      emailCursor : JCursor;
004      sProjection : TJavaObjectArray<JString>;
005      sWhere : TJavaObjectArray<JString>;
006    begin
007      Result := '';
008
009      sProjection := TJavaObjectArray<JString>.Create(1);
010      sProjection.Items[0] :=
TJCommonDataKinds_Email.JavaClass.ADDRESS;
011
012      sWhere := TJavaObjectArray<JString>.Create(2);
013      sWhere.Items[0] :=
TJCommonDataKinds_Email.JavaClass.CONTENT_ITEM_TYPE;
014      sWhere.Items[1] := StringToJString(aContact.ID);
015
016      emailCursor :=
jcr.query(TJContactsContract_Data.JavaClass.CONTENT_URI,sProject
ion, StringToJString('mimetype = ? AND lookup = ?'), sWhere, nil);
017      try
018        if emailCursor.getCount > 0 then
019        begin
020          emailCursor.moveToNext;
021          Result := JStringToString(emailCursor.getString(0));
022        end;
023      finally
024        emailCursor.close;
025        emailCursor := nil;
026      end;
027    end;
```

版權所有 請勿翻印

要查詢連絡人的電話使用的方法和剛才查詢 Email 的方式類似，只是我們需要到 ContactsContract.CommonDataKinds.Phone 類別中查詢：

ContactsContract.CommonDataKinds.Phone

extends Object
implements ContactsContract.DataColumnsWithJoins ContactsContract.CommonDataKinds.CommonColumns

java.lang.Object
 ↳android.provider.ContactsContract.CommonDataKinds.Phone

Class Overview

A data kind representing a telephone number.

You can use all columns defined for `ContactsContract.Data` as well as the following aliases.

Column aliases

Type	Alias	Data column
String	NUMBER	DATA1

而 ContactsContract.CommonDataKinds.Phone 類別已經封裝在 Delphi 的 TJCommonDataKinds_Phone 類別中：

版權所有 請勿翻印

```
[JavaSignature('android/provider/ContactsContract$CommonDataKind
s$Phone')]
  JCommonDataKinds_Phone = interface(JObject)
    ['{B3BC4EC6-2FEB-4F10-9D45-275FDE754A78}']
  end;
  TJCommonDataKinds_Phone =
class(TJavaGenericImport<JCommonDataKinds_PhoneClass,
JCommonDataKinds_Phone>) end;
```

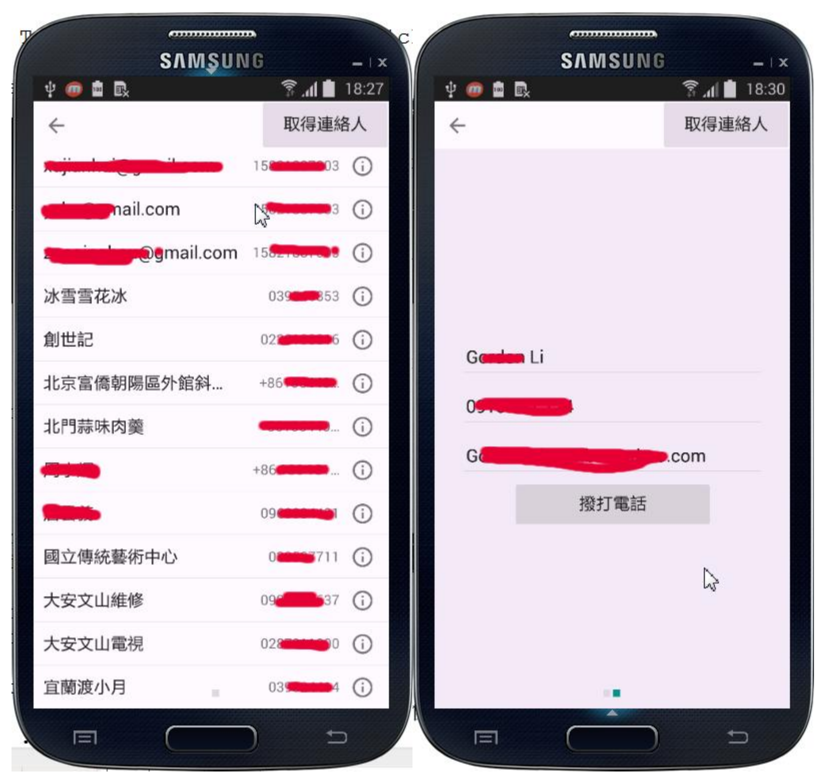
因此 GetContactPhone 方法使用的技巧和 GetContactEMail 方法類似，我們就不再贅述了：

```
001    function TfmMainForm.GetContactPhone(aContact :
TContactPerson): String;
002    var
003      PhoneCursor : JCursor;
004      sProjection : TJavaObjectArray<JString>;
005      sWhere : TJavaObjectArray<JString>;
006
007    begin
```

```
008     sProjection := TJavaObjectArray<JString>.Create(1);
009     sProjection.Items[0] :=
TJCommonDataKinds_Phone.JavaClass.NUMBER;
010
011     sWhere := TJavaObjectArray<JString>.Create(2);
012     sWhere.Items[0] :=
TJCommonDataKinds_Phone.JavaClass.CONTENT_ITEM_TYPE;
013     sWhere.Items[1] := StringToJString(aContact.ID);
014
015     PhoneCursor :=
jcr.query(TJContactsContract_Data.JavaClass.CONTENT_URI,sProject
ion, StringToJString('mimetype = ? AND lookup = ?'), sWhere, nil);
016     try
017       if PhoneCursor.getCount > 0 then
018       begin
019         PhoneCursor.moveToNext;
020         Result := JStringToString(PhoneCursor.getString(0));
021       end;
022     finally
023       PhoneCursor.close;
024       PhoneCursor := nil;
025     end;
026   end;
```

版權所有 請勿翻印

最後編譯範例 App 並執行在第 1 個頁面可查詢到所有連絡人資訊，點選任一連絡人就可以在第 2 個頁面中看到他的電話和 Email 資訊了。



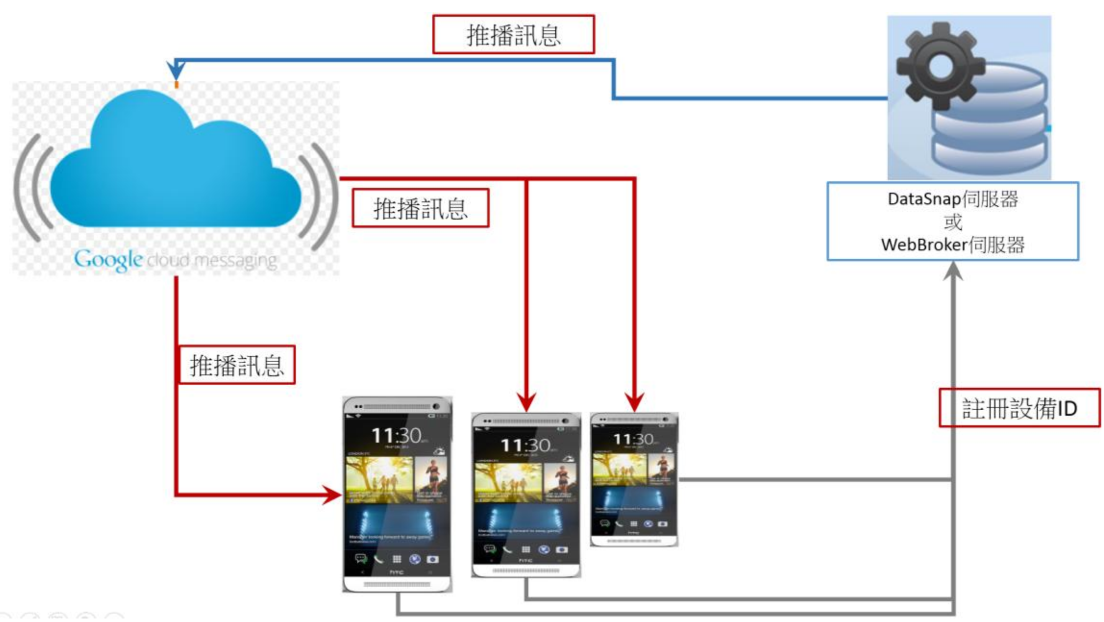
15-4　使用 Google GCM 實作推播功能

版權所有 請勿翻印

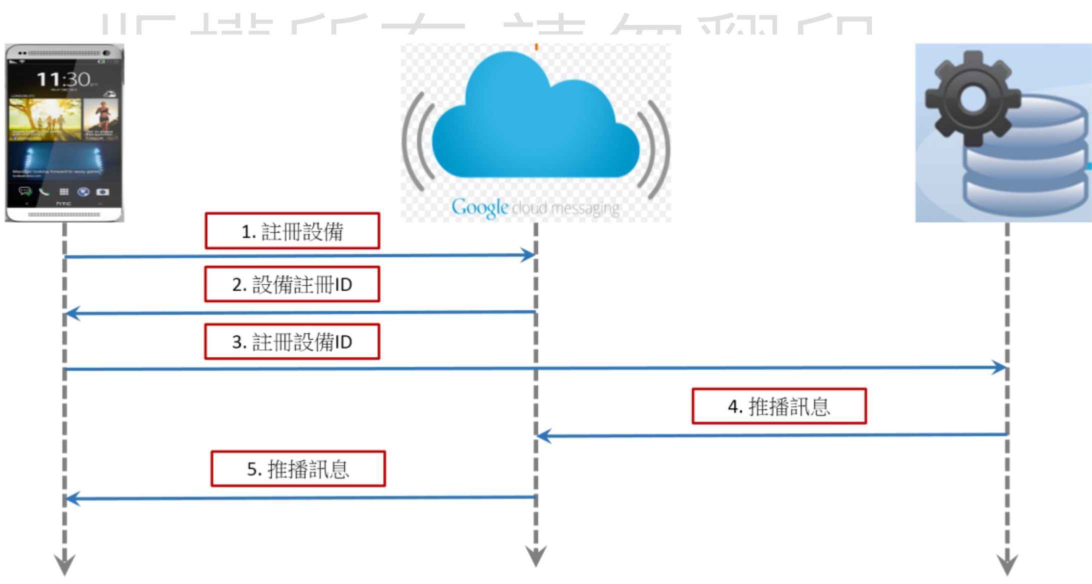
在最後一個小節中讓我們使用一個更複雜範例，那就是如何使用 Delphi 實作推播的功能，這個範例不但需要呼叫 Java 程式碼，也需要從 Java 接收回傳通知。但在說明如何開發之前讓我們簡單的介紹一下 GCM 的架構。

GCM 是 Google Cloud Messaging 的簡寫，它是 Google 提供的免費服務可主動推播訊息給 Android 設備。在 GCM 架構中是由 3 個物件組成的，第 1 即是 GCM 本身，第 2 是用戶端的 Android 設備以及一個伺機器。在 Delphi 技術領域中我們可以使用下圖來說明 GCM 的架構連作。

在 Delphi 中我們可以使用 DataSnap 或是 WenBroker 技術撰寫一個伺服器，這個伺服器可使用 HTTP 推播訊息給 GCM，GCM 就會把此訊息自動推播給已經註冊要接收推播訊息的 Android 用戶端。至於下圖中的 Android 用戶端就是使用 Delphi 撰寫的 App，它要執行工作最多，首先它需要向 GCM 註冊表明要接收推播訊息，接著它也要向 DataSnap/WenBroker 伺服器註冊以便伺服器將來指定要推播訊息給那一個 Android 用戶端。



下圖是這 3 者之間基本的互動流程圖，一開始 Android 用戶端 App 須要先向 GCM 註冊以取得註冊 ID，再把此註冊 ID 向伺服器註冊。接著伺服器向 GCM 推播訊息，在此推播訊息動作中伺服器可指明推播給那一個 Android 用戶端，那一群 Android 用戶端或是所有的 Android 用戶端。



瞭解了 GCM 架構的基本原理後我們就可以開始進行開發的工作了，要完成完整的 GCM 開發我們需要完成 3 個工作，它們是

1. 啟動使用 GCM 功能
2. 開發 GCM 用戶端 App
3. 開發 DataSnap 伺服器

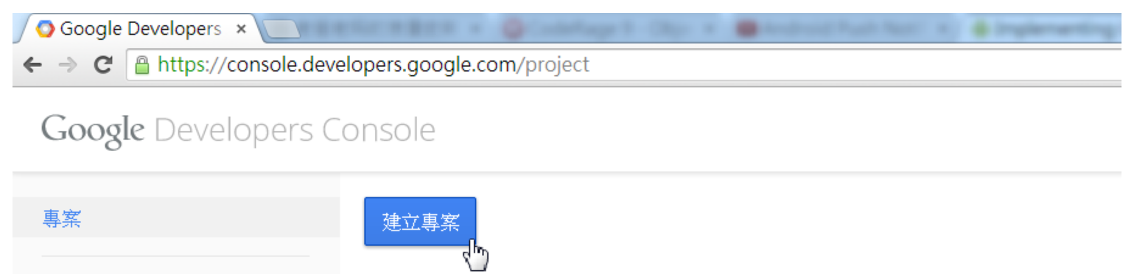
在下面的小節中將分別說明如何完成每一個步驟。

15-4-1　啟動使用 GCM 功能

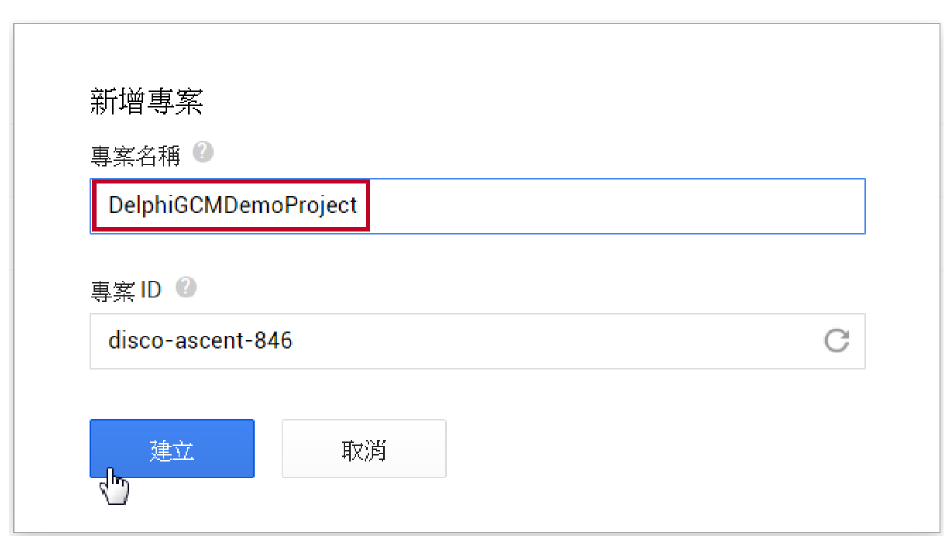
在使用 GCM 之前您必須先到

```
https://console.developers.google.com
```

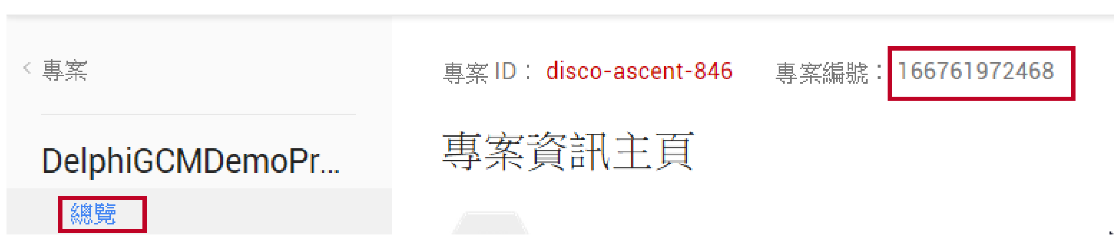
建立專案並開啟 GCM 的功能，請點選下圖中的”建立專案”按鈕：



取一個項目名稱，例如”DelphiGCMDemoProject”，並點選”建立”按鈕：

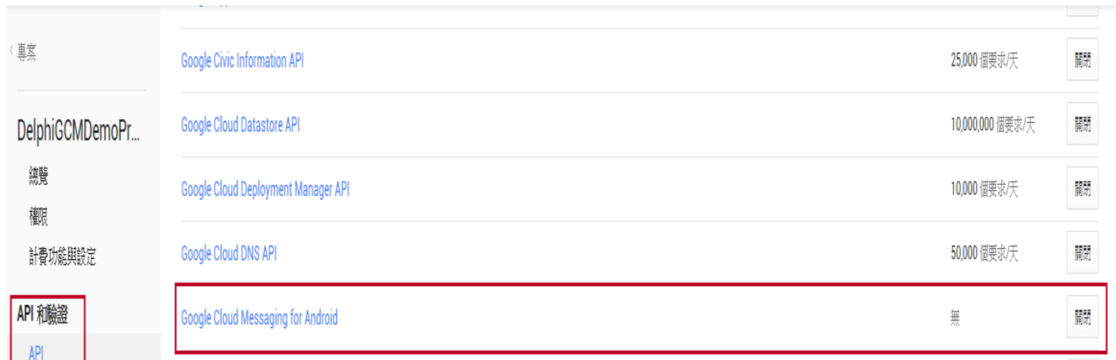


成功建立專案之後在”總覽”頁面可看到一個專案編號，此編號在稍後的開發中非常的重要請保存下來：

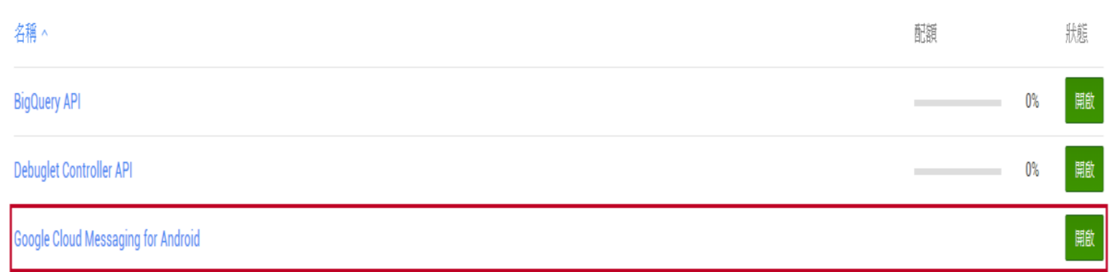


接下來我們要開啟 GCM 的功能，請到 API 和驗證下的 API 子項在右方找到如下圖的 Google Cloud Messaging for Android 並點選右方的"關閉"按鈕以開啟使用這項服務：

開啟



開啟之後在此頁面最前面的已啟用的 API 中應該就可以看到如下圖的"開啟"狀態了：



接著我們需要公開這個 API 讓 Android 的用戶端設備可以使用，請到 API 和驗證下的憑證子項中點選"建立新的金鑰"：

再點選"伺服器金鑰"：

接著會出現下圖的畫面要求您輸入稍後執行 DataSnap 伺服器的硬體 IP 以便 Google 控制流量和進行計費的計算(GCM 在 1000 人以內是免費的)。由於筆者是在個人 PC 中實作這個範例因此可以暫時不輸入 IP：

建立伺服器金鑰並設定允許使用的 IP 位址

請將這個金鑰妥善保存在您的伺服器中，避免外洩。

每個 API 要求都是由您所控管裝置上執行的軟體所產生。 系統會使用每個要求的 userIp 參數 (如有指定) 中 所提供的位址，實行使用者限制。如果要求中 缺少 userIp 參數，系統會改用您的裝置 IP 位址。 瞭解詳情

接受這些伺服器 IP 位址發出的要求

每行一個 IP 位址或子網路。範例：192.168.0.1、172.16.0.0/16、2001:db8::1 或 2001:db8::/64

建立　取消

最後點選上面"建立"按鈕，就會出現下面重要的資訊頁面，其中的 API 金鑰是稍後開發必要的資訊，請和前面的專案編號一樣把它保存下來。

到了這裡我們已經完成了申請，開啟和設定 GCM 服務的工作了，我們可以開始進行實際的開發工作了，但在說明之前請再次確定您已經有了：

- 專案編號
- API 金鑰

15-4-2 開發 GCM 用戶端 App

在開發 GCM 架構中 GCM 用戶端是比較複雜的，因為 GCM 用戶端需要完成下列的數項工作：

1. 向 GCM 註冊用戶端設備並取得設備註冊 ID
2. 向伺服器註冊從 GCM 取得的設備註冊 ID
3. 接收來自 GCM 的通知訊息

其中第 1 項工作需要呼叫 GCM API，這在前面的章節中已經說明如何呼叫 Java 程式碼所以不困難。第 1 項工作需要實作一個 DataSnap 伺服器並呼叫它的服務方法，這也不困難。最困難的是第 3 項工作，因為 GCM 會推播訊息通知給 Java 的回叫函式，我們必須把這個回叫的執行導引到我們的 Delphi 程式碼中。但是記得 Delphi 是一個 RAD 工具，因此只要我們瞭解了 GCM 的工作原理之後再想通如何在 Delphi 中根據這些原理來開發，那麼就很簡單了，而且簡單到您會很吃驚。

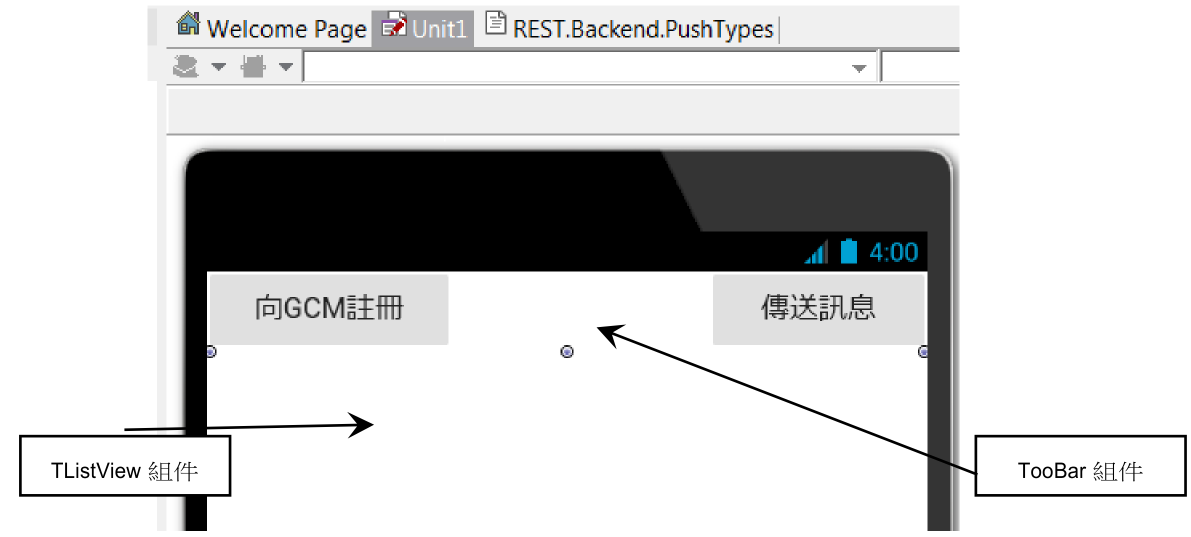
在下面的小節中將一一說明如何完成這上面列出的開發步驟。

版權所有 請勿翻印

15-4-3 建立 Multi-Device 專案

讓我們從簡單的地方開始，先開發一個能自我接受 GCM 訊息的 App，成功之後再加入 DataSnap 伺服器提供所有用戶端設備的功能，最後再開發一個 VCL 用戶端能夠推播訊息到用戶端設備。

在 Delphi IDE 中先建立一個 Multi-Device 專案，在主表單中加入 ToolBar，2 個 TButton 和一個 TListView 元件如下所示：



接著我們就要實作步驟 1 向 GCM 註冊用戶端設備並取得設備註冊 ID，一般來說要完成步驟 1，開發人員需要完成下面 2 項工作：

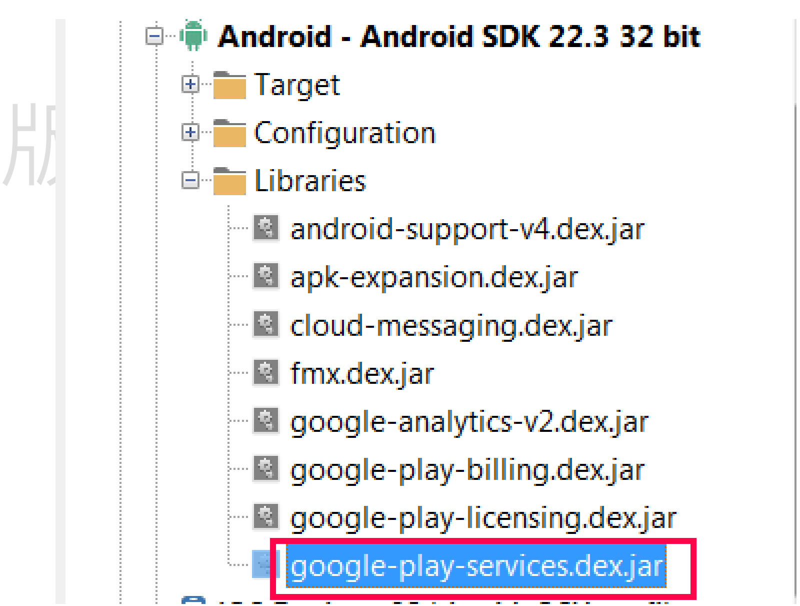
1. 查詢是否有 Google Play Service

2. 有的話就可以向 GCM 註冊

要查詢是否有查詢是否有 Google Play Service，我們可以使用 Delphi 中的 TJGooglePlayServicesUtil 類別，它的 JGooglePlayServicesUtilClass 介面中的 isGooglePlayServicesAvailable 類別方法可提供查詢：

```
{class} function isGooglePlayServicesAvailable(context: JContext):
Integer; cdecl;
```

如果 isGooglePlayServicesAvailable 回傳 True，那應就可以開始註冊的工作，在 RIO 這應該一定回傳 True，因為項目的 Libraries 中已經包含了提供 Google Play Service 的 jar 檔案了：

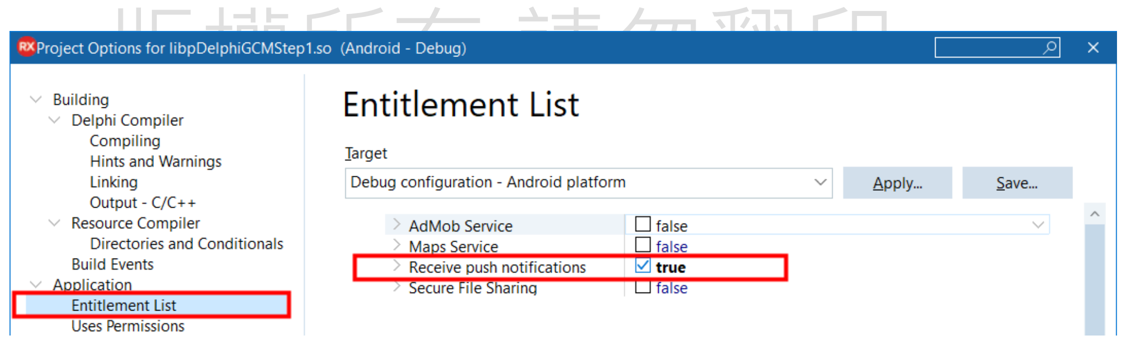


要向 GCM 註冊我們可以使用 TJGoogleCloudMessaging 類別中的 register 方法，它接受一個 TJavaObjectArray<JString>型態的參數：

```
[JavaSignature('com/google/android/gms/gcm/GoogleCloudMessaging'
)]
  JGoogleCloudMessaging = interface(JObject)
    ['{A4F0B5B0-FCB2-45F8-A3CB-D37481F2FBD8}']
```

```
    procedure close; cdecl;
    function getMessageType(intent: JIntent): JString; cdecl;
    function register(senderIds: TJavaObjectArray<JString>):
JString; cdecl;
    procedure send(to_: JString; msgId: JString; data: JBundle);
cdecl; overload;
    procedure send(to_: JString; msgId: JString; timeToLive: Int64;
data: JBundle); cdecl; overload;
    procedure unregister; cdecl;
  end;
  TJGoogleCloudMessaging =
class(TJavaGenericImport<JGoogleCloudMessagingClass,
JGoogleCloudMessaging>) end;
```

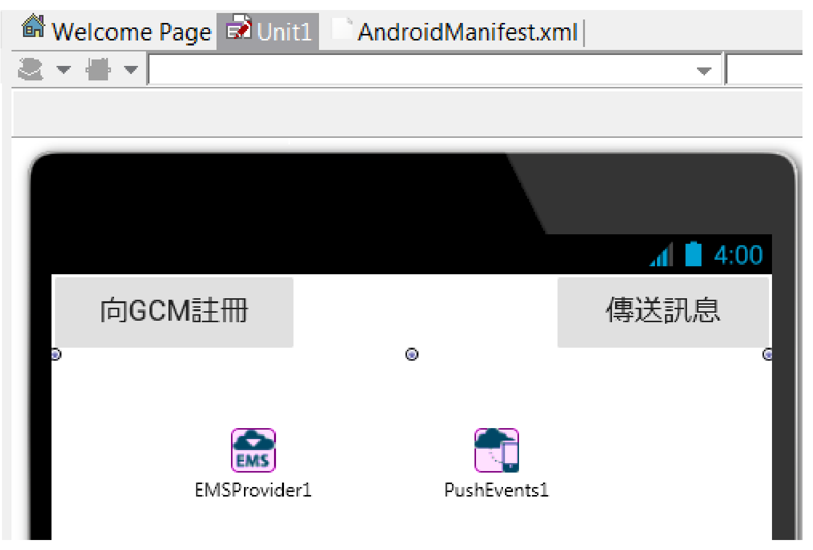
很複雜嗎？別擔心，馬上告訴您非常簡單的方法來完成這個步驟，在解釋之前我們需要先開啟此專案能接收 GCM 訊息的能力，請點選專案，點選滑鼠加鍵選擇 Options... 選項在 Entitlement List 子項中勾選 Receive push notifications，如下所示：



如此一來 Delphi 就會在項目的 AndroidManifest.xml 檔中加入正確的許可權和接收 GCM 訊息的接收器了。

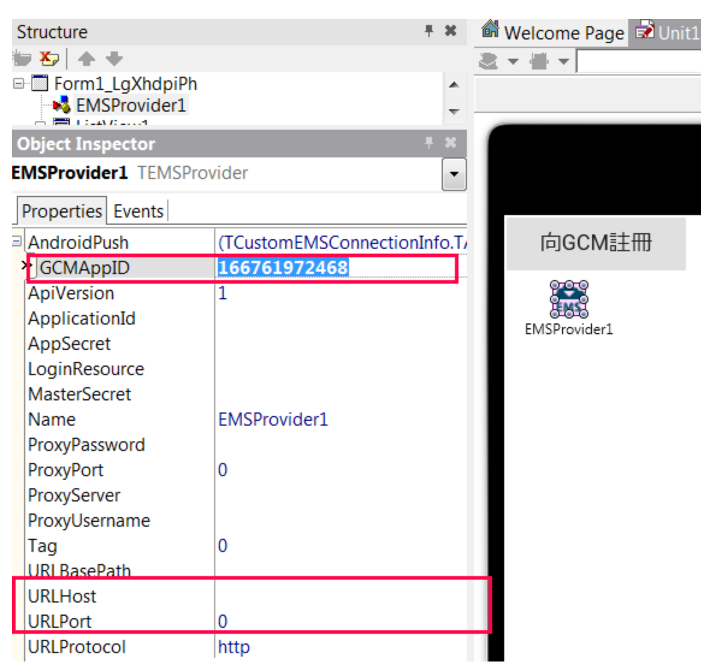
好了，那麼要如何前面的工作呢？非常簡單我們只需要轉個腦筋藉由使用 RIO 中的 BAAS Client 元件就可以了。在 BAAS Client 元件組中有 TEMSProvider 和 TPushEvents 這 2 個元件，雖然 Delphi 的英文手冊中說明 TEMSProvider 元件是連結到 EMS 伺服器使用的，而 TPushEvents 則是需要使用 Parse 或 Kinvey 等第 3 方供應商才能使用的，但不用管手冊說什麼，因為只需要轉個腦筋把 TEMSProvider 元件想成是連結到 Google，把 TPushEvents 想成是連結到 GCM 不就可以了嗎？

所以請在主表單中加入這 2 個元件：



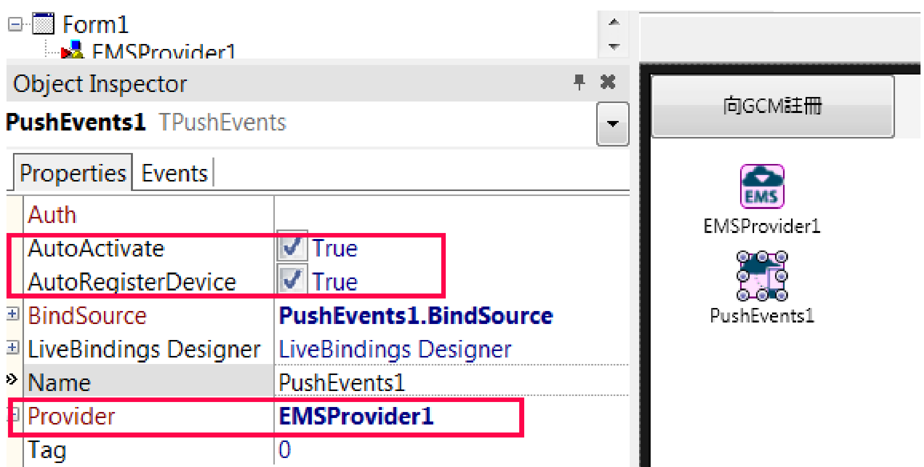
接下來要設定這 2 個元件，首先設定 TEMSProvider 組件。設定 TEMSProvider 元件的關鍵點是完全忽略 TEMSProvider 元件連結 EMS 伺服器的相關設定，我們只需要設定它的 AndroidPush 特性下的 GCMAppID 子特性，GCMAppID 的設定值就是我們前面說明的 Google 項目 ID 值，至於其他的特性，例如 MasterScret，AppSecret 等特性請都留成空白。這樣的設定等於讓 TEMSProvider 元件連結連結到 Google 的 GCM 服務伺服器而不是 EMS 伺服器，如下圖所示：

版權所有 請勿翻印



接著對 TPushEvents 元件進行如下的設定：

特性	設定值
AutoActivate	True
AutoRegisterDevice	True
Provider	EMSProvider1



版權所有 請勿翻印

TPushEvents 元件的 AutoRegisterDevice 特性就可以向 Google GCM 自動註冊用戶端移動設備並取得設備 ID 和 GCM 註冊 ID。這是因為 TPushEvents 元件是從 TCustomPushEvents 類別繼承下來，在 TCustomPushEvents 類別中有一個關鍵的特性 PushConnection，它是 TPushServiceConnection 類別物件：

```
property PushConnection: TPushServiceConnection read
GetPushServiceConnection;
```

另外還有下面 3 個關鍵特性：

```
    property DeviceRegistered: Boolean read GetDeviceRegistered;
    property DeviceToken: string read GetDeviceToken;
    property DeviceID: string read GetDeviceID;
```

上面的 DeviceToken 特性值即是用戶端移動設備的 GCM 註冊 ID，而 DeviceID 特性值則是用戶端移動設備 ID。所以一旦 AutoRegisterDevice 特性值設定為 True 之後，那麼在程式碼中就可以使用下面的方式取得用戶端移動設備的 GCM 註冊 ID：

```
If (PushEvents.DeviceRegistered) then
```

```
sGCMID := PushEvents.DeviceToken;
```

在 PushConnection 中有一 Service 特性它是 TPushService 類別物件：

```
  property Service: TPushService read FService;
```

當我們用 TEMSProvider 連結 Google GCM 時，這個 Service 其實會是 TGCMPushService 物件：

```
TGCMPushService = class(TPushService)
```

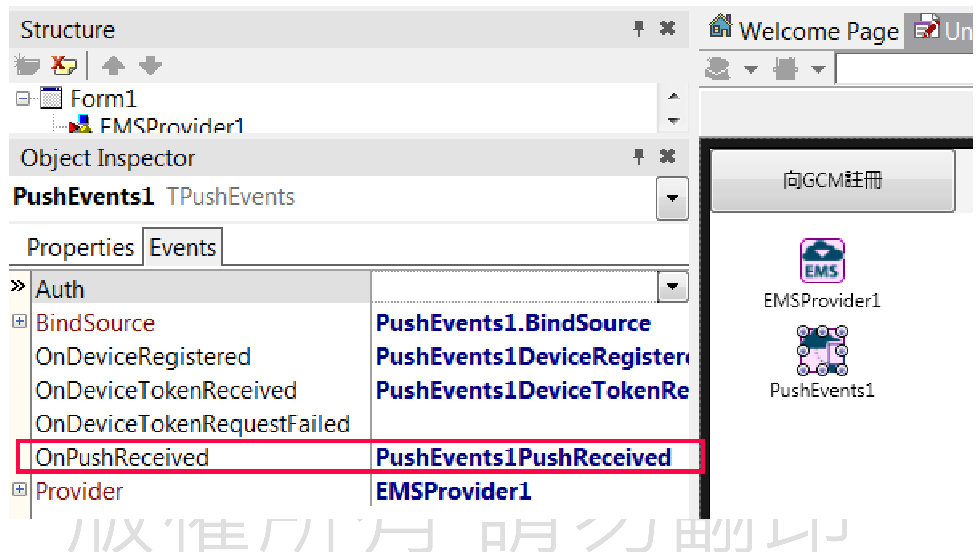
在 TGCMPushService 類別中的 Register 方法就會自動幫助我們呼叫前面說明的使用 JGoogleCloudMessaging 等介面方法向 GCM 註冊：

```
procedure TGCMPushService.Register(const LGCMAppId: string);
var
  LGCM: JGoogleCloudMessaging;
  LSenderIds: TJavaObjectArray<JString>;
  LToken: JString;
begin
  /// prepare sender-ids (this is usually your app-id)
  LSenderIds := TJavaObjectArray<JString>.Create(1);
  LSenderIds.SetRawItem(0, (StringToJString(LGCMAppId) as
ILocalObject).GetObjectID);

  FDeviceToken := '';
  LGCM :=
TJGoogleCloudMessaging.JavaClass.getInstance(SharedActivityConte
xt);
  try
    LToken := LGCM.Register(LSenderIds);
    FDeviceToken := (JStringToString(LToken));
    FStatus := TPushService.TStatus.Started;
    GCMPushService.DoChange([TPushService.TChange.Status,
TPushService.TChange.DeviceToken]);
  except
    on E: Exception do
    begin
      FStatus := TPushService.TStatus.StartupError;
      GCMPushService.FStartupError := E.Message;
```

版權所有 請勿翻印

```
      GCMPushService.DoChange([TPushService.TChange.Status]);
    end;
  end;
end;
```

如何？很棒吧，只要腦筋轉一下就可以叫 Delphi 自動幫忙我們完成 15-3-2 小節中的第 1，2 個步驟啦。最後的第 3 個步驟更簡單完全不需要再使用 Java 程式碼，只要使用 TPushEvents 元件的 OnPushReceived 事件即可：



RIO 之後 Delphi 提供了一個方法可直接把 GCM 傳來的訊息串接到 TPushEvents 元件的 OnPushReceived 事件，那就是 GCMIntentService。因此要讓 OnPushReceived 事件接收 GCM 傳來的訊息，請開啟您的專案目錄下的 AndroidManifestTemplate.xml 檔案(如果沒有的話就請先 Build 一下你的項目即會自動產生)，在它的 application 元素中加入如下的子元素：

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

例如下圖就是筆者在 AndroidManifestTemplate.xml 檔案中加入定的畫面：

```
<%activity%>
<service android:name="com.embarcadero.gcm.notifications.GCMIntentService" />
<receiver android:name="com.embarcadero.firemonkey.notifications.FMXNotificationAlarm" />
<%receivers%>
```

加入了 GCMIntentService 並儲存 AndroidManifestTemplate.xml 檔案後就可以在 OnPushReceived 事件撰寫接收 GCM 訊息的程式碼。OnPushReceived 事件會收到 TPushData 物件參數，其中的 GCM 特性就是 Google GCM 傳來的訊息：

```
procedure TForm1.PushEvents1PushReceived(Sender: TObject;
  const AData: TPushData);
begin
  ListView1.Items.Add.Text := AData.GCM.Title;
  ListView1.Items.Add.Text := AData.GCM.Msg;
  ListView1.Items.Add.Text := AData.GCM.Message;
end;
```

接著我們在主表單的”向 GCM 註冊”按鈕中存取設備 ID 和 GCM 註冊 ID：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (PushEvents1.DeviceRegistered) then
  begin
    ListView1.Items.Add.Text := PushEvents1.DeviceID;
    ListView1.Items.Add.Text := PushEvents1.DeviceToken;
  end;
end;
```

版權所有 請勿翻印

現在在手機中執行此範例 App 並點選”向 GCM 註冊”按鈕就可以看到如下的結果畫面，我們已經可以成功取得這 2 個 ID 了：

接著我們要測試 TPushEvents 元件的 OnPushReceived 事件是否可真的接收 GCM 訊息，那麼我們必須先簡單的實作向 Google GCM 發出訊息再藉由

GCM 服務把訊息傳向用戶端。要向 Google GCM 發出訊息需要遵照 Google 的檔說明：

```
http://developer.android.com/google/gcm/http.html
```

其中規定要向 Google GCM 發送訊息，在 HTTPq 表頭中需要包含如下的資訊：

```
Authorization: key=YOUR_API_KEY
Content-Type: application/json for JSON;
```

上面 YOUR_API_KEY 就是在前面我們申請的”伺服器金鑰”，因此在範例 App 主表單的”傳送訊息”按鈕中我們需要實作如下的程式碼：

```
001   const
002     YOUR_GCM_SENDERID = '166761972468';
003     YOUR_API_ID = 'AIXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXg';
004     MessageTemplate = '{' +
005                       ' "registration_ids" : ["%s"],' +
006                       ' "data" : {"message" : "%s"}' +
007                       '}';
008
009
010   procedure TForm1.Button2Click(Sender: TObject);
011   const
012     sendUrl = 'https://android.googleapis.com/gcm/send';
013   var
014     AuthHeader: STring;
015     idHTTP: TIDHTTP;
016     SSLIOHandler: TIdSSLIOHandlerSocketOpenSSL;
017     lData : TStringStream;
018     lMessage : String;
019   begin
020     idHTTP := TIDHTTP.Create(nil);
021     try
022       SslIOHandler :=
TIdSSLIOHandlerSocketOpenSSL.Create(nil);
023       idHTTP.IOHandler := SSLIOHandler;
024       idHTTP.HTTPOptions := [];
```

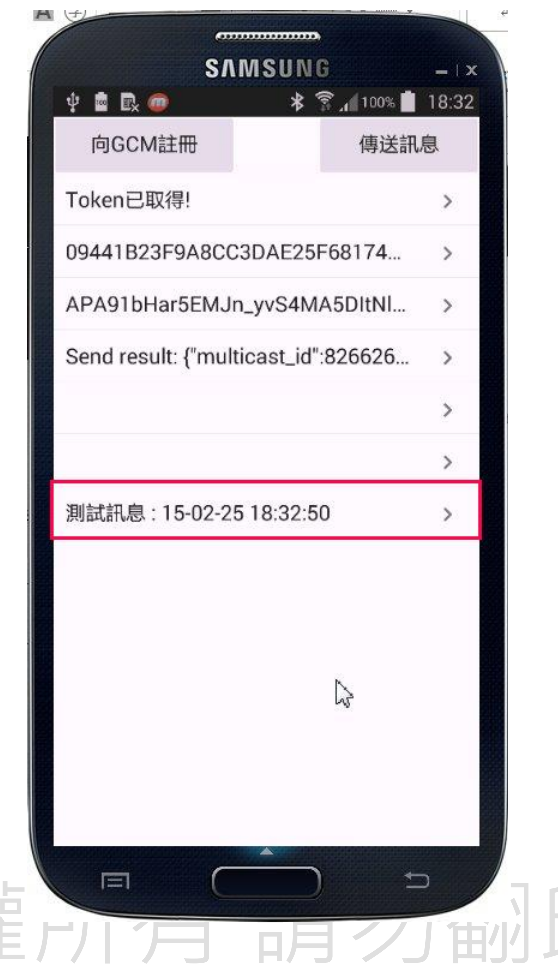
版權所有 請勿翻印

```
025       lMessage := '測試訊息 : ' + FormatDateTime('yy-mm-dd
hh:nn:ss', Now);
026       lMessage := Format(MessageTemplate,
[PushEvents1.DeviceToken, lMessage]);
027       lData := TStringStream.Create(lMessage, TEncoding.UTF8);
028       try
029         idHTTP.Request.Host := sendUrl;
030         AuthHeader := 'Authorization: key=' + YOUR_API_ID;
031         idHTTP.Request.CustomHeaders.Add(AuthHeader);
032         IdHTTP.Request.ContentType := 'application/json';
033         ListView1.Items.Add.Text := 'Send result: ' +
idHTTP.Post(sendUrl, lData);
034       finally
035         lData.Free;
036       end;
037     finally
038       FreeAndNil(idHTTP);
039     end;
040   end;
```

版權所有 請勿翻印

在上面的 030~032 就是在 HTTP 表頭加入必要的資訊，033 行藉由 TIDHTTP 組件提出 Post 請求把要傳遞到用戶端的資訊傳給 GCM 服務。

再次執行範例 App，點選” 傳送訊息”按鈕我們就可以看到類似如下的結果畫面，TPushEvents 元件的 OnPushReceived 事件果然收到了 GCM 傳來的訊息：



版權所有 請勿翻印

瞭解 GCM 原理並且轉動腦筋之後我們幾乎不用寫任何程式碼就可以完成 GCM 用戶端的開發工作了，接下來讓我們再完成 GCM 伺服端的開發以便把整個 GCM 架構串連起來。

15-4-4 開發 DataSnap 伺服器

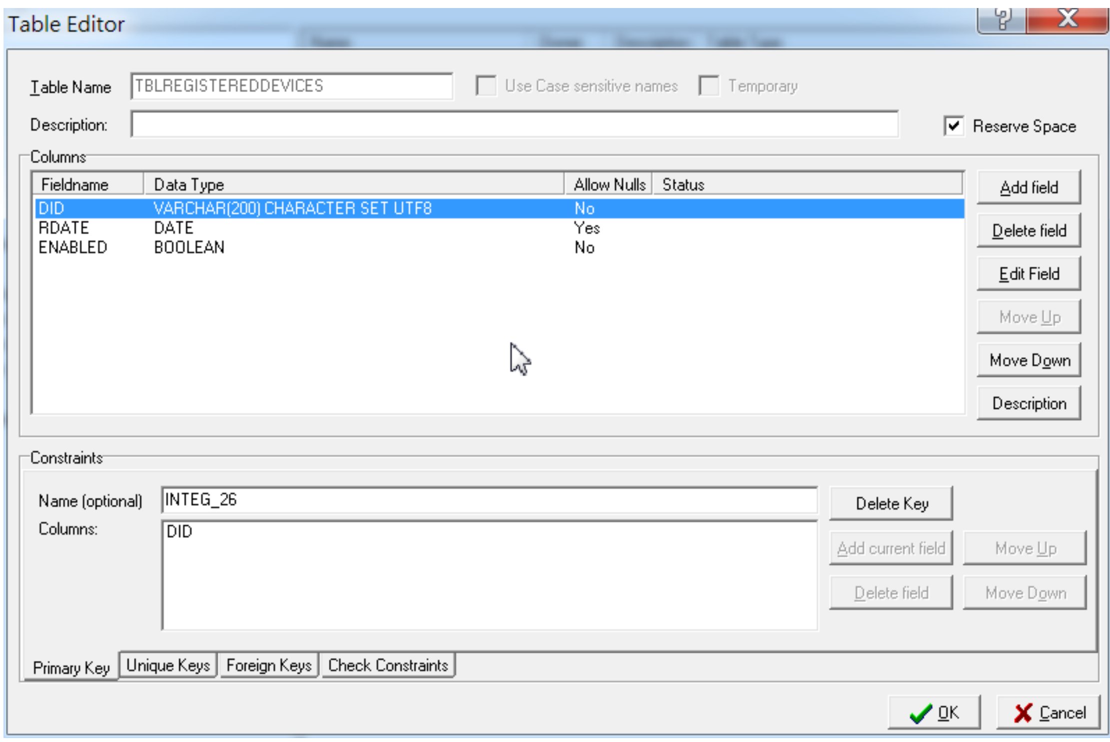
現在我們需要一個伺服器，它主要有 2 個工作：

1. 讓用戶端設備註冊，以便伺服器可要求 GCM 服務向那一個用戶端設備傳遞訊息

2. 向 GCM 服務發出訊息以便傳遞到指定的用戶端設備

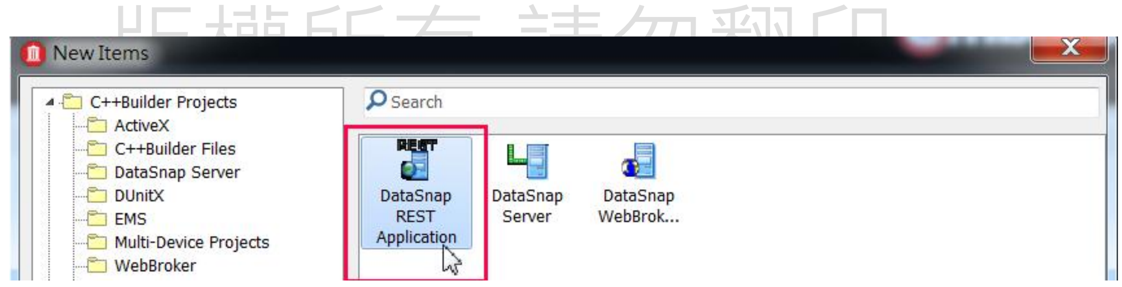
在本小節中為了說明方便我們將使用 DataSnap 開發一個 HTTP 型態的伺服器，讀者可以在瞭解之後自行開發其他型態的伺服器，例如高效率的 DataSnap 伺服器，RESTful 型態的伺服器，或是 CCS(XMPP) 型態的伺服器。

首先讓我們建立一個 InterBase 範例資料表 TBLREGISTEREDDEVICES 以便儲存用戶端設備註冊的 GCM 註冊 ID：

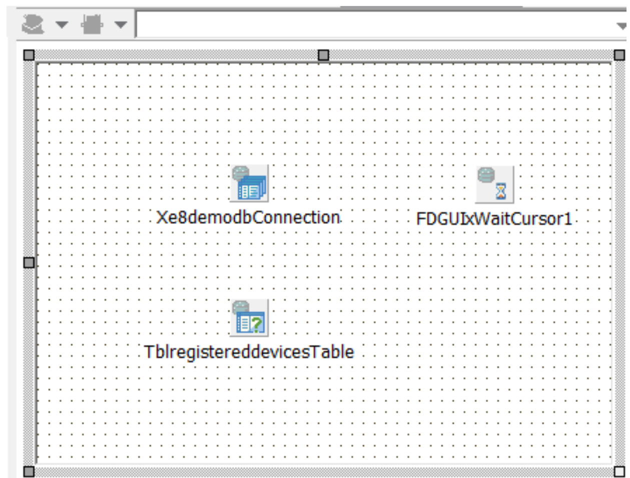


接著建立一個 DataSnap REST 伺服器：

版權所有 請勿翻印



再于其中建立一個資料模組並放入 FireDAC 元件連結到 TBLREGISTEREDDEVICES 範例資料表：



範例 DataSnap REST 伺服器的 ServerMethodUnit 中提供 3 個服務方法，RegisterdeviceID 可讓用戶端設備註冊，PushMessageTo 可推播訊息到特定的用戶端設備，而 PushMessageToAll 則可推播訊息到到所有註冊的用戶端設備：

```
    function RegisterdeviceID(const sID : String) : Boolean;
    function PushMessageTo(const sDID : String; const sMessage :
String) : Boolean;
    function PushMessageToAll(const sMessage: String) : Boolean;
```

版權所有 請勿翻印

RegisterdeviceID 方法把用戶端設備傳遞來的 GCM 註冊 ID 寫入到 TBLREGISTEREDDEVICES 範例資料表中，由於 DataSnap REST 伺服器可服務多個用戶端，因此每一用戶端在寫入時要進行同步控制：

```
001    function TServerMethods1.RegisterdeviceID(const sID:
String) : Boolean;
002    begin
003      TMonitor.Enter(dmPushDemo);
004      try
005        Result := dmPushDemo.RegisterdeviceID(sID);
006      finally
007        TMonitor.Exit(dmPushDemo);
008      end;
009    end;
```

PushMessageTo 方法則使用了前面範例 App 相同的技術藉由 Indy 組件向 Google 的 GCM 提出推播訊息請求：

```
001    function TServerMethods1.PushMessageTo(const sDID,
```

```
sMessage: String) : Boolean;
002    var
003      AuthHeader: STring;
004      idHTTP: TIDHTTP;
005      SSLIOHandler: TIdSSLIOHandlerSocketOpenSSL;
006      lData : TStringStream;
007      lMessage : String;
008      lResult : String;
009    begin
010      idHTTP := TIDHTTP.Create(nil);
011      try
012        SslIOHandler :=
TIdSSLIOHandlerSocketOpenSSL.Create(nil);
013        idHTTP.IOHandler := SSLIOHandler;
014        idHTTP.HTTPOptions := [];
015        lMessage := Format(MessageTemplate, [sDID, sMessage + '
-傳遞時間 : ' + FormatDateTime('yy-mm-dd hh:nn:ss', Now)]);
016        lData := TStringStream.Create(lMessage, TEncoding.UTF8);
017        try
018          idHTTP.Request.Host := sendUrl;
019          AuthHeader := 'Authorization: key=' + YOUR_API_ID;
020          idHTTP.Request.CustomHeaders.Add(AuthHeader);
021          IdHTTP.Request.ContentType := 'application/json';
022          lResult := idHTTP.Post(sendUrl, lData);
023          Result := IsItSuccessful(lResult);
024        finally
025          lData.Free;
026        end;
027      finally
028        FreeAndNil(idHTTP);
029      end;
030    end;
```

版權所有 請勿翻印

最後的 PushMessageToAll 方法從一一取出 TBLREGISTEREDDEVICES 範例資料表中每一個註冊的設備 ID，再呼叫 PushMessageTo 方法傳遞訊息給每一個用戶端設備：

```
001    function TServerMethods1.PushMessageToAll(const sMessage:
String) : Boolean;
```

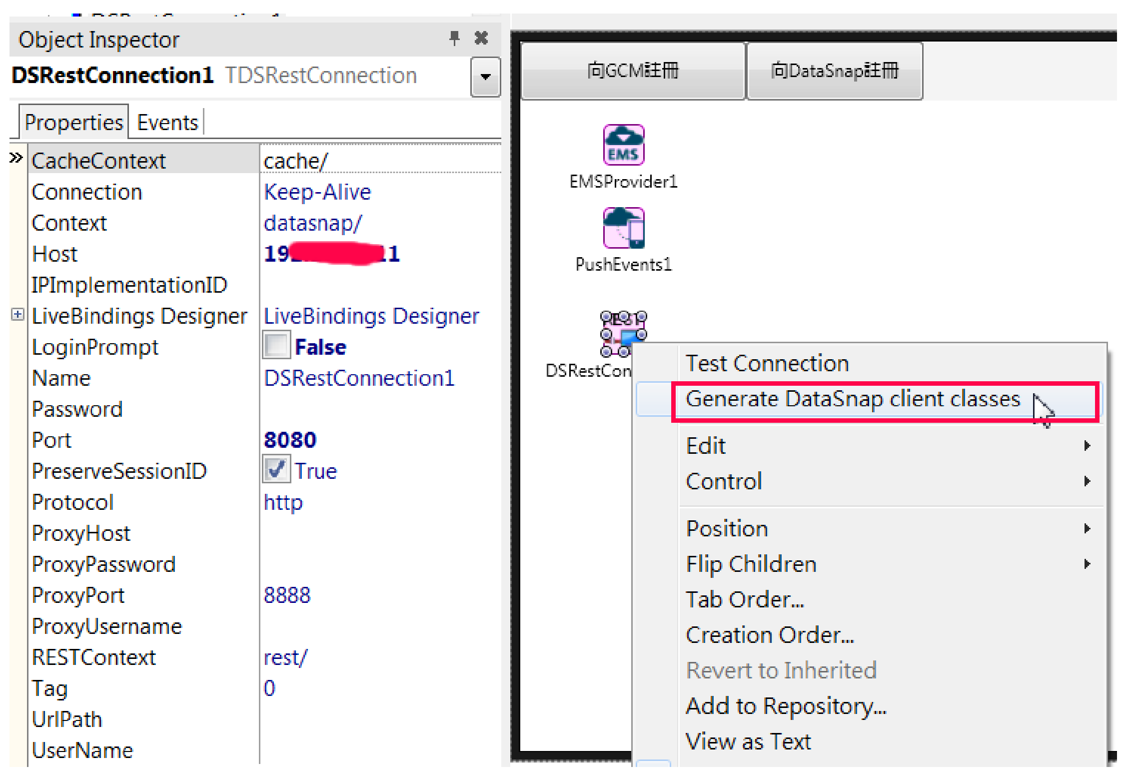
```
002    var
003      sDID : String;
004
005      function PushToAll : Boolean;
006      begin
007        Result := True;
008        if (not dmPushDemo.TblregistereddevicesTable.Active)
then
009          dmPushDemo.TblregistereddevicesTable.Active := True
010        else
011          dmPushDemo.TblregistereddevicesTable.First;
012        while (not dmPushDemo.TblregistereddevicesTable.Eof) do
013        begin
014          sDID :=
dmPushDemo.TblregistereddevicesTable.FieldByName('DID').Value;
015          Result := PushMessageTo(sDID, sMessage) and Result;
016          dmPushDemo.TblregistereddevicesTable.Next;
017        end;
018      end;
019
020    begin
021      TMonitor.Enter(dmPushDemo);
022      try
023        Result := PushToAll;
024      finally
025        TMonitor.Exit(dmPushDemo);
026      end;
027    end;
```

版權所有 請勿翻印

現在請執行此範例 DataSnap REST 伺服器準備讓用戶端設備註冊並推播訊息給用戶端設備。

15-4-5 向 DataSnap 伺服器註冊設備 ID

回到前面的範例 App 準備加入向 DataSnap REST 伺服器註冊 GCM ID 的功能。首先要讓範例 App 連結範例 DataSnap REST 伺服器，請在主表單中加入 TDSRestConnection 元件，設定 DataSnap REST 伺服器的 IP 位址再使用滑鼠右鍵產生用戶端連結程式碼：



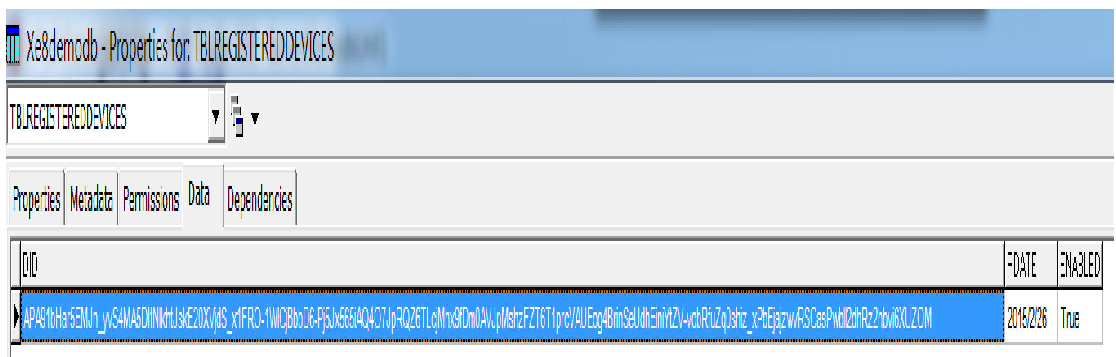
接著在” 向 DataSnap 註冊”按鈕中使用用戶端連結程式碼呼叫 DataSnap REST 伺服器提供的 RegisterdeviceID 方法註冊：

版權所有 請勿翻印

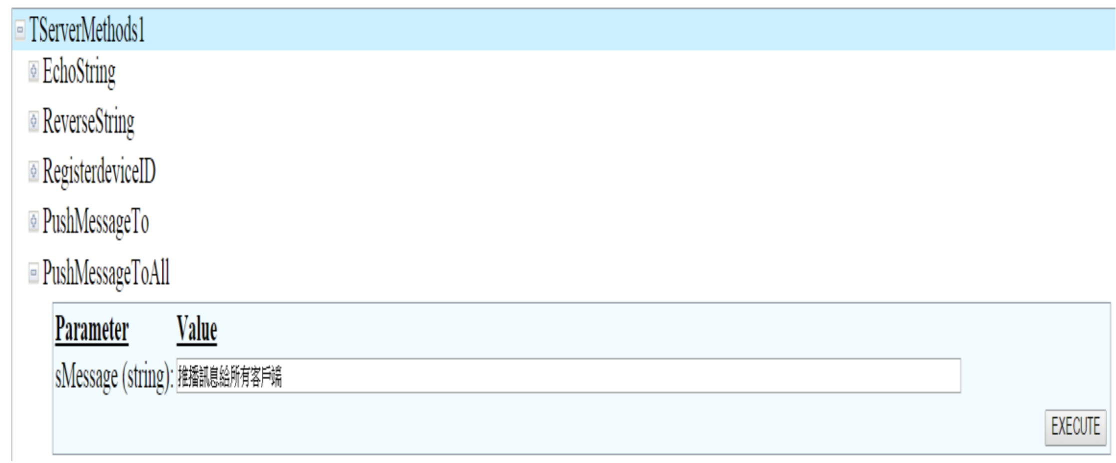
```
001    procedure TfmMainForm.btnRegisterToServerClick(Sender:
TObject);
002    var
003      aServer: TServerMethods1Client;
004    begin
005      aServer :=
TServerMethods1Client.Create(Self.DSRestConnection1);
006      try
007        if (aServer.RegisterdeviceID(PushEvents1.DeviceToken))
then
008          ListView1.Items.Add.Text := '註冊成功';
009      finally
010        aServer.Free;
011      end;
012    end;
```

執行範例 App 點選” 向 DataSnap 註冊”按鈕就可以看到下面”註冊成功”的訊息：

此時如果開啟 TBLREGISTEREDDEVICES 範例資料表就可以看到用戶端設備的 GCM ID 已經成功寫入了：



此時如果使用流覽器執行 PushMessageToAll 方法如下所示：

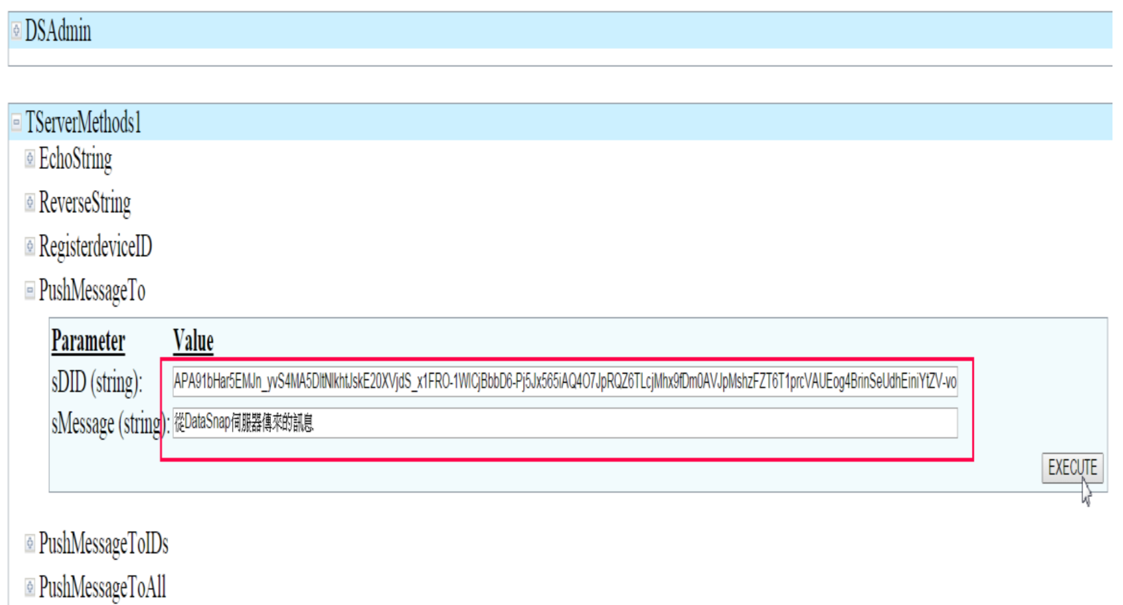


那就應該會在所有註冊的用戶端設備收到推播訊息，例如下面的 2 個畫面就是筆者的 Samsung S4 和 HTC M8 收到範例 DataSnap REST 伺服器藉由 GCM 推播來的訊息：

SAMSUNG
11:28 3月 2日, 週一
Wi-Fi
位置
聲音
畫面
旋轉
藍牙
+1
自動
通知
清除
作為媒體裝置連接
輕觸 USB 選項。
無 SIM 卡
請插入 SIM 卡
Mobizen 11:27
遠端連接WIN-ROTR0B6E9D9和WiFi
pDelphiGCMStep1 11:25
推播訊息給所有客戶端 -傳遞時間 : 15-03-02 ..
僅限緊急通話

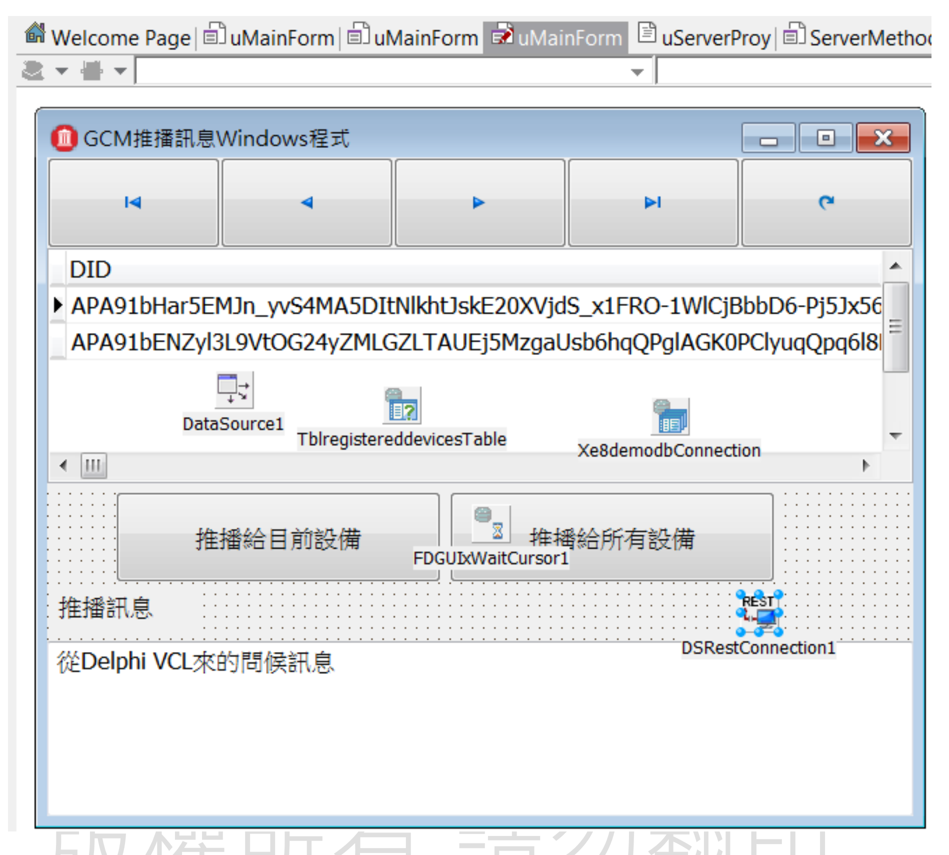
上午11:25
向GCM註冊
向伺服器註冊
傳送訊息
12EBDB10677ABD41CF9D2427AE3BA168
APA91bENZyl3L9VtOG24yZMLGZLTAUEj5MzgaUsb6hqQPglAG...
註冊成功
推播訊息給所有客戶端 -傳遞時間 : 15-03-02 11:23:01

當然如果執行 PushMessageTo 方法也可以如下圖推播訊息給特定的用戶端設備：



15-4-6 開發 Windows 用戶端

要開發一個可推播訊息的 Windows 用戶端也很簡單，我們只要使用前面討論的技術，仍然藉由 TDSRestConnection 元件呼叫範例 DataSnap 伺服器的服務方法即可完成，例如下面的 2 個畫面就是範例 Windows 用戶端推播訊息給手機的結果：



版權所有 請勿翻印

讀者需要注意的是 Google 在 2018 年 4 月 10 日停止開發 GCM，並且將在 2019 年 4 月 11 日從 SDK 中移除 GCM 伺服端和用戶端的 API，但根據目前已有的檔來看 GCM 的應用程式將可繼續執行，只是 2019 年 4 月 11 日之後就不能再開發新的 GCM 應用程式了。

現在 Google 要求開發人員轉移使用 FireCloud 技術，詳情可參考:

https://firebase.google.com/docs/cloud-messaging/

目前 Delphi 的確可以藉由第 3 方函式庫開發 FireCloud 的應用程式，筆者也已經成功使用 Delphi 10.2 和 10.3 開發出 FireCloud 的應用程式，未來 Embarcadero 可能直接會在 RAD Studio 中支援 Google 的 FireCloud 技術。

16 物聯網開發

物聯網(IoT, Internet of Things)技術正如火如荼的席捲資訊系統的開發，特別在各種穿戴式設備於 2015 年逐漸成熟和成為主流用戶端設備之後，把所有設備都連結起來提供更方便，更豐富，更全面和更及時的資訊就成為現今資訊系統最重要的目前，而 Delphi 從 RIO 版就開始提供強大的物聯網開發功能。

由於物聯網包含的設備眾多，在本小節中我們將討論如何使用 Delphi 開發可使用 Beacon 設備的物聯網架構。

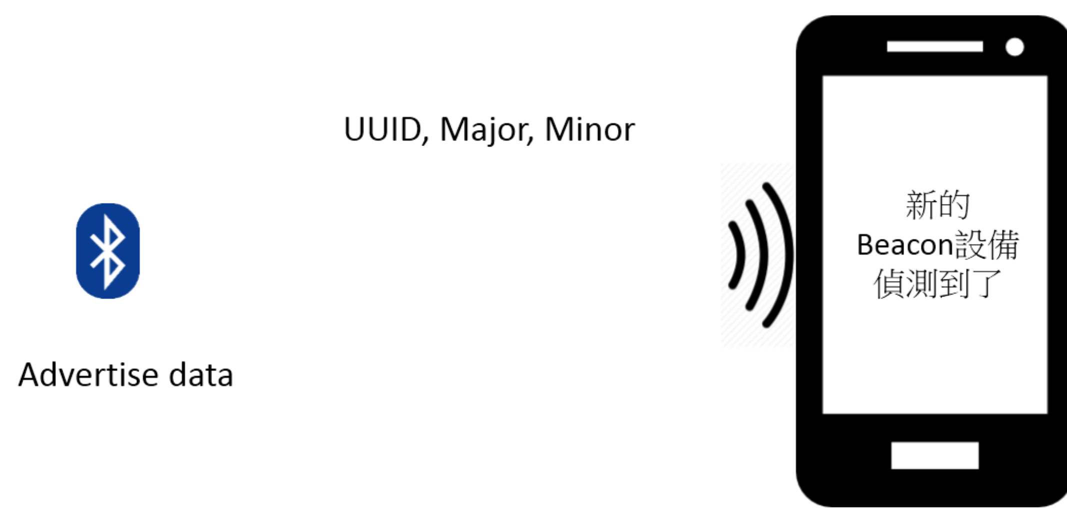
16-1 什麼是 Beancon 技術

版權所有 請勿翻印

Beacon 是一個可提供製造商特定資料的低功耗藍牙技術的設備，當 App 進入特定 Beacon 設備的範圍入之後就可以進行連結，當 App 進入到 Beacon 有效的範圍內，Beacon 就會發送一串識別資訊給 App， App 偵測到識別資訊後便會觸發一連串的動作，例如是從雲端查詢/下載資訊，也可能是開啟其他 App 或連動裝置。目前在市面上已可購買到 Beacon 設備，例如下圖就是一些 Beacon 設備：

一般來說 Beacon 則可將定位範圍精准到 2～100 公尺內，但 Beacon 會受到實體物件的影響，例如牆面，桌子，皮包等等的阻隔而影響信號的強度，有效距離和精確度。

當 App 進入特定 Beacon 設備的信號範圍內時，App 可以取得 Beacon 設備的識別資訊，例如 UUID，Beacon 設備的 Major ID 和 Minor ID，如下圖所示：



因此 App 可以精確的掌握位置和此 Beacon 設備的詳細資訊，接著 App 便可以根據這些資訊連結到雲端或是遠端伺服器查詢資料，或是由端伺服器主動推播資料給 App。

版權所有 請勿翻印

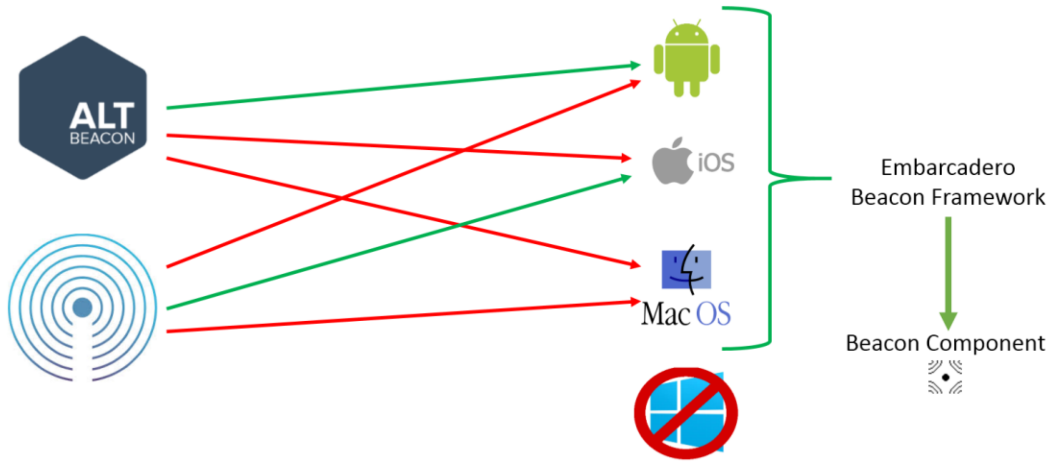
16-2 Beancon 種類

目前 Beacon 技術分為 2 大類，但彼此非常的相像只有一點點的不同，它們是：

種類	說明
iBeacon	使用 Apple 格式的設備，使用 iBeacon 格式的設備必須先向 Apple 註冊
AltBeacon	開放的格式，它的規格公開在 http://altbeacon.org/

Delphi 在 RIO 版開始同時支持 iBeacon 和 AltBeacon。

由於 Beacon 技術是根基于低功耗藍牙技術，因此不是每一個平臺都可支援 iBeacon/AltBeacon，下面的圖形說明了目前在 RIO 中每一個平臺對於支援 Beacon 技術的現況：



從上圖可知由於 Windows 平台對於 iBeacon/AltBeacon 的限制，因此 Windows 平臺尚不能開發 Beacon 的 App。

此外 Delphi 雖然在 XE7 便可以支援低功耗藍牙技術的開發，但由於 Apple 限制要開發 iBeacon App 必須使用 Core Location Services API 而不能用 BLE API，因此 RIO 提供了新的 Beacon 框架和 Beacon 元件來說明 Delphi 開發人員開發 iBeacon/AltBeacon 的 App。

版權所有 請勿翻印

16-3 Beancon 資料格式和意義

每一個 Beacon 設備都需要提供如下的資料：

欄位	說明
UUID	獨特的 ID，代表唯一公司的 Beacon 設備
Major ID	代表唯一公司的 Beacon 設備中的特定 Beacon 群組
Minor ID	代表唯一公司的 Beacon 設備中的特定 Beacon 群組中的特定 Beacon 設備
TxPower	此常數值代表從一公尺接收 Beacon 設備的信號強度，TxPower 結合 RSSI(Received Signal Strength Indicator)後即可

Delphi App 可在取得 UUID，Major ID 和 Minor ID 後掃瞄 TxPower 和 RSSI 值來決定距離 Beacon 設備有 3 遠。

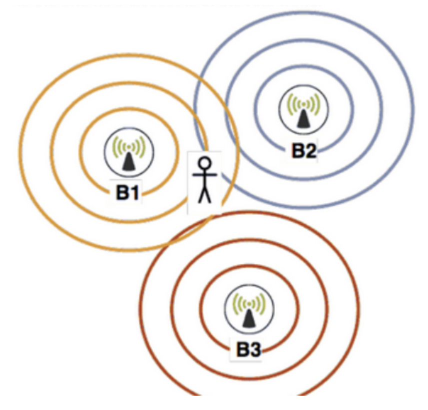
例如下面是 Beacon 設備可提供資料的範例，Beacon 設使用一個唯一的 UUID 來識別特定公司，用 Major ID 做為地區的識別，最後用 Minor ID 做為不同服裝部門的識別：

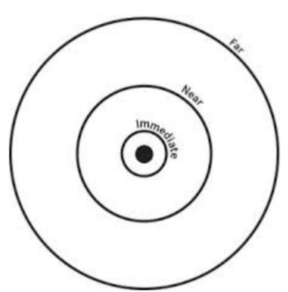
百貨地點		臺北	台中	高雄
UUID		1D614A54-0149-4361-9BF1-42389A2AE58B		
Major		1	2	3
Minor	男裝	10	10	10
	女裝	20	20	20
	童裝	30	30	30

因此當 Delphi App 接近這些 Beacon 設備並取得 UUID，Major ID 和 Minor ID 後充就可以知道是在什麼地方的什麼部門，接著就可以再藉由例如 RESTful 技術連到遠端伺服器，例如 DataSnap，取得此部門的優惠活動資訊。

在一個場合中多個 Beacon 設備可以形成一個所謂的 Region，當然也可以同時形成多個 Regions。例如我們可以 Major ID 做為 Region 的識別，而 Proximity 則代表 App 對於 Region 或是特定 Beacon 設備的距離：

Regions 和Proximity





RSSI
Received Signal Strength Indicator

下面的表格實際的列出了 iBeacon/AltBeacon 的資料格式：

長度	說明	iBeacon範例	AltBeacon範例
1 個位元組	數據長度	1A	1B
1 個位元組	FF(一定是 FF,代表是製造商資料)	FF	FF

2 個位元組	公司 ID	4C 00 *APPLE INC	18 01 *Creative Technology Ltd.
2 個位元組	BEACON 型態	02 15 *iBEACON	BE AC *Beac-on
16 個位元組	UUID	2F 23 44 54 CF 6D 4A 0F AD F2 F4 91 1B A9 FF A6	2F 23 44 54 CF 6D 4A 0F AD F2 F4 91 1B A9 FF A6
2 個位元組	Major ID – Beacon 群組	00 01	00 01
2 個位元組	Minor ID – Beacon 單位	00 01	00 01
1 個位元組	TX Power – Rssi	BE	BE
1 個位元組	製造商保留資料(iBeacon 無此資料)	無	00

16-4 Beancon 接近狀態

當 App 接近 Beacon 設備會根據 Proximity 的值來決定 App 和 Beacon 設備的距離，一般來說可分為 4 種狀態：

接近狀態	說明
Immediate	App 距離 Beacon 設備 0 到 0.5 公尺
Near	App 距離 Beacon 設備 0.5 到 1.5 公尺
Far	App 距離 Beacon 設備 1.5 公尺以外
Unkonw	未知距離(可能太遙遠或是有東西阻擋訊號)

版權所有 請勿翻印

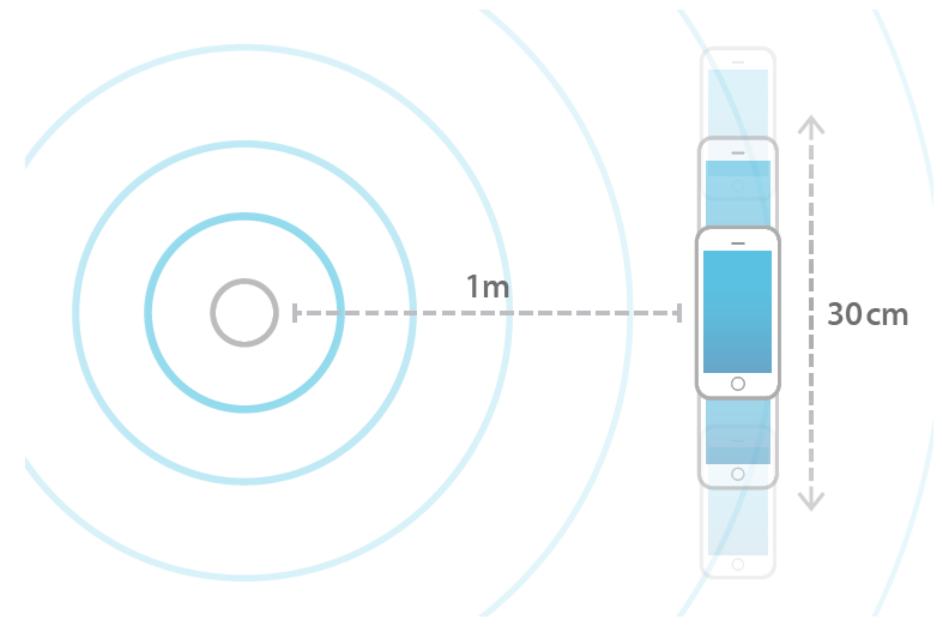
因此 App 可根據 Proximity 來決定要採取什麼行動。

16-5 Beancon 設備校正

在開發 Beacon App 時我們需要對 Beancon 設備進行校正的工作以取得精確的 RSSI 數值好計算 App 距離 Beancon 設備有多遠。要校正 Beancon 設備 Apple 建議使用下面的步驟：

1. 安裝 Beancon 設備並讓它開發發射訊號
2. 使用移動設備並執行 Bluetooth 4.0 radio 在距離 Beancon 設備 1 公尺左右花至少 10 秒時間測量信號強度
3. 以如下圖的方式緩慢前後移動手機/平板等移動設備 30 分分
4. 取得 RSSI 值
5. 平均計算 RSSI 值

6. 在 Beancon 設備組態資訊中記錄此 RSSI 值

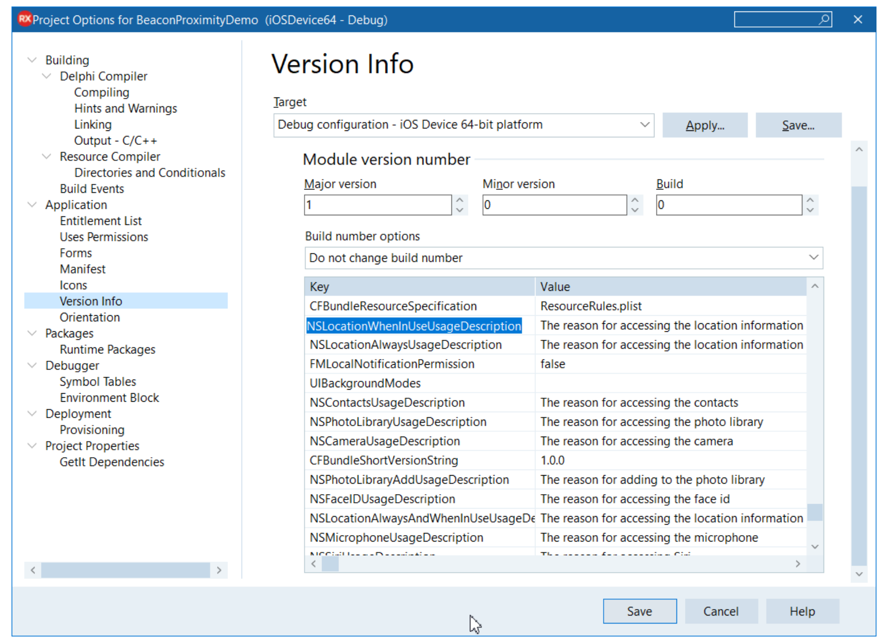


16-6 使用 Apple iBeancon

版權所有 請勿翻印

如果讀者是使用 Apple 的 iBeacon 標準設備，那就需要進行一些額外的設定才能順利使用。

請在 IDE 中開啟專案的選項定，在 version Info 選項中可以看到下的畫面:



請設定其中的下面 3 項子資訊:

➢ NSLocationWhenInUseUsageDescription

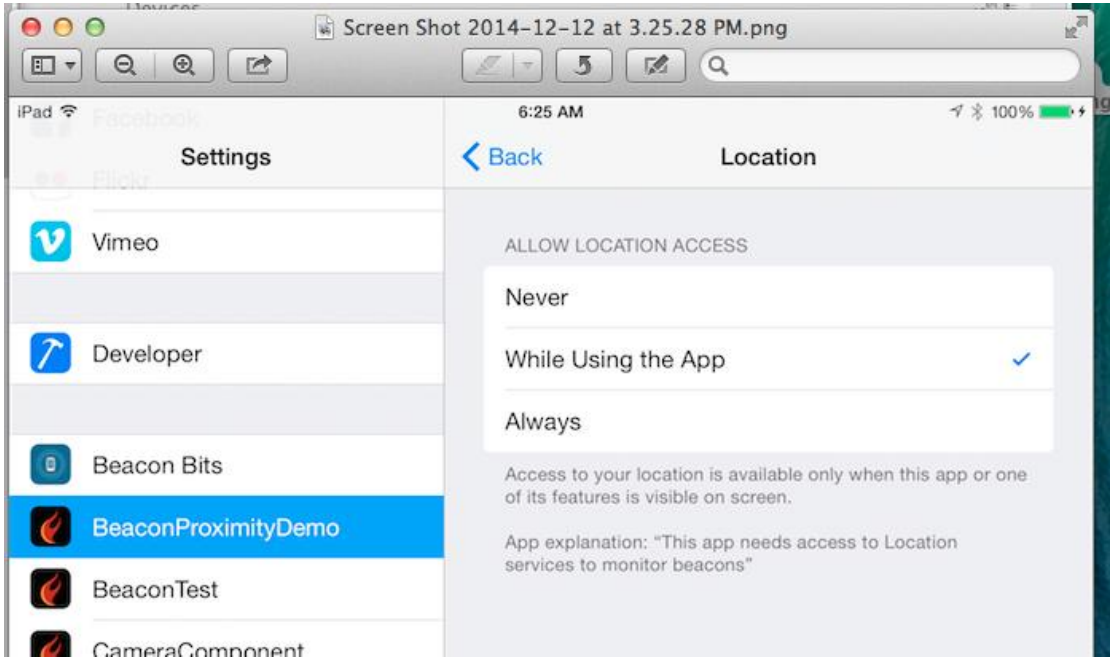
➢ NSLocationUsageDescription

➢ NSLocationAlwaysUsageDescription

請在這 3 項子資訊中輸入說明文字，例如：

```
"This app needs access to Location services to monitor beacons"
```

那麼在 iOS App 的組態資訊中就可以看到說明文字了:



16-7 開發 Beacon App 和物聯網應用架構

在對什麼是 Beacon 以及對 Beacon 和 Beacon 設備有了基本的瞭解之後我們就可以開始使用 Delphi 來開發物聯網型態的 App 了。

在使用 Delphi 開發 Beacon App 時基本上開發人員需要掌握 2 個基本能力：

➢ 自動偵測 Beacon 設備 ：第 1 種開發物聯網應用的架構是自動偵測附近的 Beacon 設備雖然再透過網路根據附近的 Beacon 設備到雲端或是到遠端伺服器查詢資料或服務。

➢ 開發已知 Beacon 設備 App：第 2 種開發物聯網應用的架構是一個公司或企業已經知道使用所有的 Beacon 設備，並只允許 App 偵測這些 Beacon 設備來定位並提供服務。

掌握了上面 2 項技術之後就可以再結合其他技術來開發連網應用架構了。在下面的小節中將一一說明如何完成上述的每一項開發工作。

16-7-1 自動偵測 Beacon 設備

如果要開發的物聯網 App 事先不知道會使用什麼 Beacon 設備，那麼 App 就需要先自動偵測附近的 Beacon 設備，再對偵測到的 Beacon 設備進行掃瞄/監督等後續的處理。

由於 Beacon 設備就是一種低功耗藍牙設備，因此在 RIO 中我們可以使用 TBluetoothLEManager 類別來自動偵測 Beacon 設備，再把偵測到的 Beacon 設備註冊給 TBeacon 元件進行掃瞄/監督等後續的處理就可以很簡單的完成物聯網 App 的開發工作。在本小節中我們將說明如何使用 TBluetoothLEManager 類別來自動偵測 Beacon 設備，於下一小節再說明如何使用 TBeacon 元件進一步的處理。

版權所有 請勿翻印

TBluetoothLEManager 類別提供了 2 個事件讓開發人員可以偵測找到的低功耗藍牙設備，其中 OnDiscoveryEnd 事件會在所有低功耗藍牙設備偵測完畢後觸發而 OnDiscoverLEDevice 事件則會在每一個低功耗藍牙設備偵測到時觸發：

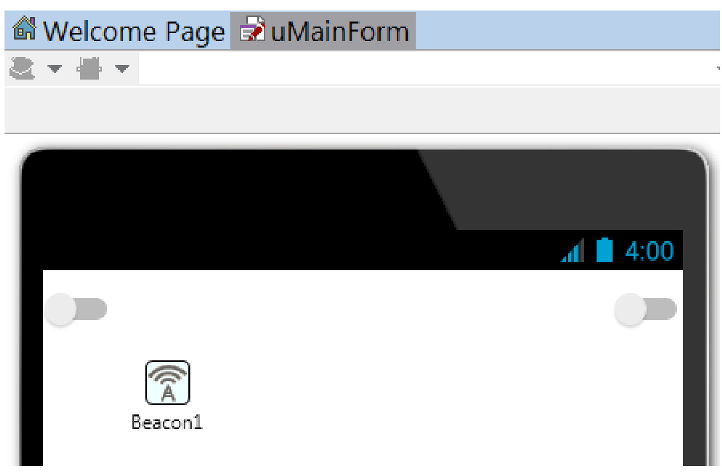
```
  property OnDiscoveryEnd: TDiscoveryLEEndEvent read
FOnDiscoveryLEEnd write FOnDiscoveryLEEnd;
     property OnDiscoverLEDevice: TDiscoverLEDeviceEvent read
FOnDiscoverLEDevice write FOnDiscoverLEDevice;
```

要偵測低功耗藍牙設備，開發人員只需要呼叫 TBluetoothLEManager StartDiscovery 方法即可：

```
  function StartDiscovery(Timeout: Cardinal; const
AFilterUUIDList: TBluetoothUUIDsList = nil): Boolean;
```

因此要偵測 Beacon 設備我們只需要指定一個事件處理函式給 OnDiscoverLEDevice 特性，再呼叫 StartDiscovery 方法。

請在 IDE 中建立一個 Multi-Device 應用程式，在主表單中加入 2 個 TSwitch 元件再加入一個 TBeacon 元件，如下所示：



主表單中左上方的 TSwitch 元件目的是開始自動偵測 Beacon 設備，右上方的 TSwitch 組件則是開始對偵測到的 Beacon 設備主進行掃瞄/監督等工作。

因此在上方的 TSwitch 元件的 OnSwitch 事件中呼叫 DoAutoScanBeacons 方法開始自動偵測 Beacon 設備：

```
procedure TfmMainForm.swtScanBeaconsSwitch(Sender: TObject);
begin
  DoAutoScanBeacons;
end;
```

版權所有 請勿翻印

接著在主表單類別中宣告一個 TBluetoothLEManager 類別物件變數，以便使用它進行自動偵測 Beacon 設備：

```
    FManager: TBluetoothLEManager;
```

DoAutoScanBeacons 使用的方法就是前面我們說的，先使用 TBluetoothLEManager 類別取得一個目前的 TBluetoothLEManager 類別物件並指定給 FManager，再指定事件處理函式 DiscoverLEDevice 給它的 OnDiscoverLeDevice 觸發事件，如此一來當 TBluetoothLEManager 每偵測到一個 Beacon 設備就會呼叫 DiscoverLEDevice，最後呼叫 StartDiscovery 方法使用 10 秒的時間自動偵測 Beacon 設備：

```
procedure TfmMainForm.DoAutoScanBeacons;
begin
  if FManager = nil then
  begin
    FManager := TBluetoothLEManager.Current;
    FManager.OnDiscoverLeDevice := DiscoverLEDevice;
```

```
  end;
  FManager.StartDiscovery(10000);
end;
```

DiscoverLEDevice 方法會在 TBluetoothLEManager 偵測到 Beacon 設備時被呼叫，它首先檢查被偵測到的低功耗藍牙設備是否包含製造商數據（$FF），如果是的話就應該是一個 Beacon 設備（請回頭參考前面的表格說明，iBeacon/AltBeacon 的第 1 個資料一定是 FF），接著呼叫 DecodeScanResponse 把偵測到的 Beacon 設備製造商數據轉回代表 Beacon 設備的 UUID，Major ID，Minor ID 等資料，最後呼叫 AddScanedBeacon 方法把把偵測到的 Beacon 設備加入到 TBeacon 元件中：

```
procedure TfmMainForm.DiscoverLEDevice(const Sender: TObject;
const ADevice: TBluetoothLEDevice;
        Rssi: Integer; const ScanResponse: TScanResponse);
var
  LBeaconDevice: TBeaconDevice;
begin
  if
ScanResponse.ContainsKey(TScanResponseKey.ManufacturerSpecificDa
ta) then
  begin
    LBeaconDevice := DecodeScanResponse;
    AddScanedBeacon(LBeaconDevice);
  end;
end;
```

版權所有 請勿翻印

我們可以在 System.Bluetooth 程式單元中找到 TScanResponseKey.ManufacturerSpecificData 果然是定義為 FF：

```
ManufacturerSpecificData=$FF
```

而 AddScanedBeacon 方法非常重要，因為它把 TBluetoothLEManager 偵測到 Beacon 設備加入到 TBeacon 元件中就可以讓 TBeacon 元件自動說明我們掃瞄/監督 Beacon 設備。AddScanedBeacon 方法先呼叫 BeaconNotAdded 方法看看我們是否已經把偵測到的 Beacon 設備加入到了 TBeacon 元件中，如果沒有的話就建立一個 TBeaconRegionItem 物件，設定好 UUID，Major ID，Minor ID 等資料後就可以加入到 TBeacon 元件中：

```
procedure TfmMainForm.AddScanedBeacon(aBeacon: TBeaconDevice);
```

```
var
  iIndex : Integer;
  aBRI : TBeaconRegionItem;
begin
  if (BeaconNotAdded(aBeacon)) then
  begin
    aBRI := TBeaconRegionItem.Create(Beacon1.MonitorizedRegions);
    aBRI.GUID := aBeacon.GUID;
    aBRI.Major := aBeacon.Major;
    aBRI.Minor := aBeacon.Minor;
    lvBeacons.Items.Add.Text := '發現 Beacon : '  +
aBeacon.GUID.toString;
  end;
```

BeaconNotAdded 方法判斷偵測到的 Beacon 設備是否已存在 TBeacon 元件中，這可以由檢查 TBeacon 元件的 MonitorizedRegions 特性串列物件中包含的 TBeaconRegionItem 物件的 UUID 值來確定：

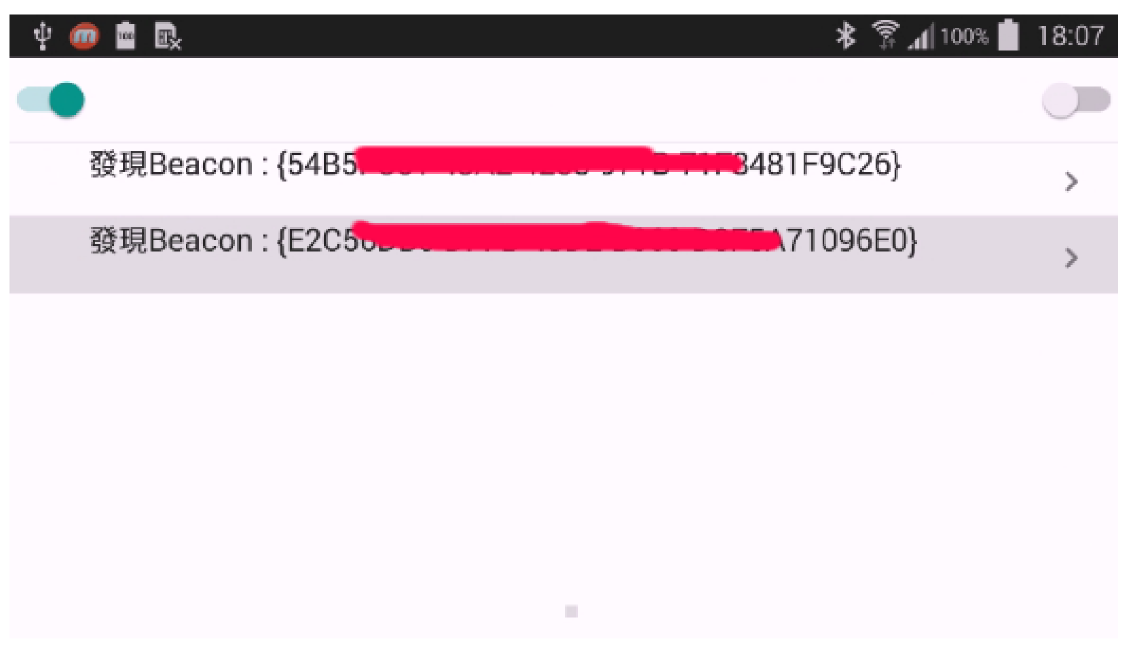
```
function TfmMainForm.BeaconNotAdded(aBeacon: TBeaconDevice):
Boolean;
var
  iIndex : Integer;
  aBRI : TBeaconRegionItem;
begin
  Result := True;
  for iIndex := 0 to Beacon1.MonitorizedRegions.Count - 1 do
  begin
    aBRI := Beacon1.MonitorizedRegions.Items[iIndex];
    if (aBRI.UUID = aBeacon.GUID.ToString)  then
    begin
      Result := False;
      Break;
    end;
  end;
end;
```

版權所有 請勿翻印

最後我們可以設定主表單中 TBeacon 元件的 On EnterRegion 和 OnExitRegion 事件來監督 Beacon 設備(請參考下一小節的說明)：

```
procedure TfmMainForm.Beacon1EnterRegion(const Sender: TObject;
  const UUID: TGUID; AMajor, AMinor: Integer);
var
  LItem: TListViewItem;
begin
  LItem := lvBeacons.Items.Add;
  LItem.Text := UUID.ToString;
  LItem.Detail := '進入 : ' + 'Major: ' + AMajor.ToString + ' Minor:
' + AMinor.ToString  + ' 時間 :' + TimeToStr(now) ;
End;
```

現在就可以執行此範例 App，下面的圖形顯示了筆者附近有 2 個 Beacon 設備：



如果再開啟主表單右上方的 TSwitch 元件，就可以看到類似下面的畫面，可以掃瞄/監督 App 距離 Beacon 設備的距離範例等資訊了：

瞭解了如何自動偵測 Beacon 設備後，我們就可以進一步說明如何處理 App 和 Beacon 設備的互動了。

16-7-2 開發已知 Beacon 設備 App

版權所有 請勿翻印

對於企業應用而言，在布建物聯網架構時使用的 Beacon 設備的 UUID 和 MajorID，MinorID 應該都已經確定了，因此不需要像上一小節要處理自動偵測 Beacon 設備的工作。

為了方便開發人員完成此開發的工作，RIO 在 System.Beacon 程式單元中定義了 TBeaconManager 類別可使用進行 Beacon 的開發工作。下面說明了如何使用 TBeaconManager 類別物件：

```
001    BeaconManager :=
TBeaconManager.GetBeaconManager(TBeaconScanMode.Standard); // 或
TBeaconScanMode.Alternative
002    try
003      //指定 Beacon Manager 事件處理函式
004      BeaconManager.OnBeaconEnter := BeaconEnter;
005      BeaconManager.OnBeaconExit := BeaconExit;
006      BeaconManager.OnEnterRegion := EnterRegion;
007      BeaconManager.OnExitRegion := ExitRegion;
008      BeaconManager.OnBeaconProximity := BeaconProximity;
009
010      //註冊要監督的 Beacon 區域
```

```
011     BeaconManager.RegisterBeacon(TGUID.Create('
{D1E10B90-38A6-4BEB-99BB-DF5A6F51F7AE}'));
012     BeaconManager.RegisterBeacon(TGUID.Create('
{F490070D-7341-40EB-B28C-43CBDBFBAA17} '));
013
014     //開始掃瞄
015     BeaconManager.StartScan;
016
017     //在此時就會觸發前面設定的事件
018
019     //停止掃瞄
020     BeaconManager.StopScan;
021
022   finally
023   //釋放 TBeaconManager 對象
024     BeaconManager.Free;
025   end;
```

從上面的程式碼中我們可以知道，要使用 TBeaconManager 類別物件，開發人員要使用下面的步驟：

版權所有 請勿翻印

1. 根據是要掃瞄 iBeacon 或是 AltBeacon，取得相對的 BeaconManager 物件(001 行)
2. 設定不同 Beacon 設備的件處理函式(004~008 行)
3. 設定要掃瞄的 Beacon 區域(Beacon 設備的 UUID 值)(011~012 行)
4. 開始掃瞄 Beacon 區域中的 Beacon 設備(014 行)
5. 處理完成之後停止掃瞄(020 行)
6. 最後釋放 TBeaconManager 物件(024 行)

為了進一步簡化開發人員的工作，RIO 封裝了 TBeacon 元件,開發人員只需要使用 TBeacon 元件就可以更簡單的完成上面的工作。TBeacon 元件提供了下面的特性設定：

特性	說明
Mode	設定使用 iBeacon 或是 AltBeacon 設備。設定 Standard 代表 iBeacon，設定 Alternative 代表 AltBeacon 設備

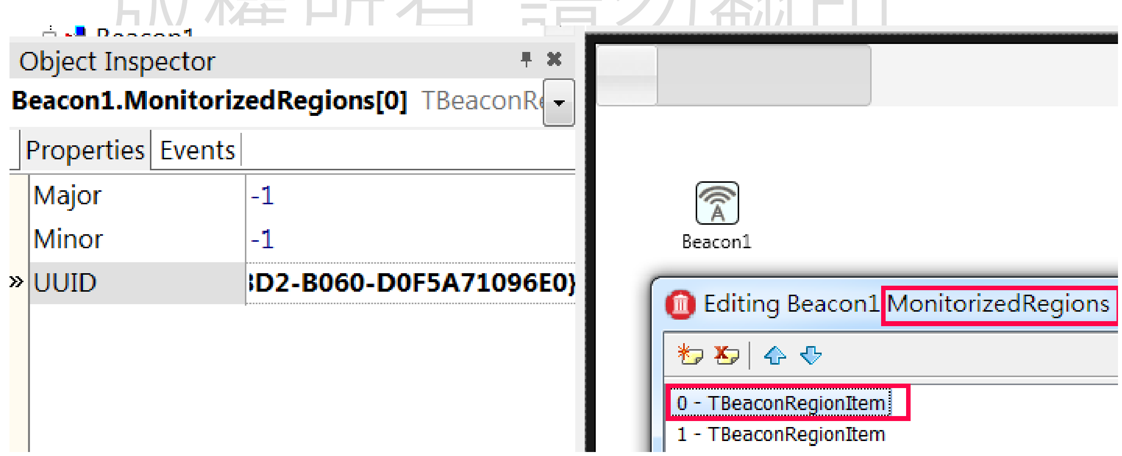
MonitorizedRegions	設定設定要掃瞄的 Beacon 區域，即 Beacon 設備的 UUID 值

TBeacon 元件也提供了下面的事件處理函式：

事件處理函式	說明
OnBeaconEnter	在進入要掃瞄的 Beacon 設備範圍時觸發
OnBeaconExit	在離開要掃瞄的 Beacon 設備範圍時觸發
OnBeaconProximity	在和 Beacon 設備的距離改變時觸發
OnEnterRegion	在進入要掃瞄的 Beacon 區域範圍時觸發
OnExitRegion	在離開要掃瞄的 Beacon 區域範圍時觸發
OnCalcDistance	在要計算和 eacon 設備的距離時觸發
OnParseManufacturerData	在解析 Beacon 設備的製造商數據時觸發

現在就可以說明如何使用 TBeacon 元件來開發已知 Beacon 設備 App 了，請在 IDE 中建立一個 Multi-Device 應用程式，在主表單中加入 TToolBar，TTabControl，TSwitch，TRectangle 和 TBeacon 元件，並在物件檢視器中點選 TBeacon 元件的 MonitorizedRegions 特性，在其中加入 2 個 TBeaconRegionItem 物件，在每一個 TBeaconRegionItem 物件的 UUID 特性中輸入你要監督掃瞄的 Beacon 設備的 UUID 值，如下所示：

版權所有 請勿翻印



在上面的 TBeaconRegionItem 物件中其 Major 和 Minor 特性值都是-1，-1 代表要監督掃瞄具有相同 UUID 值的 Beacon 設備，不管其 Major 和 Minor 特性值是什麼。

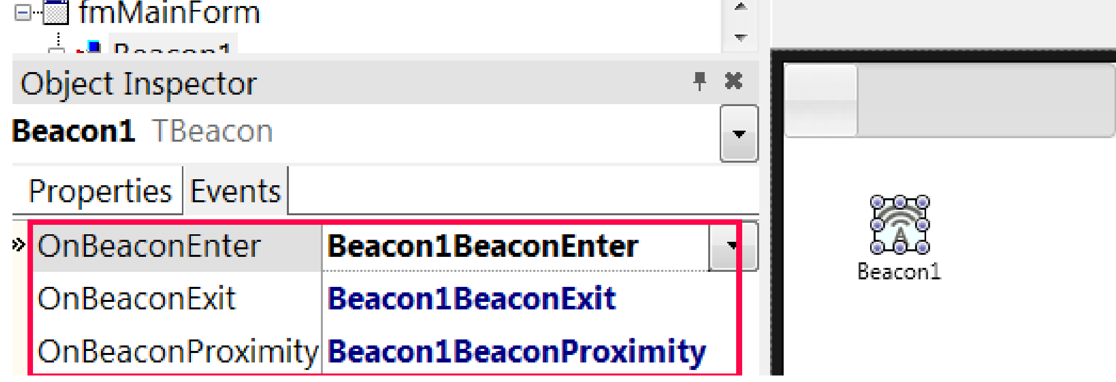
先在主表單中 TSwitch 元件的 OnSwitch 事件處理函式根據 TSwitch 元件的開啟狀況開始掃瞄或是停止掃瞄：

```
procedure TfmMainForm.StartScanBeacons;
begin
```

```
  Beacon1.StartScan;
end;

procedure TfmMainForm.StopScanBeacons;
begin
  Beacon1.StopScan;
end;

procedure TfmMainForm.Switch1Switch(Sender: TObject);
begin
  if (Switch1.IsChecked) then
    StartScanBeacons
  else
    StopScanBeacons;
end;
```

要開始掃瞄或是停止掃瞄只需要呼叫 **TBeacon** 元件的 **StartScan** 和 **StopScan** 方法即可。

接著為主表單中的 **TBeacon** 元件設定如下的事件處理函式：



在的 **TBeacon** 元件的 **On BeaconEnter** 和 **On BeaconExit** 事件中我們只是顯示一訊息代表進入或是離開 **Beacon** 設備的有效範圍：

```
procedure TfmMainForm.Beacon1BeaconEnter(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  DisplayBeaconInfo(ABeacon, '進入 : ');
end;
```

```
procedure TfmMainForm.Beacon1BeaconExit(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  DisplayBeaconInfo(ABeacon, '離開 : ');
end;

procedure TfmMainForm.DisplayBeaconInfo(const ABeacon: IBeacon;
const sStatus : String);
var
  alvi : TListViewItem;
begin
  TMonitor.Enter(FLock);
  try
    alvi := Self.lvBeacons.Items.Add;
    alvi.Text := sStatus + ABeacon.GUID.ToString;
    alvi.Detail := 'Major: ' + ABeacon.Major.ToString + ' Minor: '
+ ABeacon.Minor.ToString  + ' time :' + TimeToStr(now);
  finally
    TMonitor.Exit(FLock);
  end;
end;
```

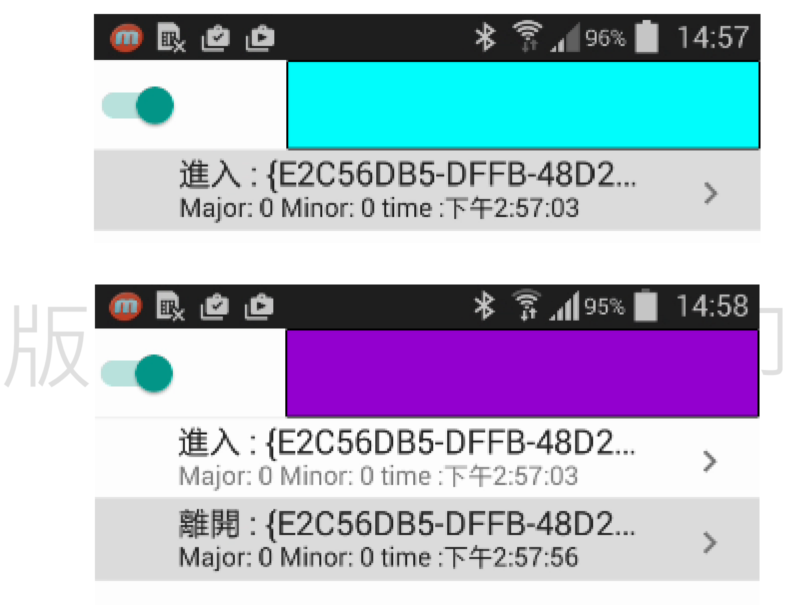
版權所有 請勿翻印

最後在 **TBeacon** 元件的 **OnBeaconProximity** 事件中我們根據 **Beacon** 設備的距離來改變主表單中 **TRectangle** 元件的顏色：

```
procedure TfmMainForm.Beacon1BeaconProximity(const Sender:
TObject;
  const ABeacon: IBeacon; Proximity: TBeaconProximity);
begin
  TMonitor.Enter(FLock);
  try
    case ABeacon.Proximity of
      TBeaconProximity.Immediate : Self.Fill.Color :=
TAlphaColorRec.Springgreen;
      TBeaconProximity.Near : Self.Fill.Color :=
TAlphaColorRec.Aqua;
      TBeaconProximity.Far : Self.Fill.Color :=
TAlphaColorRec.Slateblue;
```

```
      TBeaconProximity.Away : Self.Fill.Color :=
TAlphaColorRec.Darkviolet;
    end;
  finally
    TMonitor.Exit(FLock);
  end;
end;
```

請編譯並執行此範例 App 並搭配使用的 Beacon 設備就可以看到類似如下的執行畫面：



TBeacon 元件果然又簡單又實用，可快速幫助開發人員開發物聯網的相關 App。

17 開發有趣的物聯網應用架構

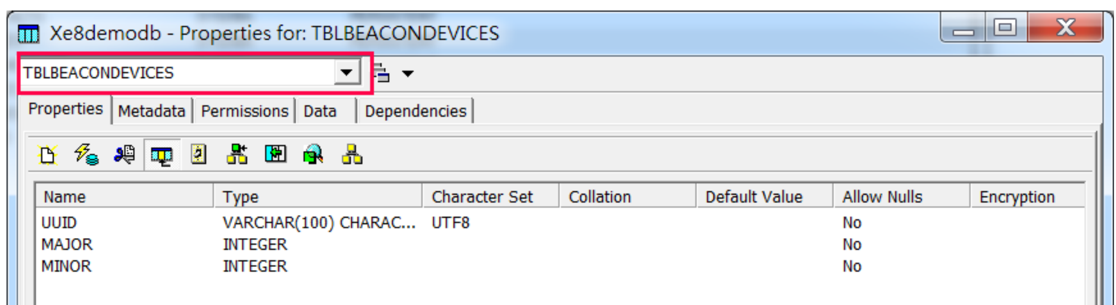
在前面的下節中已經說明了如何藉由 TBeacon 元件開發物聯網相關的 App，但只是偵測和掃瞄/監督 Beacon 設備並沒有什麼實際的做用。開發人員應該再結合其他的技術來提供有意義的服務，例如在本書前面說明的推播技術，開發人員可以結合 Beacon 設備和推播技術以及後端的服務伺服器來提供類似下圖的架構：

當前端App偵測和掃瞄/監督Beacon設備之後可以呼叫後端的DataSnap服務來擷取更完整的資料或是呼叫後端的雲端服務，而遠端的 Windows 用戶端也可以藉由推播技術把需要的資訊主動推播到前端的 App 中，讀者可以回頭參考本書前面討論的內容來開發更有趣的物聯網應用 App，下面簡單說明如何融合前面章節討論的推播技術，DataSnap 技術和 Beacon 技術來開發一有趣又實用的物聯網範例架構。

版權所有 請勿翻印

17-1 範例資料表

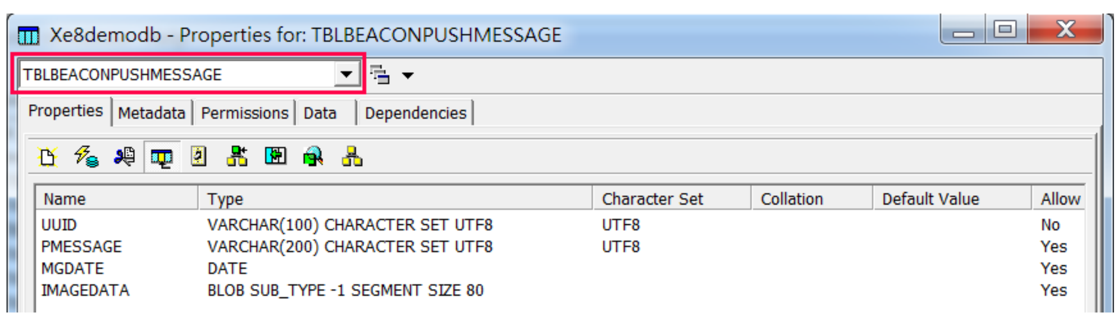
首先在 RIODEMODB.GDB 中讓我們加入 2 個資料表：TBLBEACONDEVICES 和 TBLBEACONPUSHMESSAGE：


Xe8demodb - Properties for: TBLBEACONDEVICES

TBLBEACONDEVICES

Properties | Metadata | Permissions | Data | Dependencies

Name	Type	Character Set	Collation	Default Value	Allow Nulls	Encryption
UUID	VARCHAR(100) CHARAC...	UTF8			No	
MAJOR	INTEGER				No	
MINOR	INTEGER				No	

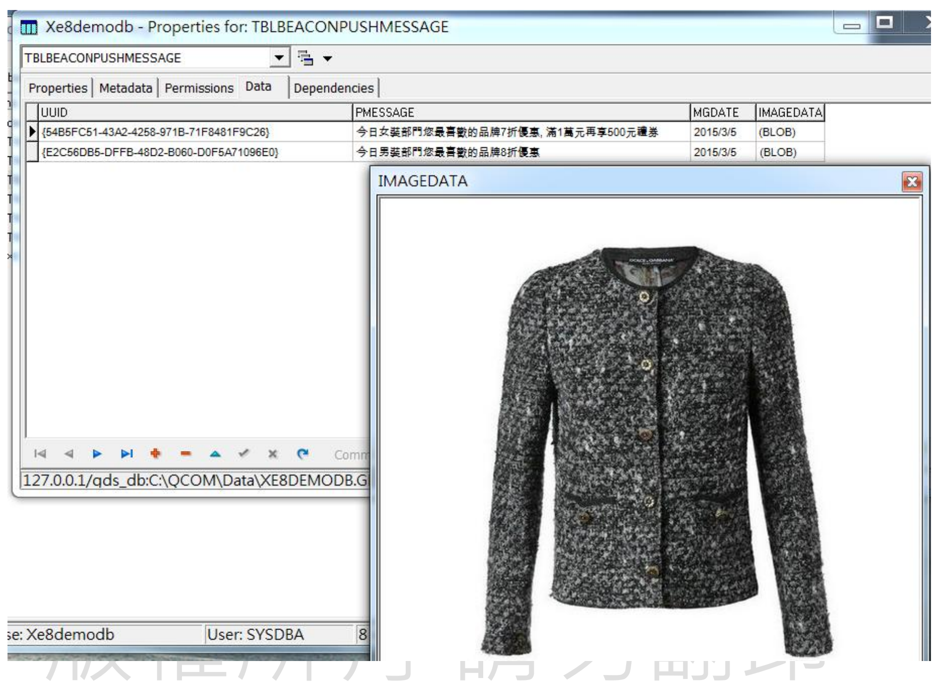

Xe8demodb - Properties for: TBLBEACONPUSHMESSAGE

TBLBEACONPUSHMESSAGE

Properties | Metadata | Permissions | Data | Dependencies

Name	Type	Character Set	Collation	Default Value	Allow
UUID	VARCHAR(100) CHARACTER SET UTF8	UTF8			No
PMESSAGE	VARCHAR(200) CHARACTER SET UTF8	UTF8			Yes
MGDATE	DATE				Yes
IMAGEDATA	BLOB SUB_TYPE -1 SEGMENT SIZE 80				Yes

其中的 TBLBEACONDEVICES 資料表是使用來註冊用戶端 App 偵測到的 Beacon 設備而 TBLBEACONPUSHMESSAGE 資料表則儲存了要推播到特定 Beacon 設備附近 App 的推播訊息。如下所示在範例 TBLBEACONPUSHMESSAGE 資料表中尚包含了圖形的資料在使用者收到推播訊息之後還可以進一步看到詳細的資訊：



17-2 仲介 DataSnap 伺服器

接著在前面章節討論 GCM 時的範例 DataSnap 伺服器中再加入如下的 2 個服務方法：

```
//For Beacon Services
function RegisterBeacon(const sID, sUUID : String; const iMajorID,
iMinorID : Integer) : Boolean;
function GetBeaconMessage(const sMessage : String) : TDataSet;
```

RegisterBeacon 方法是在用戶端偵測到 Beacon 設備時向 DataSnap 伺服器使用的，而 GetBeaconMessage 方法則可以根據特定的 Beacon 設備取得包含推播訊息的 TDataSet 物件。

RegisterBeacon 方法的實作如下，它很簡單只是藉由範例程式中的資料模組把偵測到的 Beacon 設備寫入 TBLBEACONDEVICES 資料表中：

```
function TServerMethods1.RegisterBeacon(const sID, sUUID: String;
const iMajorID, iMinorID : Integer): Boolean;
```

```
begin
  TMonitor.Enter(dmPushDemo);
  try
    Result := dmPushDemo.RegisterBeacon(sID, sUUID, iMajorID,
iMinorID);
  finally
    TMonitor.Exit(dmPushDemo);
  end;
  PushBeaconMessage(sID, sUUID);
end;
```

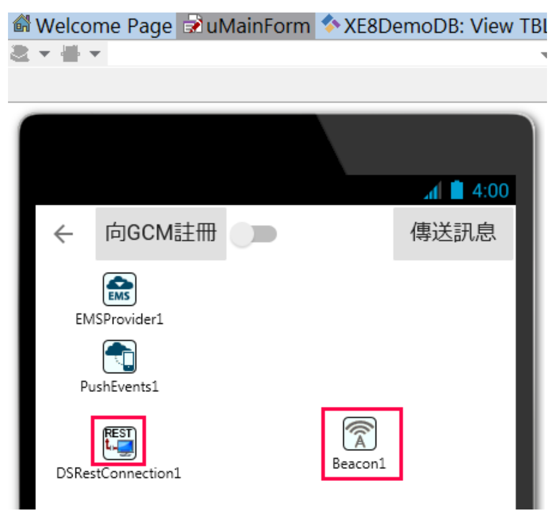
一旦用戶端 App 偵測到特定的 Beacon 設備並呼叫 RegisterBeacon 方法註冊 Beacon 設備之後就會繼續呼叫 PushBeaconMessage 方法，而 PushBeaconMessage 方法則是利用前面在推播章節中已經實作的 PushMessageTo 推播方法把 TBLBEACONPUSHMESSAGE 資料表中屬於偵測到特定的 Beacon 設備推播訊息推播到用戶端的 App 中：

```
function TServerMethods1.PushBeaconMessage(const sDID, sUUID :
String): Boolean;
var
  aDataSet : TFDQuery;
begin
  Result := True;
  TMonitor.Enter(dmPushDemo);
  try
    aDataSet := dmPushDemo.GetBeaconMessage(sUUID);
    while (not aDataSet.Eof) do
    begin
      Result := PushMessageTo(sDID,
aDataSet.FieldByName('PMESSAGE').AsString) and Result;
      aDataSet.Next;
    end;
  finally
    aDataSet.Active := False;
    TMonitor.Exit(dmPushDemo);
  end;
end;
```

版權所有 請勿翻印

17-3 移動用戶端

最後讓我們在用戶端 App 中使用 TBeacon 元件偵測 Beacon 設備，使用 TDSRestConnection 元件連結範例 DataSnap 伺服器再使用 TEMSProvider 和 TPushEvents 元件啟動推播功能：



在使用者開啟主表單中的 TSwitch 元件後就啟動 TBeacon 元件開始偵測 Beacon 設備：

版權所有 請勿翻印

```
  if (swActivateBeacon.IsChecked) then
    Beacon1.Enabled := True
  else
    Beacon1.Enabled := False;
```

在 TBeacon 元件偵測到 Beacon 設備之後就呼叫範例 DataSnap 伺服器的 RegisterBeaconToServer 方法註冊 Beacon 設備並開始接收屬於此特定 Beacon 設備的推播訊息：

```
procedure TfmMainForm.Beacon1BeaconEnter(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  ListView1.Items.Add.Text := '發現 Beacon : ' +
ABeacon.GUID.ToString;
  RegisterBeaconToServer(ABeacon);
  TabControl1.TabIndex := 1;
end;
```

RegisterBeaconToServer 方法藉由 TDSRestConnection 元件自動產生的用戶端連結程式碼以連結範例 DataSnap 伺服器：

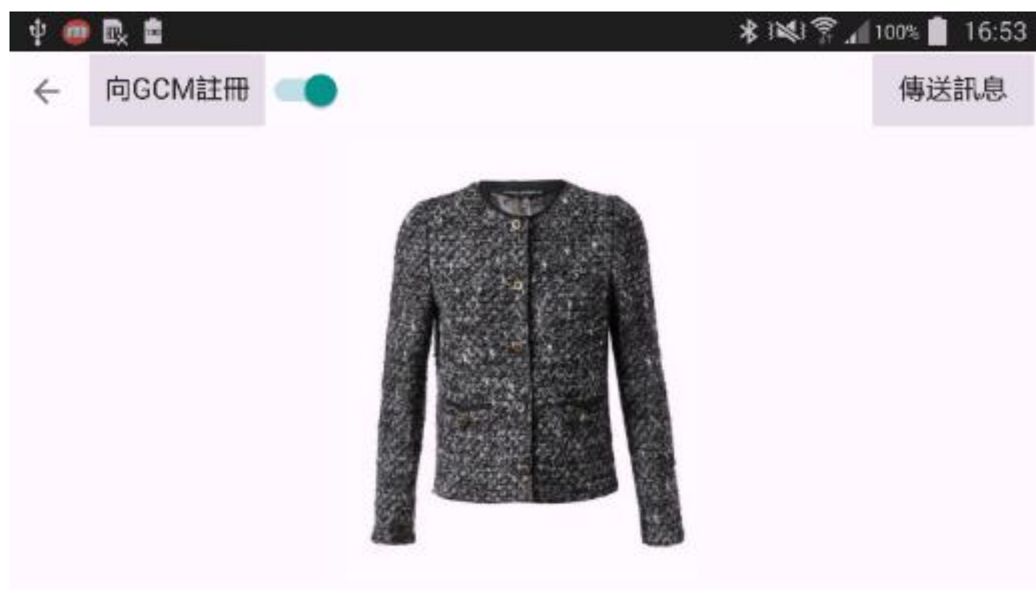
```
procedure TfmMainForm.RegisterBeaconToServer(const ABeacon:
IBeacon);
var
  aServer: TServerMethods1Client;
begin
  aServer :=
TServerMethods1Client.Create(Self.DSRestConnection1);
  try
    if (aServer.RegisterBeacon(PushEvents1.DeviceToken,
ABeacon.GUID.ToString, ABeacon.Major, ABeacon.Minor)) then
      ListView1.Items.Add.Text := 'Beacon 註冊成功';
  finally
    aServer.Free;
  end;
end;
```

版權所有 請勿翻印

17-4 執行 DataSnap 伺服器和用戶端 App

編譯並執行 DataSnap 伺服器和用戶端 App，就可在用戶端 App 看到如下的執行結果，用戶端 App 能夠自動偵測 Beacon 設備，自動接收推播訊息，並且可在用戶端 App 中查詢到圖形的推播資料了：

16:33
向GCM註冊
傳送訊息
今日男裝部門您最喜歡的品牌8折優惠 -傳遞時間 : 15-03-25 16:33:33
今日女裝部門您最喜歡的品牌7折優惠, 滿1萬元再享500元禮券 -傳遞時間...



16:53
向GCM註冊
傳送訊息

18 開發 Android Service App

從 Delphi 支援移動開發功能之後就有許多開發人員想要知道如何能夠使用 Delphi 來開發 Android 服務類型的 App。到了 RIO Delphi 終於可以支援開發服務類型的 App，在本小節中將詳細說明什麼是服務類型的 App 以及如何使用 Delphi 來開發。

依據 Android 開發人員手冊的說明，所謂的服務是指：

1. 一種機制允許 App 中部份程式碼可在背景執行，Android App 可藉由呼叫 StartService()方法要求系統啟動服務，服務會在背景執行直到有人停止它或是服務自己呼叫 StopService()方法停止執行。

2. 一種機制允許 App 把功能輸出可讓其他 App 呼叫使用，其他 App 可藉由呼叫 bindService()方法來連結服務。

從上面的說明我們可以瞭解基本上如果在 App 中有部份程式碼的功能是我們想持續的執行，即使是 App 本身已經結束時仍然可繼續執行，那麼就可以把這些功能撰寫成服務，而且成為服務的功能甚至可以讓其他 App 呼叫使用。

版權所有 請勿翻印

服務的另一個用處就是如果 Android 的 App 被推送到背景的話那麼這個 App 就有可能被系統結束(例如記憶體不足或是系統資源不足時)，如果 App 不希望被系統選擇結束，那麼 App 可以啟動一個服務，如此一來就可以減少被結束的機會。

服務有許多的實用場合，例如如果我們想持續記錄目前的 GPS 位置，那就可以把這個功能封裝成服務。又例如在物聯網應用中 App 需要持續的掃瞄低功率藍牙設備或是 Beacon，那麼服務就是很適合使用的機制。

18-1 Android 服務種類

Android 服務種類基本上可分成：

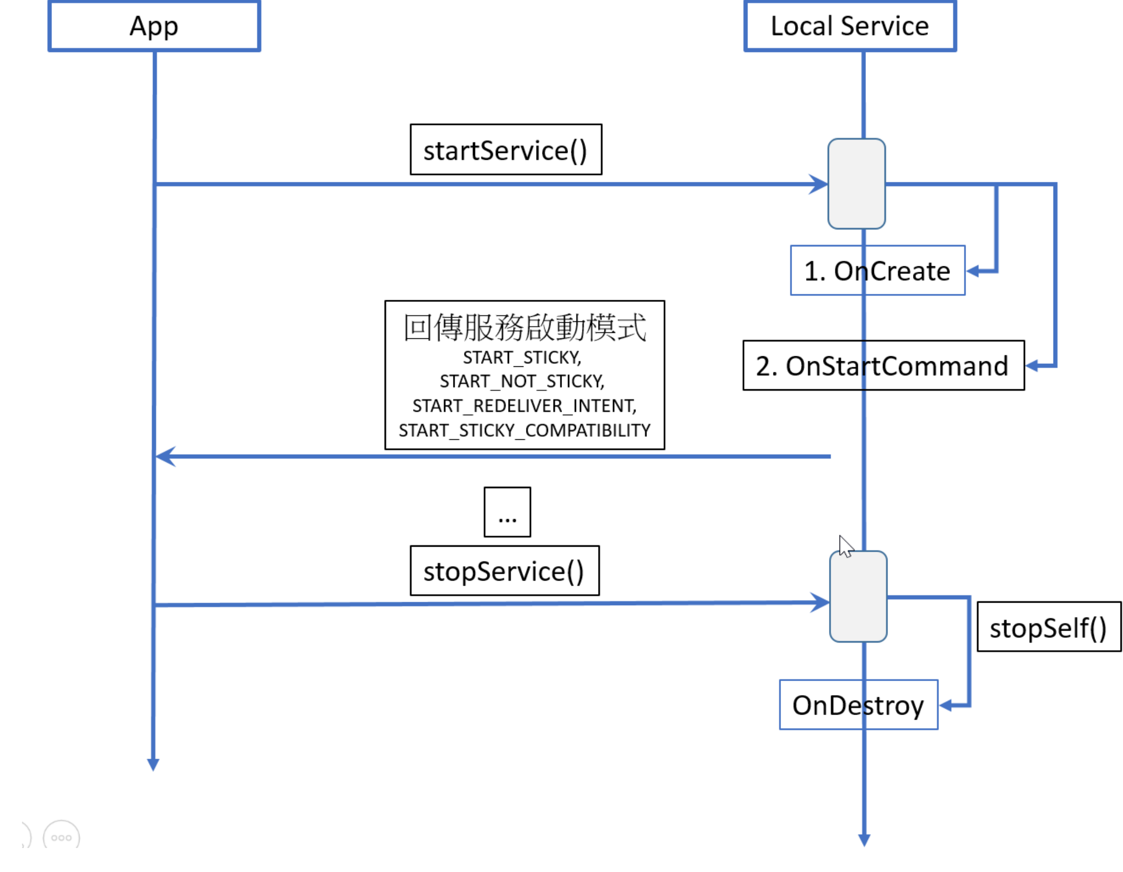
- 本機服務(Local Service)：是指由 App 在程式中啟動執行的服務，只供本身 App 使用
- 遠端服務(Remote Service)：是指由 App 在程式中啟動執行的服務，但除了供本身 App 使用化外，也可以讓其他 App 連結使用

如前所述，由於服務是在背景執行，因為當 App 啟動了服務之後前景的 App 可能會繼續執行，也可能會終止執行。那麼前景的 App 執行行為會對背景服務產生什麼影響？當前景 App 結束時由它啟動的背景服務會不會也結束執行呢？

要回答這些問題我們就需要瞭解背景服務的生命週期，它的啟動方式以及前景 App 的生命週期。

18-2 服務生命週期

要瞭解前景 App 和背景服務的關係可以從前景 App 如何啟動背景服務來說明。當前景 App 要啟動背景服務時，可藉由呼叫 startService()方法，此時 Android 系統就會建立背景服務並呼叫背景服務的 onCreate()事件，接著再呼叫 onStartCommand()事件，onStartCommand()事件執行完畢之後會回傳背景服務啟動模式，這個回傳的背景服務會決定背景服務的生命週期，稍後我們會說明。如果要停止背景服務的話 App 可以呼叫 stopService()方法，或是背景服務也可以呼叫 stopService()方法停止自己，而在呼叫 stopService()方法或是 stopService()方法後 Android 系統會呼叫背景服務的 onDestroy()事件。這整個流程可由下圖說明：



背景服務本身可由 onStartCommand()事件回傳的數值來告訴 App 它的生命週期型態，下面的表格說明了背景服務可回傳的的數值以及它們代表的意義：

服務回傳執行模式	說明
START_STICKY	如果啟動服務的 App 被結束，而背景服務已經被

	啟動過，那背景服務會保持在已被啟動的狀況，下次 App 再次試著啟動背景服務時，會再次呼叫背景服務的 onStartCommand()方法
START_NOT_STICKY	如果啟動服務的 App 被結束，而背景服務已經被啟動過，那背景服務會脫離啟動的狀況，下次 App 需要再次重新開機背景服務
START_REDELIVER_INTENT	如果啟動服務的 App 被結束，而背景服務已經被啟動過，那背景服務會保持在已被啟動的狀況，下次 App 再次試著啟動背景服務時，會所新啟動背景服務並再次使用上次呼叫 onStartCommand()方法的參數再次呼叫背景服務的 onStartCommand()方法
START_STICKY_COMPATIBILITY	和使用 START_STICKY 模式很類似，只是再次啟動背景服務時並不保證一定會再次呼叫背景服務的 onStartCommand()方法

從上面的說明可以瞭程式師可藉由在 onStartCommand()方法回傳的數值來定義背景服務的執行模式，在稍後的範例說明中就可以看到 Delphi 的服務功能可以讓我們進行這些設定。

版權所有 請勿翻印

18-3 App 生命週期

由於服務是由一個 App 啟動的，因此 App 和背景服務之間就會有一定的互動及影響，因此如果 Android 系統目前已經沒有資源，例如記憶體太少，時就會開始試著結束一些 App，那麼 Android 系統會如何決定結束那些 App？如果被選定的 App 有啟動背景服務時會如何？

在 Android 開發手冊中清楚的說明了這 2 者之間生命週期的影響如下：

1. 如果 App 正在啟動背景服務，而且背景服務正在執行 onCreate(),，onStartCommand()，或 onDestroy()方法，那麼 Android 系統便會把 App 視為目前正在最前端執行的 App 而不會結束它。

2. 如果背景服務已啟動，那麼此 App 便會被視為比目前可視的 App 不重要，但比不可視的 App 重要。

3. 但如果有其他 App 連結到背景服務(Android 的遠端服務)，那麼擁有背景服務的 App 和目前正在最前端執行的 App 一樣重要。

18-4 服務類別

Android 系統的服務功能是由 2 個類別提供的，它們是：

- android.app.Service
- android.app.IntentService

IntentService 類別是從 Service 類別繼承下來的，IntentService 類別主要是提供了一個 onHandleIntent(Intent intent)抽象方法，這個方法可以自動產生一個工作執行緒執行並接收一個由 App 傳遞來的 Intent 參數。

由於 Android 的本機服務或是遠端服務都可以由這 2 個類別實作，因此在 Android 系統中一共有 4 種不同的服務。在 Delphi 中是由 TAndroidBaseService 類別提供 android.app.Service 和 android.app.IntentService 的實作，而且 TAndroidBaseService 類別是由 TDataModule 類別繼承下來，Delphi 如此設計的原因是為了讓 Delphi 程式師可以在 TAndroidBaseService 中使用 Delphi 的原件在背景服務中提供功能。

TAndroidBaseService 是提供 Android 服務的母類別，而真正提供 android.app.Service 實作功能的則是 TAndroidService 類別，提供 android.app.IntentService 實作功能的則是 TAndroidIntentService 類別。

版權所有 請勿翻印

下面是 TAndroidService 類別的基本宣告，它提供了 android.app.Service 的功能：

```
  TAndroidService = class(TAndroidBaseService)
  published
    property OnCreate;
    property OnDestroy;
    property OnStartCommand: TOnStartCommandEvent read
FOnStartCommand write FOnStartCommand;
    property OnBind;
    property OnUnBind;
    property OnRebind;
    property OnTaskRemoved;
    property OnConfigurationChanged;
    property OnLowMemory;
    property OnTrimMemory;
    property OnHandleMessage;
```

```
end;
```

下面的表格說明了 TAndroidService 類別事件的意義：

事件	說明
OnCreate	當背景服務被建立時呼叫
OnDestroy	當背景服務被結束時呼叫
OnStartCommand	當每次 App 使用 startService()方法啟動背景服務時呼叫
OnBind	當 App 使用 bindService()方法系結服務時呼叫
OnUnBind	當 App 使用 unBindService()方法系結服務時呼叫
OnTaskRemoved	在背景服務執行時如果使用者移除 App 工作時呼叫
OnConfigurationChanged	當設備組態改變時被呼叫
OnLowMemory	當系統記憶體不夠時呼叫，要求減少使用的記憶體
OnTrimMemory	當系統決定減少 App 的記憶體時呼叫
OnHandleMessage	當工作執行緒被要求執行工作時呼叫

有了 Android 服務功能的基本瞭解後就可以準備使用 Delphi 來開發您的 Android 背景服務了，讓我們先簡單的說明使用 Delphi 開發 Android 背景服務的基本步驟。

版權所有 請勿翻印

18-5 開發服務流程

前面已經說明了在 Delphi 中是由 TAndroidService 和 TAndroidIntentService 類別提供開發 Android 背景服務的功能，又由於 Android 背景服務分成本機服務和遠端服務，因此在下面將分別說明這 2 種背景服務的開發流程。

本機服務開發流程

要開發本機服務，Delphi 程式師需要使用下面的流程：

1. 開發包含服務程式碼的 Android 服務專案
2. 開發包含 Android 服務的 App
3. 把步驟 1 開發的 Android 服務加入到 App 中
4. App 使用 Android 本機服務

在稍後的章節中會說明如何使用 Delphi 來實作上述的步驟。

遠端服務開發流程

要開發遠端服務，Delphi 程式師需要使用下面的流程：

1. 開發包含服務程式碼的 Android 服務專案
2. 開發包含 Android 服務的 App
3. 把步驟 1 開發的 Android 服務加入到 App 中
4. App 使用 Android 本機服務
5. 再開發另一 App，使用 bindService()方法系結步驟 4 中的 Android 服務，對於此新的 App 而言步驟 4 中的 Android 服務就是遠端服務

18-6 使用 Delphi 開發 Android 服務

為了支援開發 Android 服務 Delphi 除了新增了 TAndroidBaseService，TAndroidService 和 TAndroidIntentService 類別之外也提了 TLocalServiceConnection 和 TRemoteServiceConnection 這 2 個輔助類別來幫助開發人員更容易的連結 Android 本機服務和 Android 遠端服務。TLocalServiceConnection 和 TRemoteServiceConnection 可以簡化開發人員需要呼叫低階 API 的工作，在下面說明開發 Android 服務的章節中將使用這 2 個輔助類別。

版權所有 請勿翻印

支援開發 Android 服務的 Delphi 類別和元件

要連結 Android 本機服務，開發人員可以使用下面的 TLocalServiceConnection 類別，例如在前面的內容中已經說明過可以呼叫 android.app.Service 類別中的 startService()方法來連結 Android 本機服務，因此在 TLocalServiceConnection 類別中就已經定義了一個類別方法 StartService()可提供連結 Android 本機服務，程式師只需要把要連結的 Android 本機服務的名稱傳遞給 TLocalServiceConnection.StartService()就可以了，一旦連結成功就會觸發 TLocalServiceConnection 類別的 OnConnected 處理函式，如果結束連結就會觸發 OnDisconnected 處理函式。

```
TLocalServiceConnection = class
  ...
public
  constructor Create;
  destructor Destroy; override;
```

```
    class procedure StartService(const AServiceName: string);
static;
    procedure BindService(const AServiceName: string; flags: Integer
= 1{TJContext.JavaClass.BIND_AUTO_CREATE});
    procedure UnbindService;
    property OnConnected: TOnLocalServiceConnected read
FOnConnected write FOnConnected;
    property OnDisconnected: TOnLocalServiceDisconnected read
FOnDisconnected write FOnDisconnected;
    property LocalService: TAndroidBaseService read FLocalService;
  end;
```

要連結 Android 遠端服務 Delphi 程式師可以使用 TRemoteServiceConnection 類別，在 TRemoteServiceConnection 中提供了 BindService()方法連結 Android 遠端服務。當 Android 遠端服務執行並且要把執行結果或是訊息傳遞給連結它的遠端 App 時，App 可以藉由 TRemoteServiceConnection 的 OnHandleMessage 事件處理函式來取得。

版權所有 請勿翻印

```
  TRemoteServiceConnection = class
    ...
  public
    constructor Create;
    destructor Destroy; override;
    /// <summary>Bind to service</summary>
    procedure BindService(const APackageName, AServiceName: string;
flags: Integer = 1{TJContext.JavaClass.BIND_AUTO_CREATE});
    /// <summary>Unbind to service</summary>
    procedure UnbindService;
    /// <summary>Event fired when you are connected</summary>
    property OnConnected: TOnRemoteServiceConnected read
FOnConnected write FOnConnected;
    /// <summary>Event fired when you are disconnected</summary>
    property OnDisconnected: TOnRemoteServiceDisconnected read
FOnDisconnected write FOnDisconnected;
    /// <summary>Event fired when a message arrive to the local
messenger</summary>
    property OnHandleMessage: TOnRemoteServiceHandleMessage read
FOnHandleMessage write FOnHandleMessage;
```

```
    /// <summary>Property to acces to the service messenger</summary>
    property ServiceMessenger: JMessenger read FJServiceMessenger;
    /// <summary>Property to acces to the local messenger</summary>
    property LocalMessenger: JMessenger read FJLocalMessenger;
    /// <summary>Property to acces to the Handler of the
listener</summary>
    property Handler: JRTLHandler read GetHandler;
  end;
```

在下面的小節中將詳細說明如使用這 2 個輔助類別。

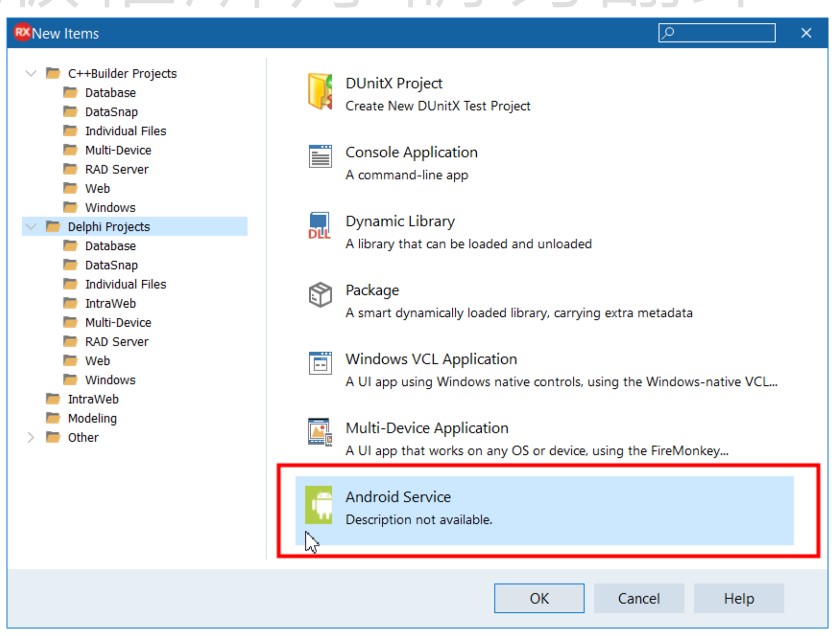
本機服務 App

現在就讓我們先開發一個簡單的 Android 本機服務並且讓它在背景執行，把它推到背景並執行其他 App，之後再次啟動 App 並存取背景服務看看在 App 結束後它是否是在持續的執行中。

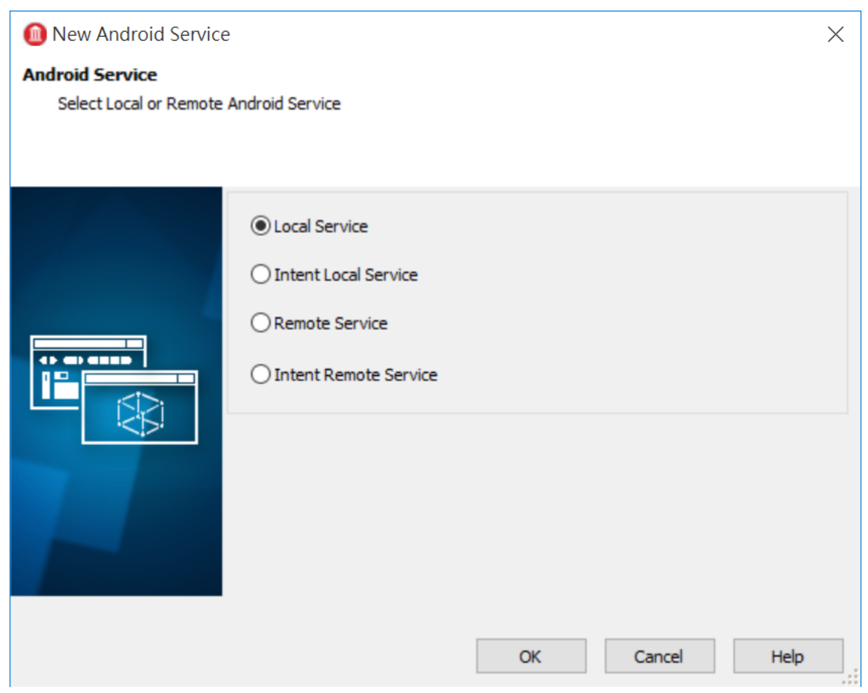
步驟 1： 開發包含服務程式碼的 Android 服務專案

第 1 步是先建立 Android 服務專案，請在 IDE 中點選 New Items 工具按鈕，在 New Items 對話盒中選擇”Android Service”圖像：

版權所有 請勿翻印

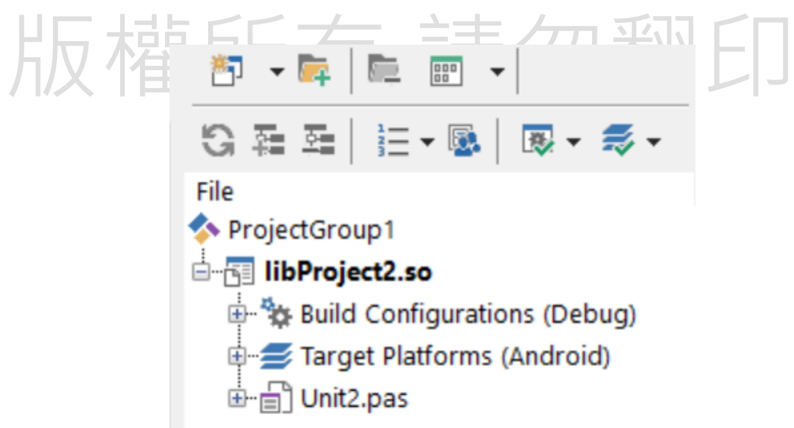


點選 Android Service”圖像後 IDE 會顯示下面的對話盒詢問開發人員要建立的 Android 服務種類，在本小節中讓我們先建立最簡單的 Android 服務種類：實作 android.app.Service 的本機服務，因此請在下面的對話盒中選擇 Local Service 選項並點選 OK：



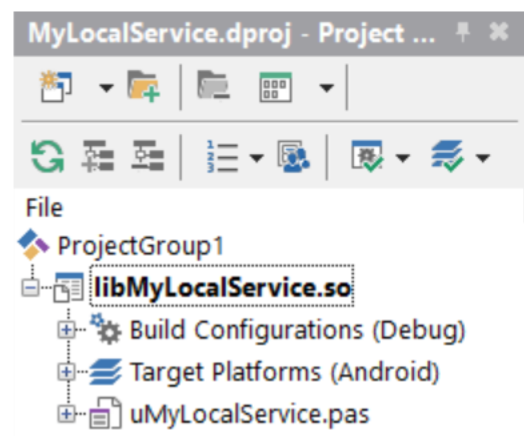
接著 IDE 便會建立一個 Android 服務如下所示，請注意此項目的結尾名稱是.so，它代表 Android 系統中的 share object library。

版權所有 請勿翻印



在專案管理員中請點選滑鼠右鍵選擇 Rename 選項把此專案重新命名為 MyLocalService 如下所示(當然您也可以選擇自己的命名)。請注意這裡的命名非常的重要，因此它就是您的 Android 本機服務的代表名稱，稍後的 App 以及遠端 App 都要使用這個名稱來連結此 Android 服務。

接著再把項目中的 UnitX.pas 重新命名，例如在本範例中命名為 uMyLocalService.pas：



uMyLocalService 的宣告如下：

```
TAndroidServiceDM = class(TAndroidService)
```

它是從 TAndroidService 類別繼承下來，而 TAndroidService 是從 TAndroidBaseService 類別繼承下來：

```
TAndroidService = class(TAndroidBaseService)
```

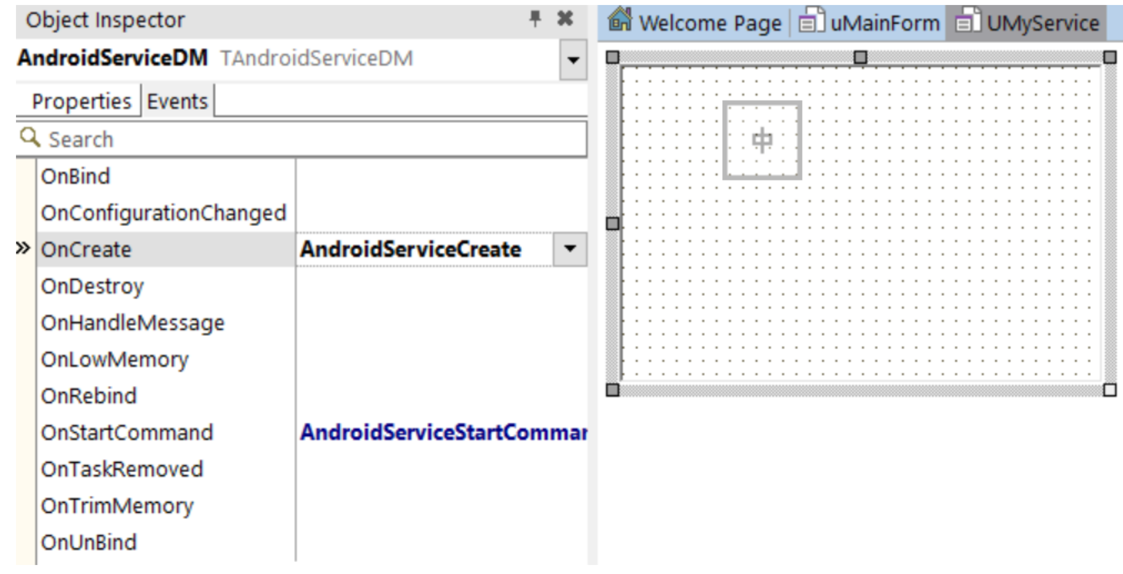
最後 TAndroidBaseService 則是從 TDataModule 類別繼承下來：

版權所有 請勿翻印

```
TAndroidBaseService = class(TDataModule)
```

因此開發人員可以像一般的資料模組一樣在 TAndroidServiceDM 中使用 Delphi 的其他元件。

現在讓我們為 TAndroidServiceDM 定義 2 個事件，OnCreate 和 OnStartCommand：



在 OnCreate 事件中我們把 TAndroidServiceDM 初次被建立的時間記錄下來：

```
uses System.DateUtils;

procedure TAndroidServiceDM.AndroidServiceCreate(Sender:
TObject);
begin
  dtStart := Now;
end;
```

版權所有 請勿翻印

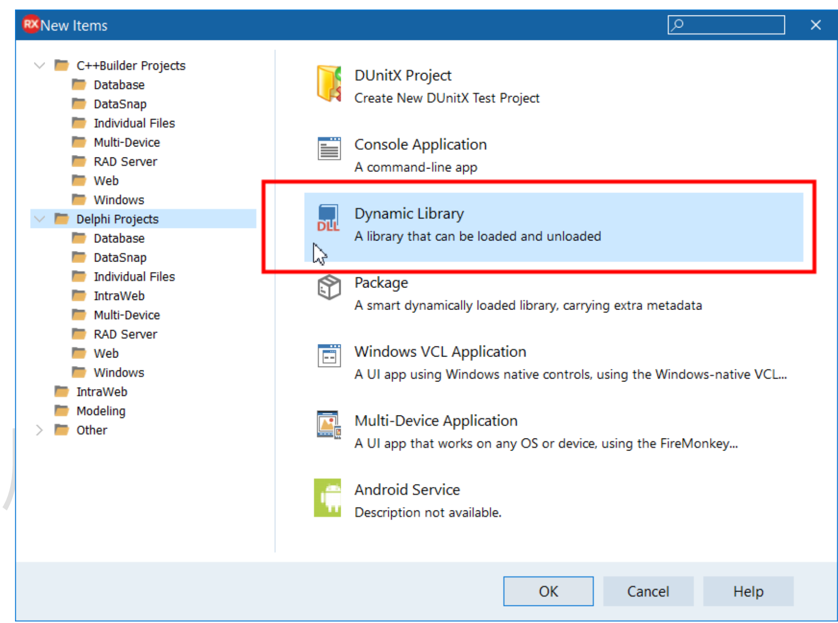
在 OnStartCommand 事件中我們設定此範例服務的型態為 START_STICKY，並且記錄每次服務重新開機的時間：

```
function TAndroidServiceDM.AndroidServiceStartCommand(const
Sender: TObject;
  const Intent: JIntent; Flags, StartId: Integer): Integer;
begin
  Result := TJService.JavaClass.START_STICKY;
  dtReStart := Now;
end;
```

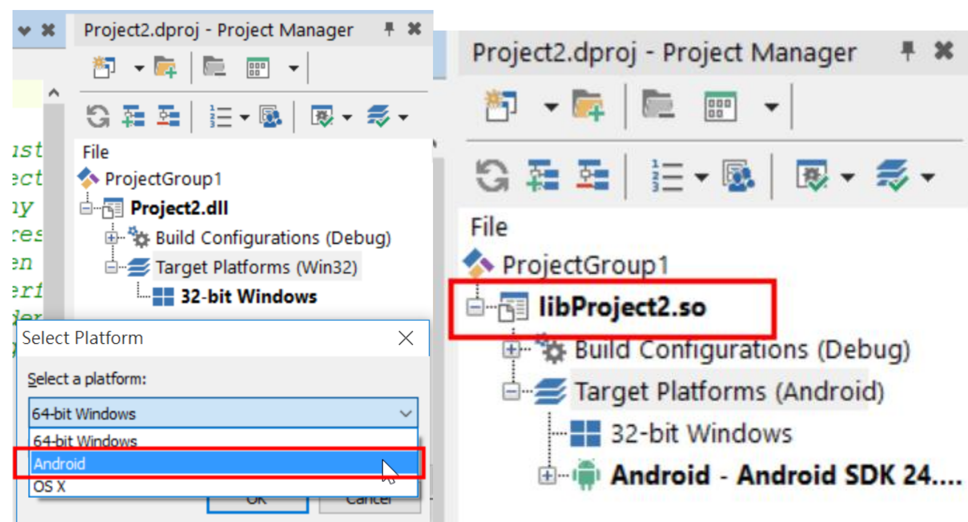
接著我們可以在 TAndroidServiceDM 中定義公用方法，如此一來就可以讓 App 呼叫服務中的方法：

```
public
  { Public declarations }
  function ReadString : String;
  function GetStartTime : TDateTime;
  function GetReStartTime : TDateTime;
end;
```

由於 App 能夠呼叫服務的公用方法，因此這也可以想成是 Android 平臺的 DLL，例如如果讀者在 RIO 中建立 DLL 的專案：



再於 DLL 項目中選擇平臺可以看見現在可建立 Android 平合的 DLL 了，一旦程式師選擇 Android 平臺就可以看見項目結尾檔名改成.so 的名稱和前面的服務專案是一樣的：



ＯＫ，回到服務專案繼續完成剛才加入的 3 個公用方法：

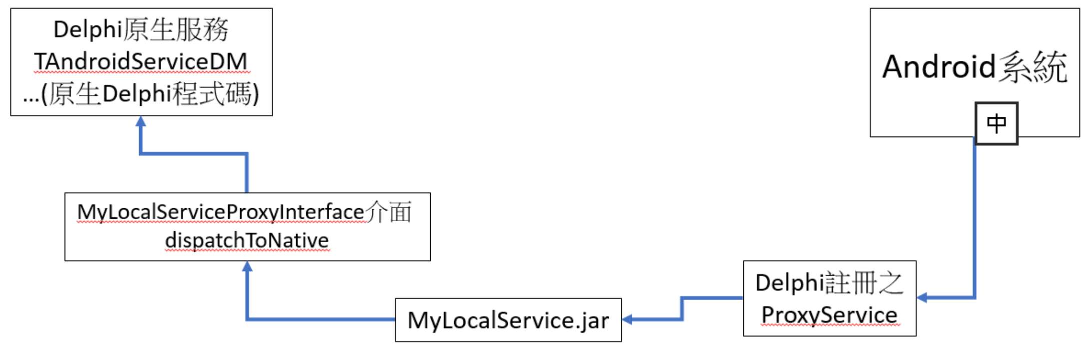
```
function TAndroidServiceDM.ReadString: String;
begin
  Result := '這是 Delphi 開發的服務!';
end;

function TAndroidServiceDM.GetReStartTime: TDateTime;
begin
  Result := dtReStart;
end;

function TAndroidServiceDM.GetStartTime: TDateTime;
begin
  Result := dtStart;
end;
```

版權所有 請勿翻印

現在我們可以編譯此範例服務專案，編譯完成之後我們可以在專案的 Android 子目錄中看到如下的重要檔案，Delphi 會產生服務的.so 檔以及一個和服務同名的.jar 檔：

Name	Ext	Size	Date	Attr
[..]		<DIR>	2015/08/05 16:50	----
AndroidManifest	xml	1,902	2015/08/05 16:50	-a--
libMyLocalService	so	18,819,280	2015/08/05 16:50	-a--
MyLocalService	jar	3,267	2015/08/05 16:50	-a--

和服務同名的.jar 文件扮演了重要的功能，它是由 IDE 根據服務名稱自動產生的，Delphi 藉由下面的架構實作了 Android Java 核心如何呼叫由 Delphi 原生程式碼撰寫的服務：

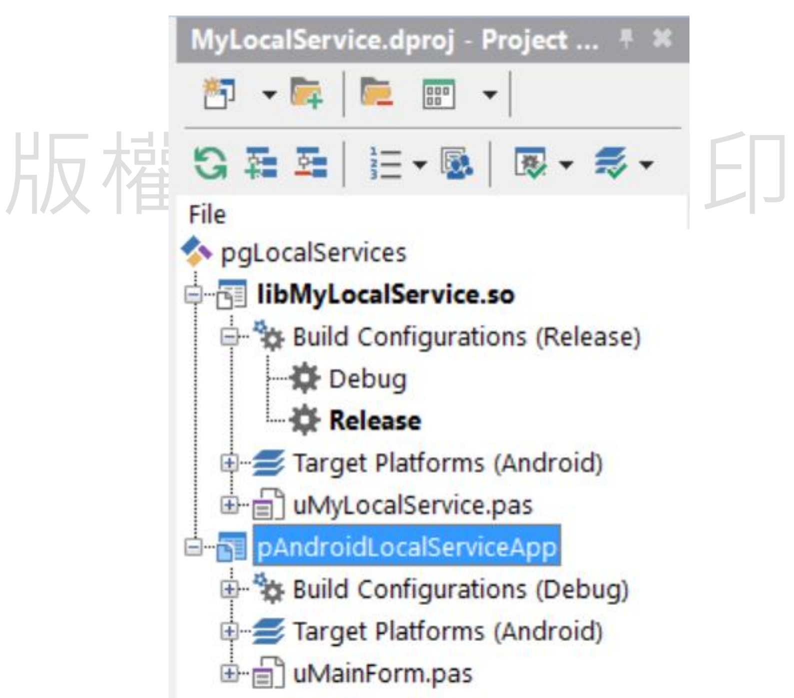


完成了 Android 服務之後就可以開始開發使用此 Android 服務的 Delphi App 了。接下來要學習如何把地服務加入到 App 中。

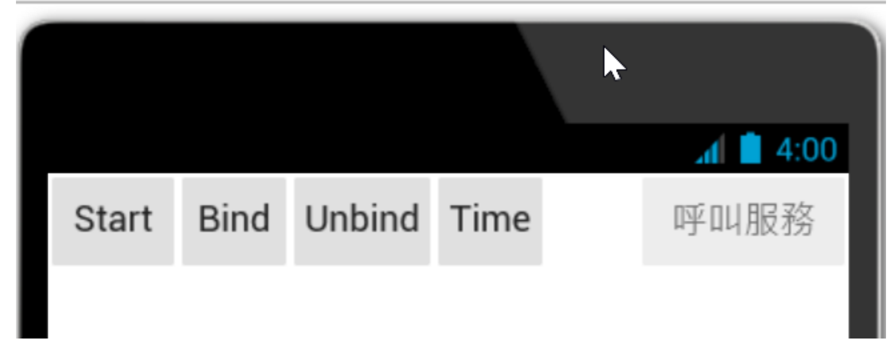
步驟 2： 開發包含 Android 服務的 App

現 1 在不原本的項目中加入一個 Multi-Device Application 項目，並命名 pAndroidLocalServiceApp，把主表單命名為 uMainForm.pas：

版權 印



開啟主表單並加入如下的按鈕：



主表單中的 Start 按鈕使用前面介紹的 TLocalServiceConnection 元件的 StartService 方法啟動服務，我們需要把服務名稱傳入做為參數。由於在前面我們命名我們的服務名稱為'MyLocalService'，因此傳入 StartService 方法的參數就必須是'MyLocalService'：

```
procedure TForm2.btnStartServiceClick(Sender: TObject);
begin
  TLocalServiceConnection.StartService('MyLocalService');
end;
```

當然要記得在主表單程式單元中加入使用：

版權所有 請勿翻印

```
System.Android.Service
```

Bind 按鈕先建立 TLocalServiceConnection 物件，設定系結服務成功之後呼叫 OnConnectLocalService 事件，最後呼叫 BindService 方法並系結已啟動的'MyLocalService'服務：

```
procedure TForm2.btnBindServiceClick(Sender: TObject);
begin
  if (FLocalService = Nil) then
  begin
    FLocalService := TLocalServiceConnection.Create;
    FLocalService.OnConnected := OnConnectLocalService;
  end;
  FLocalService.BindService('MyLocalService');
end;
```

而 FLocalService 當然是一個型態為 TLocalServiceConnection 的物件變數：

```
  private
    { Private declarations }
      FLocalService: TLocalServiceConnection;
      procedure OnConnectLocalService(const ALocalService:
TAndroidBaseService);
```

OnConnectLocalService 事件只是啟動主表單中’ 呼叫服務’ 和’Time’按鈕，因為只有在成功系結’MyLocalService'服務之後才能呼叫服中的方法：

```
procedure TForm2.OnConnectLocalService(
  const ALocalService: TAndroidBaseService);
begin
  btnGetServiceTime.Enabled := True;
  btnCallService.Enabled := True;
end;
```

Unbind 按鈕則是切斷系結的服務並釋放 FLocalService 物件：

```
procedure TForm2.btnUnbindServiceClick(Sender: TObject);
begin
  if (FLocalService <> Nil) then
  begin
    FLocalService.UnbindService;
    FreeAndNil(FLocalService);
  end;
End;
```

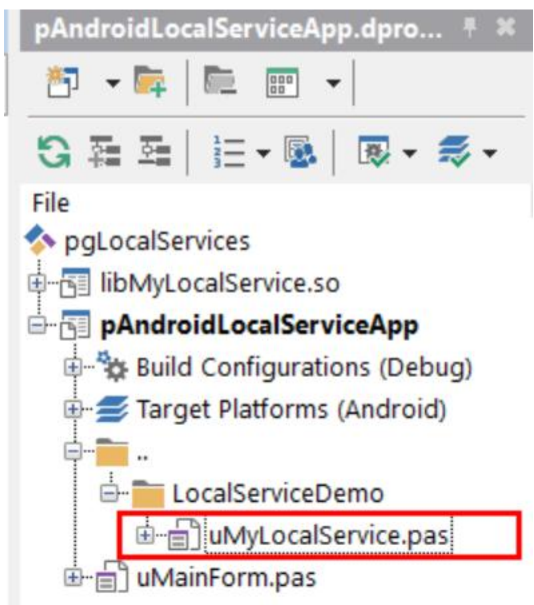
版權所有 請勿翻印

‘呼叫服務’按鈕是呼叫’MyLocalService'服務中的 ReadString 方法並顯示服務回傳的字串。在‘呼叫服務’按鈕展示了如何呼叫的重要技巧，當 App 成功啟動並系結服務後就可以藉由 TLocalServiceConnection 物件的 LocalService 特性取得服務物件，接下來我們再把此服務物件轉換型態為 TAndroidServiceDM 類別物件，之後就可以呼叫其中的方法了：

```
procedure TForm2.btnCallServiceClick(Sender: TObject);
begin
  if ((FLocalService <> Nil) and (FLocalService.LocalService <>
Nil)) then
  begin

ShowMessage( TAndroidServiceDM(FLocalService.LocalService).ReadS
```

```
tring() );
  end;
end;
```

當然我們要先把服務的程式單元加入到 App 的專案中：



版權所有 請勿翻印

並在主表單中加入使用此服務的程式單元：

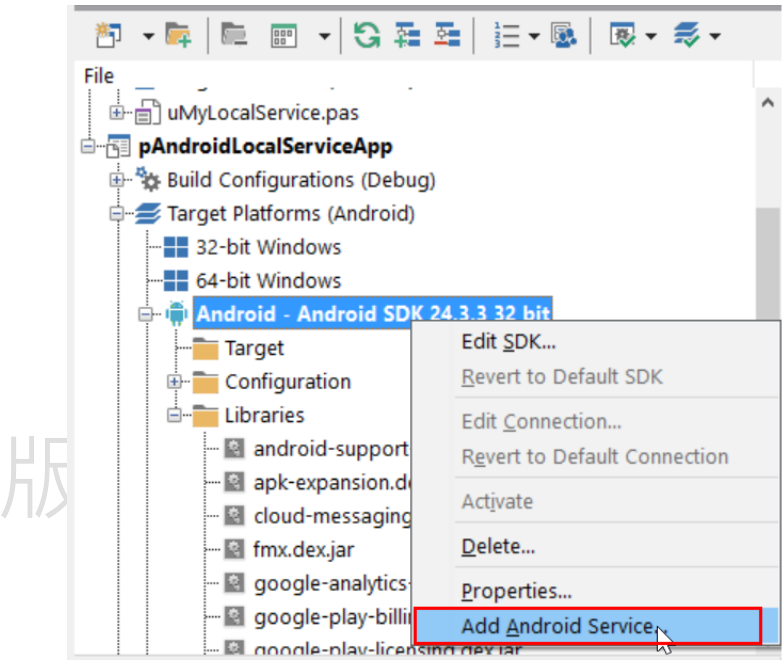
```
uses uMyLocalService;
```

最後的 Time 按鈕則是向服務取得服務的啟動時間和重新開機時間以及總執行時間。它使用了向'呼叫服務'按鈕相同的技巧：

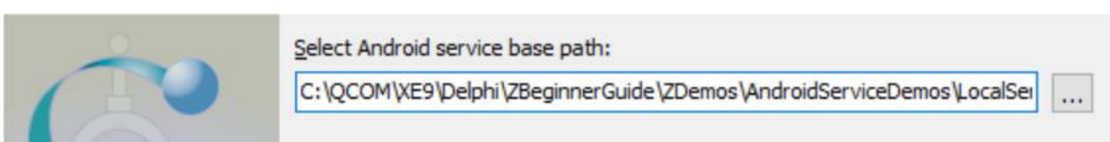
```
procedure TForm2.btnGetServiceTimeClick(Sender: TObject);
begin
  if ((FLocalService <> Nil) and (FLocalService.LocalService <>
Nil)) then
  begin
    ListView1.Items.Add.Text :=
'----------------------------------';
    ListView1.Items.Add.Text := '現在時間 : ' + DateTimeToStr(Now);
    ListView1.Items.Add.Text := '服務建立時間 : ' +
DateTimeToStr(TAndroidServiceDM(FLocalService.LocalService).GetS
tartTime());
    ListView1.Items.Add.Text := '服務重啟時間 : ' +
DateTimeToStr(TAndroidServiceDM(FLocalService.LocalService).GetR
```

```
eStartTime());
    ListView1.Items.Add.Text := '服務執行時間 : ' +
SecondsBetween(TAndroidServiceDM(FLocalService.LocalService).Get
ReStartTime(), Now).ToString;
  end;
end;
```

現在請在項目群組中使用 Build All 同時編譯服務和 App 專案，成功之後點選 App 項目的 Android 平臺右擊滑鼠再點選 Add Android Service…選項：



IDE 接著會顯示如下的對話盒詢問服務所在的目錄，請點選您的服務專案的目錄：



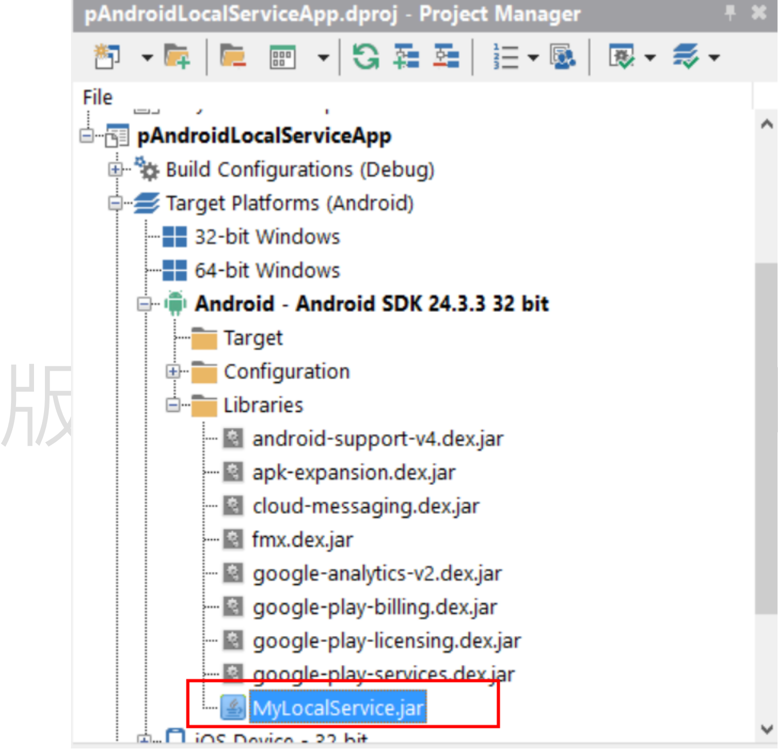
點選'Next >>'按鈕後 Add new Android Service 對話盒便會找到其中的服務.so 檔，服務.jar 檔以及服務的 pas 檔：

TAndroidServicesIDEWizardNewServiceInfoPage

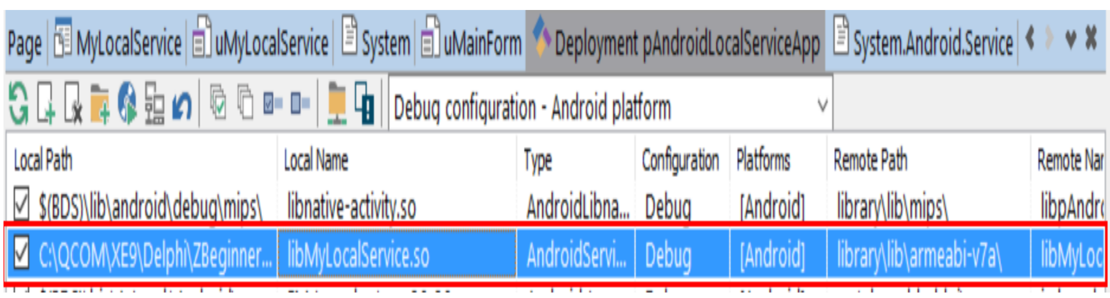
The following files will be added in the project C:\QCOM\XE9\Delphi\ZBeginnerGuide\ZDemos\AndroidServiceDemos\LocalService\ServiceApp\pAndroidLocalServiceApp.dproj

C:\QCOM\XE9\Delphi\ZBeginnerGuide\ZDemos\AndroidServiceDemos\LocalService\LocalServiceDemo\Android\Release\libMyLocalService.so
C:\QCOM\XE9\Delphi\ZBeginnerGuide\ZDemos\AndroidServiceDemos\LocalService\LocalServiceDemo\Android\Release\MyLocalService.jar
C:\QCOM\XE9\Delphi\ZBeginnerGuide\ZDemos\AndroidServiceDemos\LocalService\LocalServiceDemo\uMyLocalService.pas

再點選’Finish’按鈕後就可以看到 IDE 把服務.jar 檔加入到了項目的 Libraries 中：



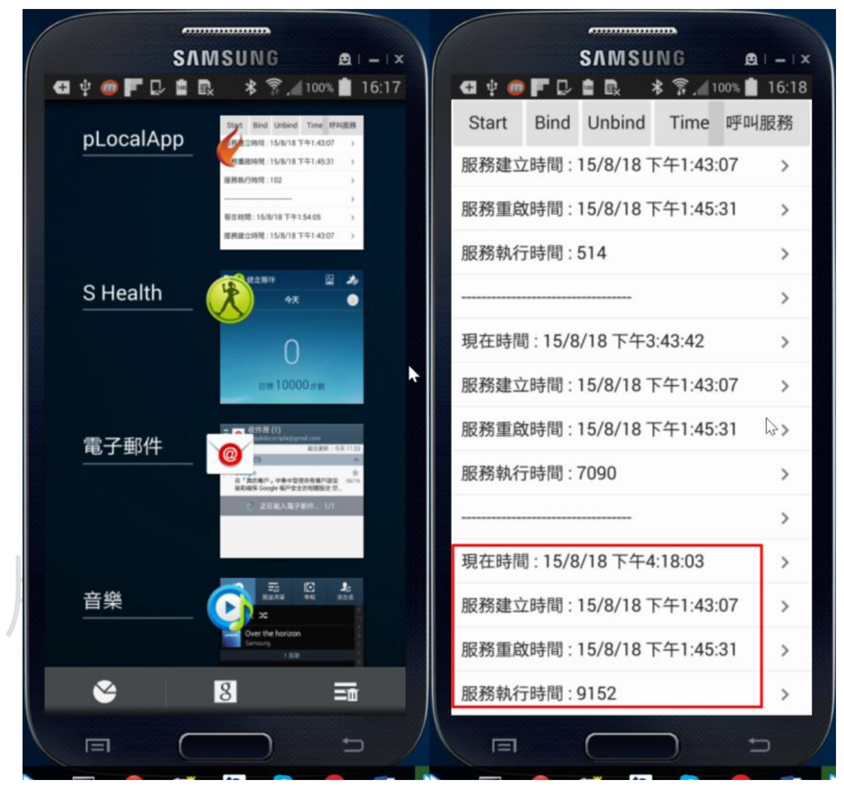
同時把服務.so 檔加入部署到 App 專案中：



最後並會在 App 專案的 AndroidManifest.xml 檔中加入如下的服務資訊：

```
    <service android:exported="false"
android:name="com.embarcadero.services.MyLocalService" />
```

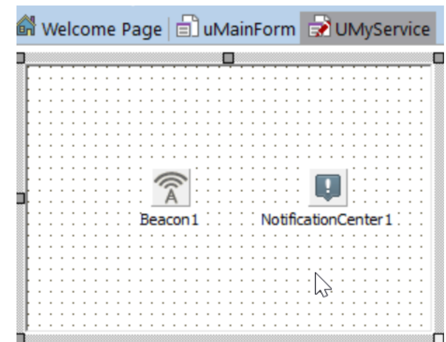
請執行此範例 App，您就可以在 Android 手機中看到類似如下的執行畫面，當您把此範例 App 丟到後面並執行其他的 App，最後再點選此範例 App 就可以看到它在背景時也持續的執行中：



恭喜您，您已經使用 Delphi 完成了您的第 1 個 Android 服務了。

在繼續說明如何開發遠端服務之前再讓我們為這個範例加入偵測 Beacon 設備的功能並證明在關閉 App 之後服務仍然可以繼續執行。

回到範例服務專案，在 TAndroidServiceDM 中加入 TBeacon 和 TNotificationCenter 元件：



在 TAndroidServiceDM 的 OnCreate 事件中開啟 Beacon：

```
procedure TAndroidServiceDM.AndroidServiceCreate(Sender:
TObject);
begin
  dtStart := Now;
  Beacon1.Enabled := True;
end;
```

再於 TBeacon 的 OnBeaconProximity 事件中呼叫 NotifyBeaconProximity 方法通知 App 偵測到 Beacon：

版權所有 請勿翻印

```
procedure TAndroidServiceDM.Beacon1BeaconProximity(const Sender:
TObject;
  const ABeacon: IBeacon; Proximity: TBeaconProximity);
begin
  if (Proximity = TBeaconProximity.Immediate) then
    NotifyBeaconProximity(ABeacon.GUID.ToString + ':' +
ABeacon.Major.ToString() + ',' + ABeacon.Minor.ToString());
end;
```

最後 NotifyBeaconProximity 方法使用 TNotificationCenter 元件觸發 Android 的通知功能：

```
procedure TAndroidServiceDM.NotifyBeaconProximity(const
sBeaconName: String);
var
  serviceNotification : TNotification;
begin
  serviceNotification := NotificationCenter1.CreateNotification;
  try
```

```
    serviceNotification.Name := '偵測 Beacon 通知!';
    serviceNotification.AlertBody := '偵測到 Beacon:' + sBeaconName;
    NotificationCenter1.PresentNotification(serviceNotification);
  finally
    serviceNotification.Free;
  end;
end;
```

現在您可以執行此 App，啟動服務再系結服務，關閉 App，最後移動到 Beacon 設備處就可以收到偵測到 Beacon 設備的通知訊息了。

19 新的 JSON 類別庫(XE11)

現今的應用軟體愈來愈開放也愈來愈依賴使用 JSON/RESTful 的架構，Delphi 早在數個版本之前就支援 JSON 的開發，例如 System.JSON 程式單元中的各個 JSON 相關類別。但由於 System.JSON 中的類別屬於早期的設計，到今日應用程式更積極趨向更快，更小的方向發展，因此提供一個更有彈性和更快速的 JSON 框架並應用於 Delphi 的 DataSnap，REST 和開發人員的 App 中就更顯需求了。因此在 Delphi XE11 中開始提供新一代的 JSON 框架。

版權所有 請勿翻印

這個新的 JSON 框架設計目標如下：

- 更快，更節省資源
- 可提供 JSON 物件和 Delphi 物件之間的序列化功能
- 可提供開發人員客制化的能力
- 支援 BSON

新的 JSON 框架主要是由數個類別所組成，它們包含了：

- TJSONReader 以及其衍生類別，例如 TJSONTextReader，TBSONReader
- TJSONWriter 以及其衍生類別，例如 TJSONTextWriter，TBSONWriter
- TJSONObjectBuilder，TJSONArrayBuilder 等封裝 TJSONWriter 的輔助類別
- TJSONConverter 提供型態轉換的類別

當然其中還有更多的類別無無一一列出，不過開發人員基本上只需要使用上述的類別應該就能完成大部份的工作。

19-1 JSON Reader/Writer 類別

新的 JSON 類別庫提供了 Reader/Writer 新類別，所謂的 Reader 是指從字串或是 2 進位元串列流讀取 JSON 的資料，而 Writer 則是指把字串資料封裝成 JSON 物件。

在 XE11 之前的版本中 Delphi 是使用 TJSONValue，TJSONObject，TJSONArray 等類別來處理 JSON 相關的開發，這些類別是使用 DOM 模型實作的。而新的 JSON Reader/Writer 類別則改用 Stream 模型實作。JSON Reader 類別是由 TJsonReader 這個抽象類別開始，真正的實作類別是 TJsonTextReader，它是從 TJsonReader 繼承下來。

而 JSON Writer 類別是由 TJsonWriter 這個抽象類別開始，接著有衍生類別 TJsonTextWriter 和 TJsonObjectWriter。TJsonTextWriter 是直接可把資料撰寫成 JSON 格式的資料，而 TJsonObjectWriter 則是把資料直接寫成 JSON 物件的格式。下面的圖形說明了新的 JSON 框架和舊的 JSON DOM 框架在實作上的不同：

版權所有 請勿翻印

其實新的 JSON 框架使用上非常的簡單，讓我們使用下的 JSON 資料來說明如何藉由 TJsonWriter 的相關類別輸出下面 JSON 格式的資料，在下一小節再使用 TJsonrReader 的相關類別再把資料讀出：

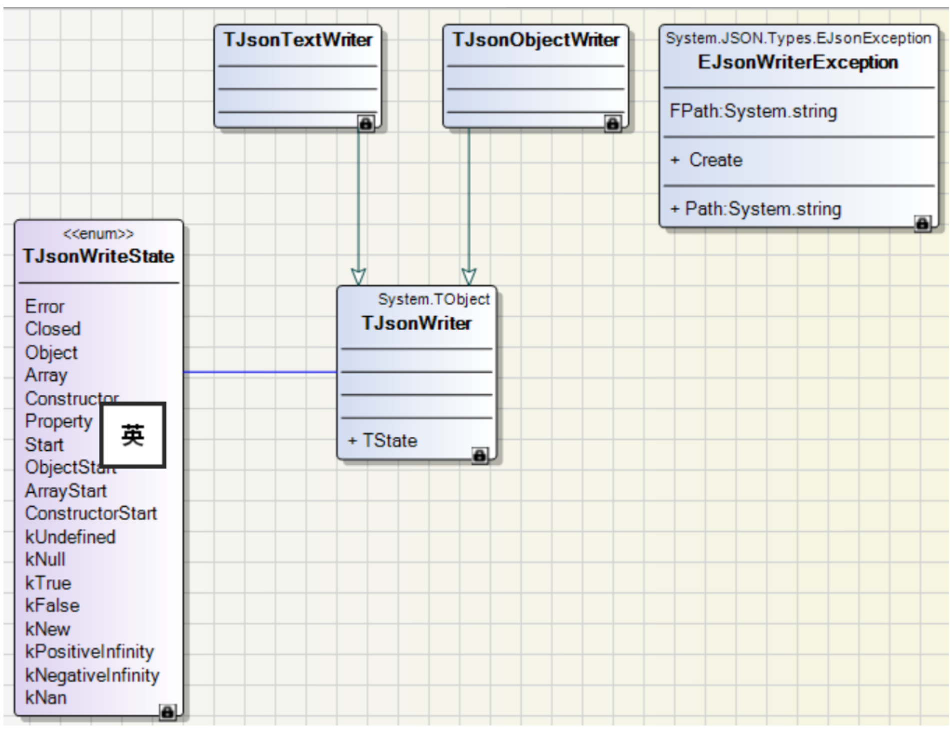
```
{
        "name" : "王小華",
```

```
        "account" : "WSH",
        "country" : "tw",
        "age" : “22”,
        "email" : [
                "wsh@gmail.com.tw",
                "wsh@hotmail.com"
        ]
}
```

上面的範例 JSON 格式的資料也使用在 FireDAC 資料庫中 MongoDB 的章節中，因為 MongoDB 就是使用 JSON/BSON 格式的資料，而 FireDAC 也使用了新的 JSON 框架來處理 JSON/BSON 格式的資料。

19-1-1 Writer 類別

新的 JSON Writer 類別是由 TJsonWriter，TJsonTextWriter 和 TJsonObjectWriter 等類別組成，下圖是它們的類別圖：



下表說明了每個類別的功能：

類別	說明
TJsonWriter	根抽象類別，實際功能由它的衍生類別實作
TJsonTextWriter	把資料寫成字串或是串列流格式的 JSON 資料

TJsonObjectWriter	把資料寫成的 JSON Object 格式的資料

要把資料寫成 JSON 的格式我們可以直接使用 TJsonTextWriter 類別，TJsonTextWriter 的建構元接受一個 TTextWriter 的物件：

```
constructor Create(const TextWriter: TTextWriter);
```

TJsonTextWriter 類別提供了許多的 Write 方法讓程式師呼叫，例如上面的資料格式是 JSON 物件，因此 TJsonTextWriter 類別提供了 WriteStartObject/WriteEndObject 方法，如果要寫出 JSON 陣列物件，那可呼叫 WriteStartArray/WriteEndArray 方法。要寫出 JSON Pair 的名稱可用 WritePropertyName，而要寫出 JSON Pair 的數值部分則可用各種覆載的 WriteValue 方法。

例如要使用 TJsonTextWriter 寫出前面”王小華”的 JSON 資料，我們可以使用下面的程式碼：

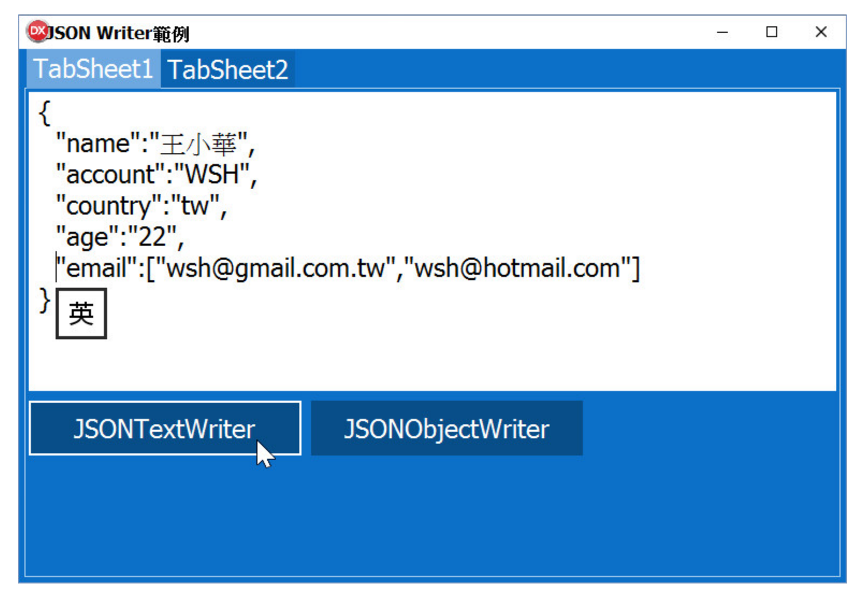
```
001   procedure TForm2.Button1Click(Sender: TObject);
002   var
003     sw : TStringWriter;
004     jtw : TJSONTextWriter;
005
006     procedure WritePair(const sName, sValue : String);
007     begin
008       jtw.WritePropertyName(sName);
009       jtw.WriteValue(sValue);
010     end;
011
012   begin
013     sw := TStringWriter.Create;
014     try
015       jtw := TJsonTextWriter.Create(sw);
016
017       jtw.WriteStartObject;
018       WritePair('name', '王小華');
019       WritePair('account', 'WSH');
020       WritePair('country', 'tw');
021       WritePair('age', '22');
022
023       //JSON 陣列
```

版權所有 請勿翻印

```
024       jtw.WritePropertyName('email');
025       jtw.WriteStartArray;
026       jtw.WriteValue('wsh@gmail.com.tw');
027       jtw.WriteValue('wsh@hotmail.com');
028       jtw.WriteEndArray;
029       jtw.WriteEndObject;
030       Memo1.Lines.Text := sw.ToString;
031     finally
032       jtw.Close;
033       jtw.Free;
034       sw.Free;
035     end;
036   end;
```

013 行先建立一個 TStringWriter 物件，準備把 JSON 資料寫入，015 行建立 TJsonTextWriter 物件。由於"王小華"是使用 JSON 物件封裝，因此 017 行呼叫 WriteStartObject 方法寫出'{'，接著呼叫 WritePair 方法呼叫 TJsonTextWriter 類別的 WritePropertyName 和 WriteValue 方法寫出 JSON Pair 資料，最後寫出 email 這個 JSON Pair 資料，但由於 email 的 Value 值又是使用 JSON 陣列封裝，因此 025 行再呼叫 WriteStartArray 方法寫出'['，最後 028，029 行分別呼叫 WriteEndArray 和 WriteEndObject 寫出']'和'}'即可完成，025 行呼叫 TJsonTextWriter 類別的 Close 方法完成寫入 TStringWriter 物件的工作。下圖是此範例執行在 Windows 10 的結果：

版權所有 請勿翻印



同樣的工作我們也可以使用 TJSONObjectWriter 類別來完成：

```
001    procedure TForm2.Button2Click(Sender: TObject);
002    var
003      jow : TJSONObjectWriter;
004
005      procedure WritePair(const sName, sValue : String);
006      begin
007        jow.WritePropertyName(sName);
008        jow.WriteValue(sValue);
009      end;
010
011    begin
012      try
013        jow := TJSONObjectWriter.Create(true);
014
015        jow.WriteStartObject;
016        WritePair('name', '王小華');
017        WritePair('account', 'WSH');
018        WritePair('country', 'tw');
019        WritePair('age', '22');
020
021        //JSON 陣列
022        jow.WritePropertyName('email');
023        jow.WriteStartArray;
024        jow.WriteValue('wsh@gmail.com.tw');
025        jow.WriteValue('wsh@hotmail.com');
026        jow.WriteEndArray;
027        jow.WriteEndObject;
028        Memo1.Lines.Text := jow.JSON.ToString;
029      finally
030        jow.Close;
031        jow.Free;
032      end;
033    end;
```

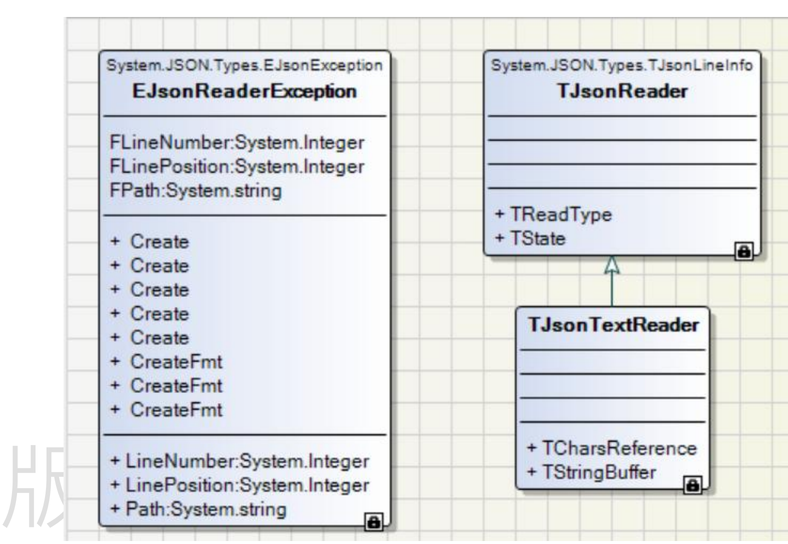
版權所有 請勿翻印

使用 TJSONObjectWriter 物件就不需要先建立 TStringWriter 物件，我們可以直接把資料寫入 TJSONObjectWriter 物件的記憶體中，最後在 028 行存取它的 JSON 特性值即可取得最的的結果。

TJSONWriter 在使用上非常的直覺又簡單，接下來我們就可以討論如何使用 Reader 類別。

19-1-2 Reader 類別

TJSONReader 類別的功能是從 JSON 格式的資料讀出我們需要的資料，因此 TJSONReader 類別是從 JSON 的字串中解析和讀取資料，所以 TJSONReader 類別組非常的簡單基本上目前只有 TJSONReader 和 TJSONTextReader 類別：



基本上 TJSONReader 類別在解析和讀取 JSON 資料時使用符號觸發處理的方式讓程式師處理 JSON 資料中的每一個元素，而原本的 System.JSON 中的類別則是需要程式師先瞭解 JSON 資料的實際格式再解析和讀取，也許讓我們使用一個簡單的範例來說明會更容易瞭解。

例如現在我們希望從下面的 JSON 物件中讀出"姓名"這個 JSON Pair 的數值：

```
{
  "姓名" : "王小華"
}
```

那麼在以前可以使用如下的程式碼：

```
001   procedure TForm2.Button4Click(Sender: TObject);
002   var
003     jv : TJSONValue;
004     jo : TJSONObject;
```

```
005      jp : TJSONPair;
006    begin
007      jv := TJSONObject.ParseJSONValue(SPEOPLEDATA);
008      try
009        if (jv is TJSONObject) then
010        begin
011          jo := jv as TJSONObject;
012          jp := jo.Pairs[0];
013          Memo2.Lines.Add(jp.JsonString.Value + ':' +
jp.JsonValue.Value);
014        end;
015      finally
016        jv.Free;
017      end;
018    end;
```

如果仔細看上面的程式碼可以發現程式師需要事先知道要處理的 JSON 格式：

版權所有 請勿翻印

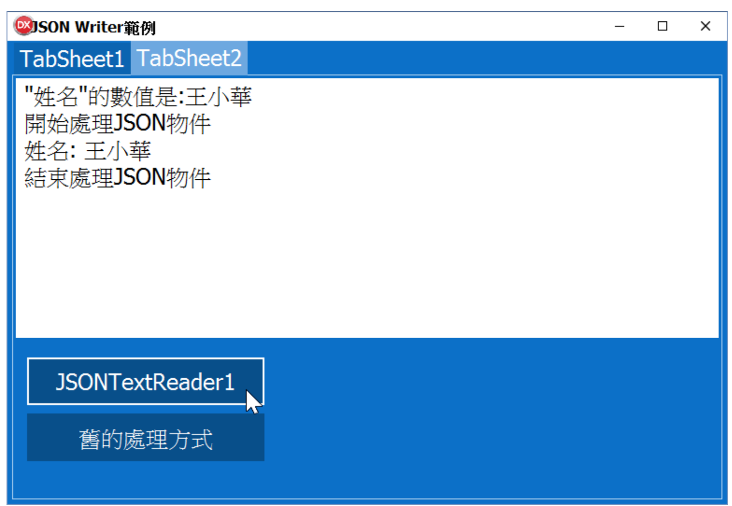
在新的 JSON 框架中可以使用如下的程式碼來解析 JSON 資料，009 行先把 JSON 數據傳入 TStringReader 物件，再于 010 行建立 TJSONTextReader 物件，013 行開始呼叫 TJSONTextReader 物件的 Read 方法讀取 JSON 資料，接著可使用一個 case 子句來判斷目前讀取到的內容再決定要如何處理：

```
001    const SPEOPLEDATA = '{"姓名" : "王小華"}';
002
003    procedure TForm2.Button3Click(Sender: TObject);
```

```
004    var
005      sr : TStringReader;
006      jr : TJSONTextReader;
007      sData : String;
008    begin
009      sr := TStringReader.Create(SPEOPLEDATA);
010      jr := TJSONTextReader.Create(sr);
011      try
012        jr.Rewind;
013        while (jr.Read) do
014        begin
015          case jr.TokenType of
016            TJsonToken.None: ;
017            TJsonToken.StartObject:
018            begin
019              Memo2.Lines.Add('開始處理 JSON 物件');
020            end;
021            TJsonToken.PropertyName:
022            begin
023              sData := jr.Value.ToString + ': ' + jr.ReadAsString;
024            end;
025            TJsonToken.String:
026            begin
027              sData := sData + jr.ReadAsString;
028            end;
029            TJsonToken.EndObject:
030            begin
031              Memo2.Lines.Add(sData);
032              Memo2.Lines.Add('結束處理 JSON 物件');
033            end;
034          end;
035        end;
036      finally
037        sr.Free;
038        jr.Free;
039      end;
040    end;
```

版權所有 請勿翻印

例如在 023 行我們可以存取 TJSONTextReader 物件的 Value 特性值取得目前讀取到的 JSON Pair 的名稱部分的內容(即"姓名")，再呼叫它的 ReadAsString 方法取得 JSON Pair 的數值部分的內容(即"王小華”)，而下圖是此 2 種方法執行的結果：



新的 JSON 框架讓程式師可以在事先不知道 JSON 資料格式的情形下更容易的解析和讀取 JSON 資料的內容。

版權所有 請勿翻印

因此我們可以使用下面的程式碼藉由 TJSONTextReader 物件來處理前面”王小華”的範例資料：

```
001    const SDEMODATA = '{"name" : "王小華", "account" : "WSH",
"country" : "tw", "age" : "22", "email" :
["wsh@gmail.com.tw","wsh@hotmail.com"]}';
002
003    procedure TForm2.Button5Click(Sender: TObject);
004    var
005      sr : TStringReader;
006      jr : TJSONTextReader;
007      sData : String;
008    begin
009      sr := TStringReader.Create(SDEMODATA);
010      jr := TJSONTextReader.Create(sr);
011      try
012        jr.Rewind;
013        while (jr.Read) do
014        begin
```

```
015         case jr.TokenType of
016           TJsonToken.StartObject:
017           begin
018             Memo2.Lines.Add('開始處理 JSON 物件');
019           end;
020           TJsonToken.PropertyName:
021           begin
022             if (jr.Value.ToString <> 'email') then
023             begin
024               sData := jr.Value.ToString + ': ' + jr.ReadAsString;
025               Memo2.Lines.Add(sData);
026             end;
027           end;
028           TJsonToken.EndObject:
029           begin
030             Memo2.Lines.Add('結束處理 JSON 物件');
031           end;
032           TJsonToken.StartArray:
033           begin
034             Memo2.Lines.Add('開始處理 JSON 陣列');
035             while (jr.Read) do
036             begin
037               case jr.TokenType of
038                 TJsonToken.String:
039                   Memo2.Lines.Add(jr.Value.ToString);
040                 TJsonToken.EndArray:
041                   Memo2.Lines.Add('結束始處理 JSON 陣列');
042               end;
043             end;
044           end;
045         end;
046       end;
047     finally
048       sr.Free;
049       jr.Free;
050     end;
```

版權所有 請勿翻印

上面的關鍵點在 032 行到 042 行，一旦 TJSONTextReader 物件處理到 JSON 陣列，我們就需要藉由 TJSONTextReader 物件的 Value 特性值來一一存取陣列中的每一個元素值，下面就是執行的結果，我們可以看到 TJSONTextReader 物件可以正確解析和讀取出 JSON 資料的內容：

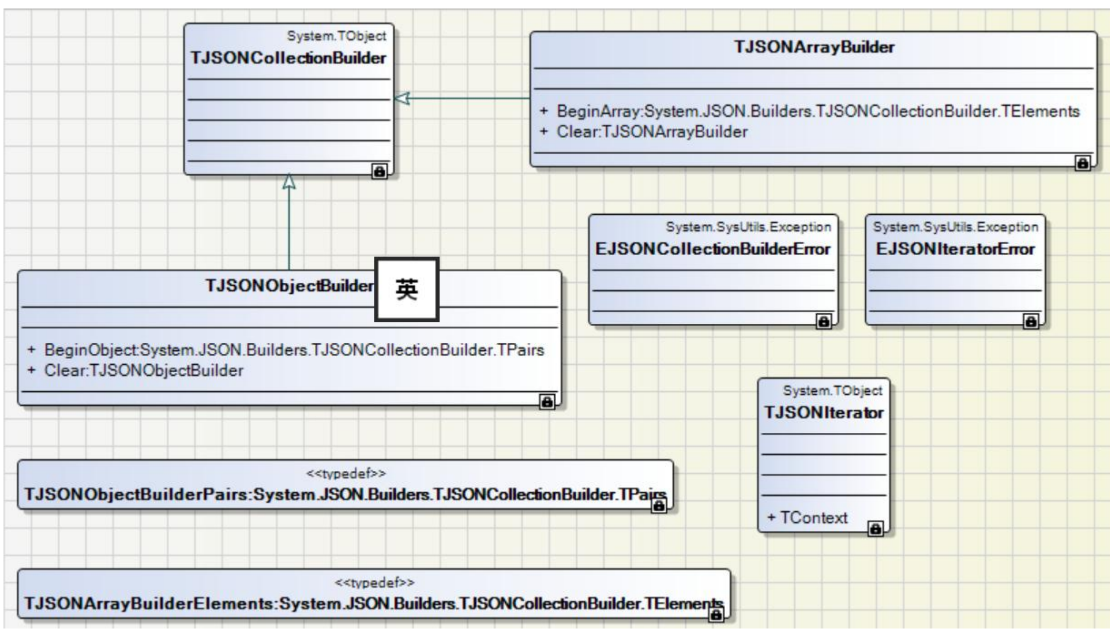
19-2 JSON Builder 類別

版權所有 請勿翻印

雖然使用新的 JSON Reader/Writer 類別可以比以前的 JSON 類別更方便快速的處理 JSON 資料，但是撰寫起程式碼來比較繁瑣，為了進一步幫助開發人員能更方便處理 JSON，因此在新的 JSON 框架中也提供了更方便使用的 JSON Builder 類別。

JSON Builder 類別使用了類似 Helper 類別的設計方式，為大多數方法都回傳方法的物件(Self)，因此程式師可以使用更簡潔的程式碼來完成工作。

JSON Builder 類別主要是由 TJSONCollectionBuilder，TJSONObjectBuilder，TJSONArrayBuilder 和 TJSONIterator 等類別組成的：



下面的表格簡介了 JSON Builder 類別的功能：

類別	說明
TJSONCollectionBuilder	JSON Builder 的抽象根類別，提供衍生類別的基礎功能
TJSONObjectBuilder	可建立 JSON 物件的 Builder 類別
TJSONArrayBuilder	可建立 JSON 陣列的 Builder 類別
TJSONIterator	執裝 TJSONReader 的類別，可讀取 JSONI 資料中的內容

版權所有 請勿翻印

為了幫助讀者瞭解如何使用這些新的 JSON Builder 類別，仍然讓我們繼續使用前面”王小華”的範例資料。

TJSONObjectBuilder 類別

TJSONObjectBuilder 類別是使用來建立 JSON 物件的，下面是它的宣告:

```
TJSONObjectBuilder = class(TJSONCollectionBuilder)
public
  function BeginObject: TJSONCollectionBuilder.TPairs;
  function Clear: TJSONObjectBuilder;
end;
```

要建立 JSON 物件程式師只需要呼叫 TJSONObjectBuilder 的 BeginObject 方法，而 BeginObject 方法會回傳 TPairs 物件。TPairs 類別提供了許多 Add 覆載方法可在 JSON 物件中加入資料，請注意的是 Add 覆載方法又回傳原先的 TPairs 物件：

```
    TPairs = class(TBaseCollection)
    public
      function Add(const AKey: string; const AValue: string): TPairs;
overload; inline;
      function Add(const AKey: string; const AValue: Int32): TPairs;
overload; inline;
      …
    End;
```

因此程式師可使用下面的語法在 JSON 物件中加入資料：

```
joBuilder.Add().Add().Add()….  .EndAll;
```

上面的 **EndAll** 方法可在 JSON 物件中加入所有結尾符號，例如 JSON 陣列結尾']'， JSON 物件結尾'}'，因此在使用 **TJSONObjectBuilder** 物件之後要記得呼叫它：

```
    procedure EndAll;
```

因此要使用 **TJSONObjectBuilder** 物件寫出前面"王小華"的範例 JSON 資料，我們可以使用如下的程式碼：

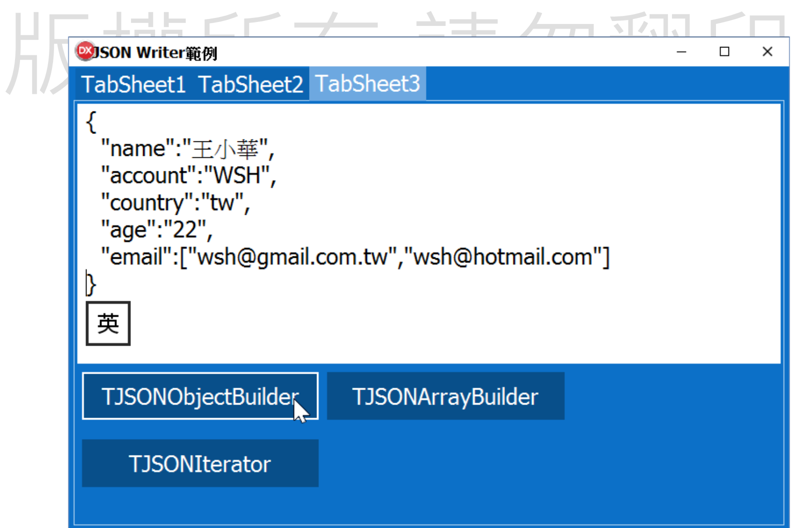
版權所有 請勿翻印

```
001    procedure TForm2.Button6Click(Sender: TObject);
002    var
003      sw : TStringWriter;
004      jtw : TJsonTextWriter;
005      joBuilder : TJSONObjectBuilder;
006    begin
007      sw := TStringWriter.Create;
008      jtw := TJsonTextWriter.Create(sw);
009      joBuilder := TJSONObjectBuilder.Create(jtw);
010      try
011        joBuilder.BeginObject
012          .Add('name','王小華')
013          .Add('account','WSH')
014          .Add('country','tw')
015          .Add('age','22')
016          .BeginArray('email')
017            .Add('wsh@gmail.com.tw')
018            .Add('wsh@hotmail.com')
```

```
019         .EndAll;
020
021         Memo3.Lines.Text := sw.ToString;
022       finally
023         joBuilder.Free;
024         jtw.Free;
025         sw.Free;
026       end;
027     end;
```

007 和 008 行分別建立了 TStringWriter 和 TJsonTextWriter 物件，009 行再建立 TJSONObjectBuilder 物件並把 TJsonTextWriter 物件傳入。從 011 行開始先呼叫 BeginObject 方法，因為"王小華"是要封裝在 JSON 物件中，接著呼叫一連串的 Add 方法在其中加入 4 個 JSON Pair，接著呼叫 BeginArray 方法開始加入 JSON 陣列，再呼叫 2 個 Add 方法在其中加入 2 個 JSON 字串，最後呼叫 EndAll 方法完成所有寫入 JSON 資料的工作，下圖就是執行上面程式碼的執行結果，我們成功的使用 TJSONObjectBuilder 物件寫出了正確的 JSON 資料：

版權所有 請勿翻印



TJSONArrayBuilder 類別

TJSONArrayBuilder 類別是使用來建立 JSON 陣列的，它的 BeginArray 方法可以開始寫出 JSON 陣列的內容，而且它也是回傳 TElements 物件：

```
TJSONArrayBuilder = class(TJSONCollectionBuilder)
public
  function BeginArray: TJSONCollectionBuilder.TElements;
```

```
    function Clear: TJSONArrayBuilder;
  end;
```

而 TElements 類別也提供了許多 Add 方法可在 JSON 陣列中加入資料，此外 TElements 類別也提供了 BeginObject 和 BeginArray 方法，這代表 JSON 陣列中的元素也可以是一個 JSON 物件或是 JSON 陣列：

```
    TElements = class(TBaseCollection)
    public
      function Add(const AValue: string): TElements; overload;
inline;
      function Add(const AValue: Int32): TElements; overload;
inline;
      function Add(const AValue: UInt32): TElements; overload;
inline;
      ...

      function BeginObject: TPairs; overload; inline;
      function BeginArray: TElements; overload; inline;
      function EndArray: TParentCollection; inline; // Does not
return nil.

      /// <summary>Encapsulate this array so that there is no access
to the parent array or object</summary>
      function AsRoot: TElements;
      ...
    end;
```

版權所有 請勿翻印

例如下面的程式碼可以建立一個包含臺北市郵遞區號的 JSON 陣列：

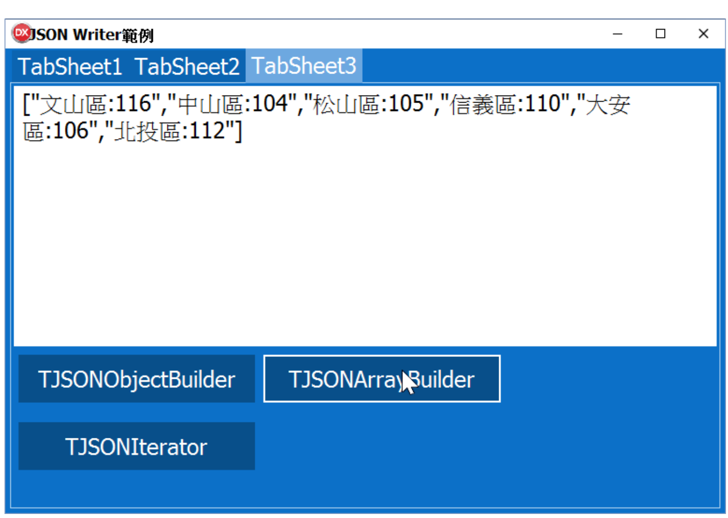
```
001    procedure TForm2.Button7Click(Sender: TObject);
002    var
003      sw : TStringWriter;
004      jtw : TJsonTextWriter;
005      jaBuilder : TJSONArrayBuilder;
006    begin
007      sw := TStringWriter.Create;
008      jtw := TJsonTextWriter.Create(sw);
009      jaBuilder := TJSONArrayBuilder.Create(jtw);
```

```
010      try
011        jaBuilder.BeginArray
012          .Add('文山區:116')
013          .Add('中山區:104')
014          .Add('松山區:105')
015          .Add('信義區:110')
016          .Add('大安區:106')
017          .Add('北投區:112')
018        .EndAll;
019        Memo3.Lines.Text := sw.ToString;
020      finally
021        jaBuilder.Free;
022      end;
023    end;
```

007 和 008 行同樣分別建立了 TStringWriter 和 TJsonTextWriter 物件，009 行再建立 TJSONArrayBuilder 物件並把 TJsonTextWriter 物件傳入，從 011 行開始先呼叫 BeginArray 方法建立 JSON 陣列，再呼叫一連串的 Add 方法在其中加入 JSON 字串，最後同樣呼叫 EndAll 方法結束。

版權所有 請勿翻印

下面就是執行的結果：



TJSONIterator 類別

TJSONIterator 類別是使用來更方便的讀取 JSON 資料的內容，使用它的基本概念是傳入一個 TJSONReader 物件給它，然後開始呼叫 Next 方法一一的讀取 JSON 資料，如果遇到 JSON 物件或是 JSON 陣列就呼叫 Recurse 方法

再一一處理 JSON 物件或是 JSON 陣列中的資料，如果沒有任何剩下的 JSON 資料，那 Next 方法就回傳 False。因此一個典型的程式碼架構就是：

```
while (jInerator.Next()) do
begin
  …
end;
```

當然在使用它處理 JSON 資料時，程式師仍然可以藉由它的 Type 特性值來判斷目前處理的 JSON 目標是什麼：

```
    property &Type: TJsonToken read FType;
```

例如如果我們想使用 TJSONIterator 類別來讀取其中下面"王小華"這筆 JSON 物件中的資料：

```
const SDEMODATA = '{"name" : "王小華", "account" : "WSH", "country" :
"tw", "age" : "22", "email" :
["wsh@gmail.com.tw","wsh@hotmail.com"]}';
```

那麼我們可以使用如下的程式碼：

版權所有 請勿翻印

```
001    procedure TForm2.Button8Click(Sender: TObject);
002    var
003      jInerator : TJSONIterator;
004      sr : TStringReader;
005      jr : TJSONTextReader;
006      sData : String;
007    begin
008      sData := '';
009      sr := TStringReader.Create(SDEMODATA);
010      jr := TJSONTextReader.Create(sr);
011      jInerator := TJSONIterator.Create(jr);
012      try
013        while (jInerator.Next()) do
014        begin
015          if ((jInerator.Index = -1) and (jInerator.&Type <>
TJsonToken.StartArray)) then
016            sData := sData + jInerator.Key + ':' +
jInerator.AsString + #13#10
```

```
017          else
018          begin
019            if (jInerator.Index = -1) then
020              sData := sData + jInerator.Key + ':'
021            else
022              sData := sData + #13#10 + jInerator.AsString;
023            jInerator.Recurse;
024          end;
025        end;
026        Memo3.Lines.Text := sData;
027      finally
028        jInerator.Free;
029        jr.Free;
030        sr.Free;
031      end;
032    end;
```

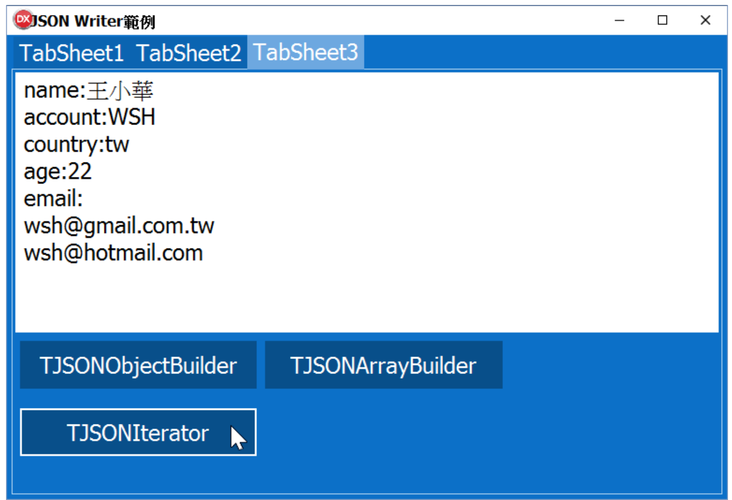
009 行使用 TStringReader 物件輸入"王小華"這筆 JSON 物件，010 行再建立 TJSONTextReader 物件，012 行 TJSONIterator 建立物件並傳入 TJSONTextReader 物件做為參數，接著呼叫 Next 方法進入迴圈一一處理每一個 JSON 元素。對於目前的 JSON 元素名稱可以存取 TJSONIterator 物件的 Key 特性值取得：

版權所有 請勿翻印

```
property Key: String read FKey;
```

而 JSON 元素值則可藉由存取它的各個 AsXXXX 特性值取得，例如 AsString 和 AsInteger 等。

017~025 行的程式碼主要是處理"王小華"這筆 JSON 物件中內含的 JSON 陣列資料，一旦進入 JSON 物件或是 JSON 陣列，那麼 Key 特性值就成為元素的索引值，而且要呼叫 Recurse 方法一一處理 JSON 物件或是 JSON 陣列中的每一個 JSON 元素。下圖就是上面使用 TJSONIterator 物件讀取出的資料結果:



19-3 BSON 類別

新的 JSON 框架中也包含了 TBsonWriter 和 TBsonReader 這 2 個能處理 BSON 的類別，基本上 BSON 就是 2 進位格式的 JSON，使用 BSON 來處理資料速度會比使用字串格式的 JSON 來得更有效率，例如 MongoDB 就是使用 BSON 處理和儲存資料。

19-3-1 TBsonWriter

版權所有 請勿翻印

TBsonWriter 類別的功能是把資料輸出成 BSON 的格式，它的建構元接收 TBinaryWriter 或是 TStream 物件：

```
constructor Create(const ABinaryWriter: TBinaryWriter); overload;
constructor Create(const Stream: TStream); overload;
```

由於建構元可接收 TStream 物件，因此我們可以傳入 TMemoryStream，TStringStream 和 TFileStream 等物件，TBsonWriter 便會把 BSON 格式的資料寫入。

TBsonWriter 類別和前面介紹的 JSON Writer 類別一樣提供了寫入各種 JSON 元素的方法，例如：

```
procedure WriteStartObject; override;
procedure WriteEndObject; override;
procedure WriteStartArray; override;
procedure WriteEndArray; override;
```

因此我們可以像前面說明的一樣呼叫這些方法寫入 JSON 元素，當然 TBsonWriter 類別也提供了各種 WriteValue 覆載方法可讓程式師寫入資料：

```
procedure WriteValue(const Value: string); override;
procedure WriteValue(Value: Integer); override;
…
```

其中一個重要的方法就是 WriteToken 方法：

```
procedure WriteToken(const Reader: TJsonReader); overload;
procedure WriteToken(const Reader: TJsonReader; WriteChildren:
Boolean); overload;
```

WriteToken 方法接收一個 TJsonReader 物件，接著 WriteToken 方法就從 TJsonReader 物件中讀出 JSON 資料再寫入 TBsonWriter 類別的 Stream 物件中。

現在讓我們繼續使用"王小華"這筆 JSON 資料並把它寫成 BSON 的格式再從 BSON 轉回 JSON。在下面的程式碼中 SDEMODATA 包含的就是"王小華"這筆 JSON 資料，我們先呼叫 Json2Bson 方法把 JSON 格式的資料轉成 BSON 格式，再呼叫 Bytes2String 方法把 BSON 格式的資料以文字的形式顯示出來(記得 BSON 是 2 進位元形式的資料嗎?)：

版權所有 請勿翻印

```
procedure TForm2.Button9Click(Sender: TObject);
begin
  Memo5.Lines.Text := Bytes2String(Json2Bson(SDEMODATA));
end;
```

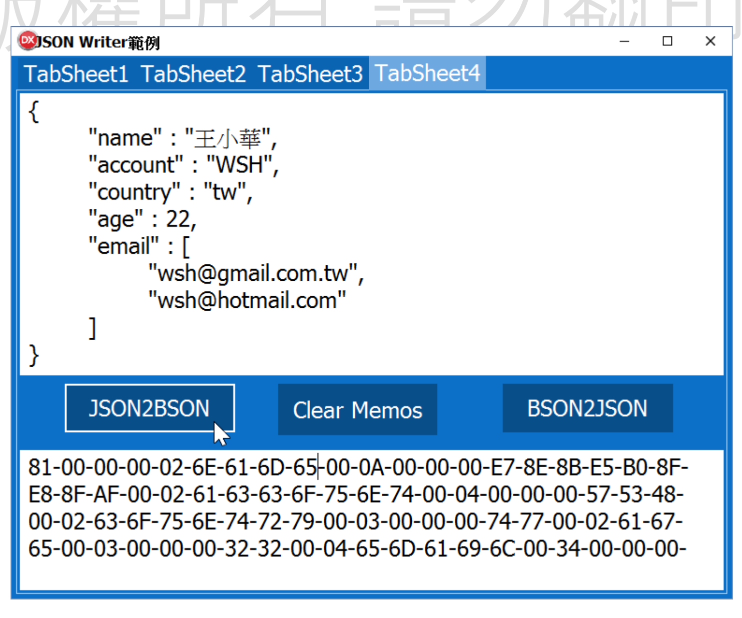
Json2Bson 方法就使用了 TBsonWriter 類別把 JSON 格式的資料轉成 BSON 格式，在下面的程式碼中首先先藉由 TJSONTextReader 物件把"王小華"這筆 JSON 資料讀入 010 行建立 TBytesStream 物件，011 行再建立 TBsonWriter 物件並傳入 TBytesStream 物件。接著我們只需要在 013 行呼叫 TBsonWriter 物件的 WriteToken 方法即可把 TJSONTextReader 物件中包含的 JSON 格式的資料以 BSON 的格式寫入 TBytesStream 物件中，最後在 016 行以 TBytes 型態回傳轉換後的 BSON 格式資料：

```
001    function TForm2.Json2Bson(const AJson: string): TBytes;
002    var
003      sr : TStringReader;
004      jr: TJSONTextReader;
005      BsonWriter: TBsonWriter;
006      Stream: TBytesStream;
007    begin
```

```
008     sr := TStringReader.Create(AJson);
009     jr := TJSONTextReader.Create(sr);
010     Stream := TBytesStream.Create;
011     BsonWriter := TBsonWriter.Create(Stream);
012     try
013       BsonWriter.WriteToken(jr);
014       SetLength(Result, Stream.Size);
015       Stream.Position := 0;
016       Stream.Read(Result, Stream.Size);
017     finally
018       jr.Free;
019       sr.Free;
020       BsonWriter.Free;
021       Stream.Free;
022     end;
023   end;
```

下圖就是執行的結果，我們可以看到可以成功的把"王小華"這筆 JSON 資料轉成下方 BSON 格式的資料：

版權所有 請勿翻印



同樣的我們也可以把 BSON 格式的資料轉回 JSON 格式的資料，下面的 Bson2Json 方法可把一個包含 BSON 格式資料的 TBytes 轉回 JSON：

```
001   function TForm2.Bson2Json(const ABson: TBytes): String;
002   var
003     BsonReader: TBsonReader;
```

```
004     Stream: TBytesStream;
005     sw : TStringWriter;
006     jtw : TJsonTextWriter;
007   begin
008     Stream := TBytesStream.Create(ABson);
009     BsonReader := TBsonReader.Create(Stream);
010     sw := TStringWriter.Create;
011     jtw := TJsonTextWriter.Create(sw);
012     jtw.Formatting := TJsonFormatting.Indented;
013     try
014       jtw.WriteToken(BsonReader);
015       Result := sw.ToString;
016     finally
017       jtw.Free;
018       sw.Free;
019       BsonReader.Free;
020       Stream.Free;
021     end;
022   end;
```

版權所有 請勿翻印

008 行先使用 TBytesStream 接收傳入的包含 BSON 格式資料的 TBytes，009 行使用下一小節介紹的 TBsonReader 物件讀入並解析 BSON 格式資料，014 行呼叫 TJsonTextWriter 類別的 WriteToken 方法把 BSON 轉成 JSON 即可。

TJsonTextWriter 類別的 WriteToken 方法宣告如下，我們只須傳入 TJsonReader 物件它即可幫我們轉換資料：

```
procedure WriteToken(const Reader: TJsonReader); overload;
```

而 TBsonReader 是從 TJsonReader 類別繼承下來，因此它就可以把 BSON 轉成 JSON：

```
TBsonReader = class(TJsonReader)
```

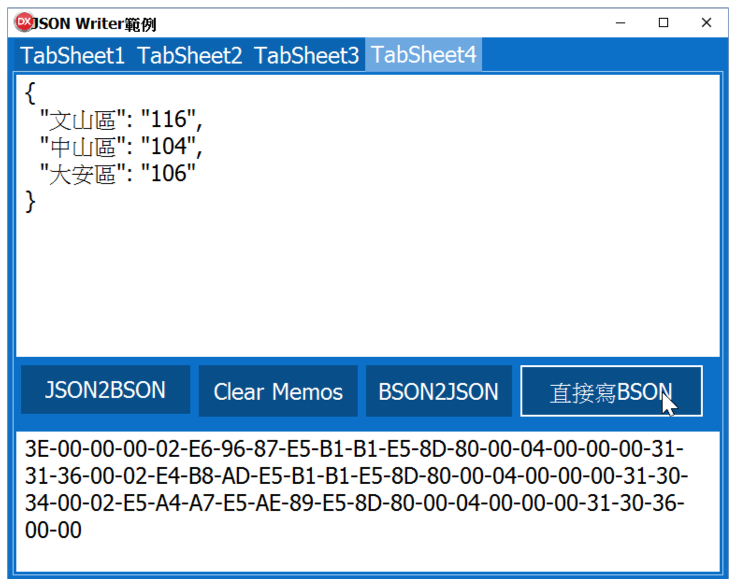
當然我們也可以直接把JSON格式的資料寫入TBsonWriter物件中即可自動轉成 BSON 格式，例如下面的程式碼直接呼叫 TBsonWriter 物件的各個 Write 方法把寫成 BSON 格式：

```
001   procedure TForm2.Button12Click(Sender: TObject);
```

```
002   var
003     BsonWriter: TBsonWriter;
004     Stream: TBytesStream;
005   begin
006     Stream := TBytesStream.Create;
007     BsonWriter := TBsonWriter.Create(Stream);
008     try
009       BsonWriter.WriteStartObject;
010       BsonWriter.WritePropertyName('文山區');
011       BsonWriter.WriteValue('116');
012       BsonWriter.WritePropertyName('中山區');
013       BsonWriter.WriteValue('104');
014       BsonWriter.WritePropertyName('大安區');
015       BsonWriter.WriteValue('106');
016       BsonWriter.WriteEndObject;
017       BsonWriter.Close;
018       Button11Click(Sender);
019       Stream.Position := 0;
020       Memo4.Lines.Text := Bson2Json(Stream.Bytes);
021       Memo5.Lines.Text :=
Bytes2String( Json2Bson(Memo4.Lines.Text) );
022     finally
023       BsonWriter.Free;
024       Stream.Free;
025     end;
026   end;
```

版權所有 請勿翻印

下面是藉由上面的程式碼把臺北市郵遞區號寫成 BSON 格式之後顯示出來再轉回 JSON 格式。直接呼叫 TBsonWriter 類別中的方法也可正確無誤的處理 JSON/BSON 格式的資料。



19-3-2 TBsonReader 類別

TBsonReader 類別在上一小節已經看到如何使用它了，也說明了它是從 TJsonReader 類別繼承下來，因此我們我可以像使用 TJsonReader 類別的方式來使用它，只是它是使用來讀取 BSON 格式資料的 Reader 類別。

TBsonReader 的建構元接受一個包含 BSON 資料的 TStream 類別物件：

版權所有 請勿翻印

```
constructor Create(const AStream: TStream; AReadAsRootValuesArray:
Boolean = False);
```

之後我們就只要把它丟給 TJsonWriter 類別物件，那麼 TJsonWriter 類別物件就會把 TBsonReader 物件中的 BSON 資料寫入 TJsonWriter 類別物件中。如果需要從頭再次讀取 BSON 格式資料，那麼只需要呼叫它的 Rewind 方法即可：

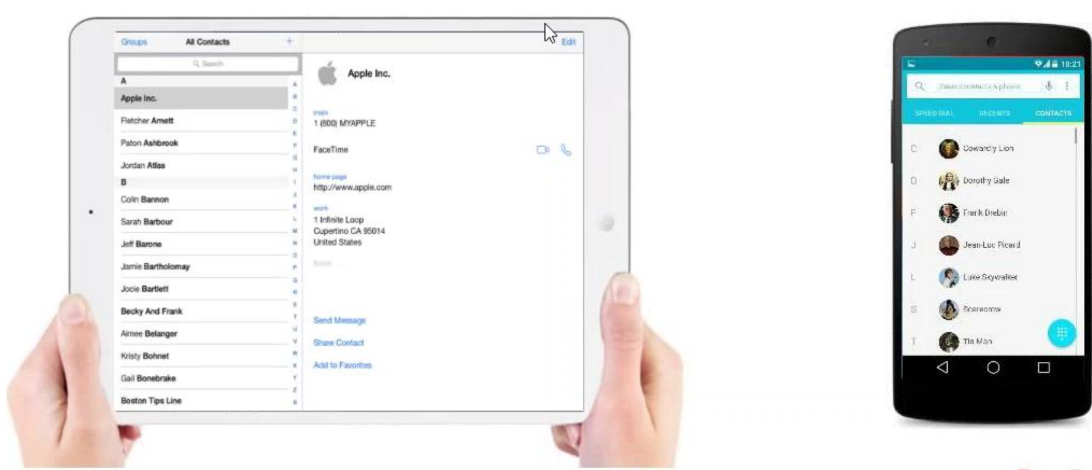
```
procedure Rewind; override;
```

由於上一小節已經介紹了如何使用 TBsonReader 類別，因此我們就不再贅述了。

20 新的 AddressBook 組件

XE11 終於提供了許多開發人員多年來的要求，那就是一個跨平臺的 TAddressBook 元件。

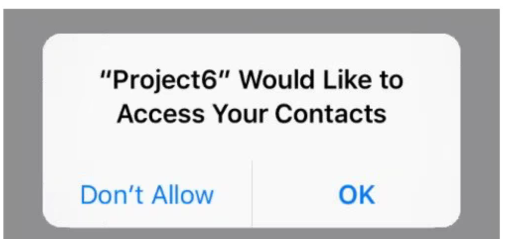
XE11 的新 TAddressBook 元件可以提供存取手機或平板的連絡資訊，並提供處理連絡資訊 CRUD 的能力，例如新增連絡人，新增連絡群組或是刪除連絡群組等功能:

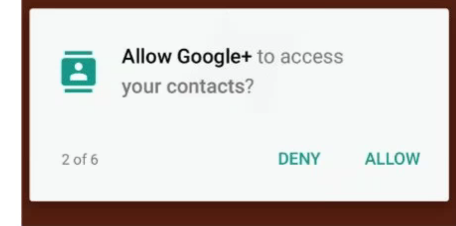


如此一來開發人員就可以使用 TAddressBook 元件在 iOS 或是 Android 進行相關工作的開發。

在使用 TAddressBook 元件存取連絡人資訊時開發人員需要設定存取權限否則無法存取連絡人資訊:

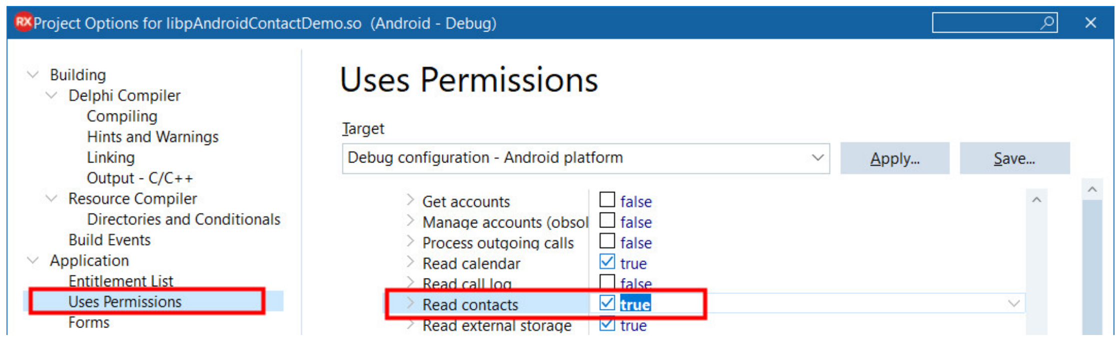
版權所有 請勿翻印





Permissions

在 Delphi 中程式師需要設定 Users Permissions 中的 Read contacts 為 true：



接著呼叫 TAddressBook 元件的 RequestPermission()方法要求存取連絡人的許可權，之後會觸發 TAddressBook 元件的 OnermissionRequest 事件:

```
procedure TfmMainForm.AddressBook1PermissionRequest(ASender:
TObject;
  const AMessage: string; const AAccessGranted: Boolean);
begin

end;
```

OnermissionRequest 事件的 AAccessGranted 參數即代表使用者是否授權存取連絡人資訊，true 代表授權而 false 則代表否決存取。

版權所有 請勿翻印

完成存取權限設定之後就可以呼叫 TAddressBook 元件的 AllContacts()方法取得代表所有連絡人資訊的 TAddressBookContacts 物件:

```
    function AllContacts: TAddressBookContacts;
```

AllContacts()方法回傳包含所有連絡人資訊的 TAddressBookContacts 物件：

```
TAddressBookContacts = class(TObjectList<TAddressBookContact>);
```

而其中每一個連絡人資訊則是由 TAddressBookContact 類別物件代表，請注意如下的 TAddressBookContact 類別宣告，TAddressBookContact 類別是一個抽象類別：

```
  TAddressBookContact = class abstract
```

這代表程式師不應該直接建立 TAddressBookContact 類別物件，而應該藉由 TAddressBook 元件的 AllContacts()方法取得。

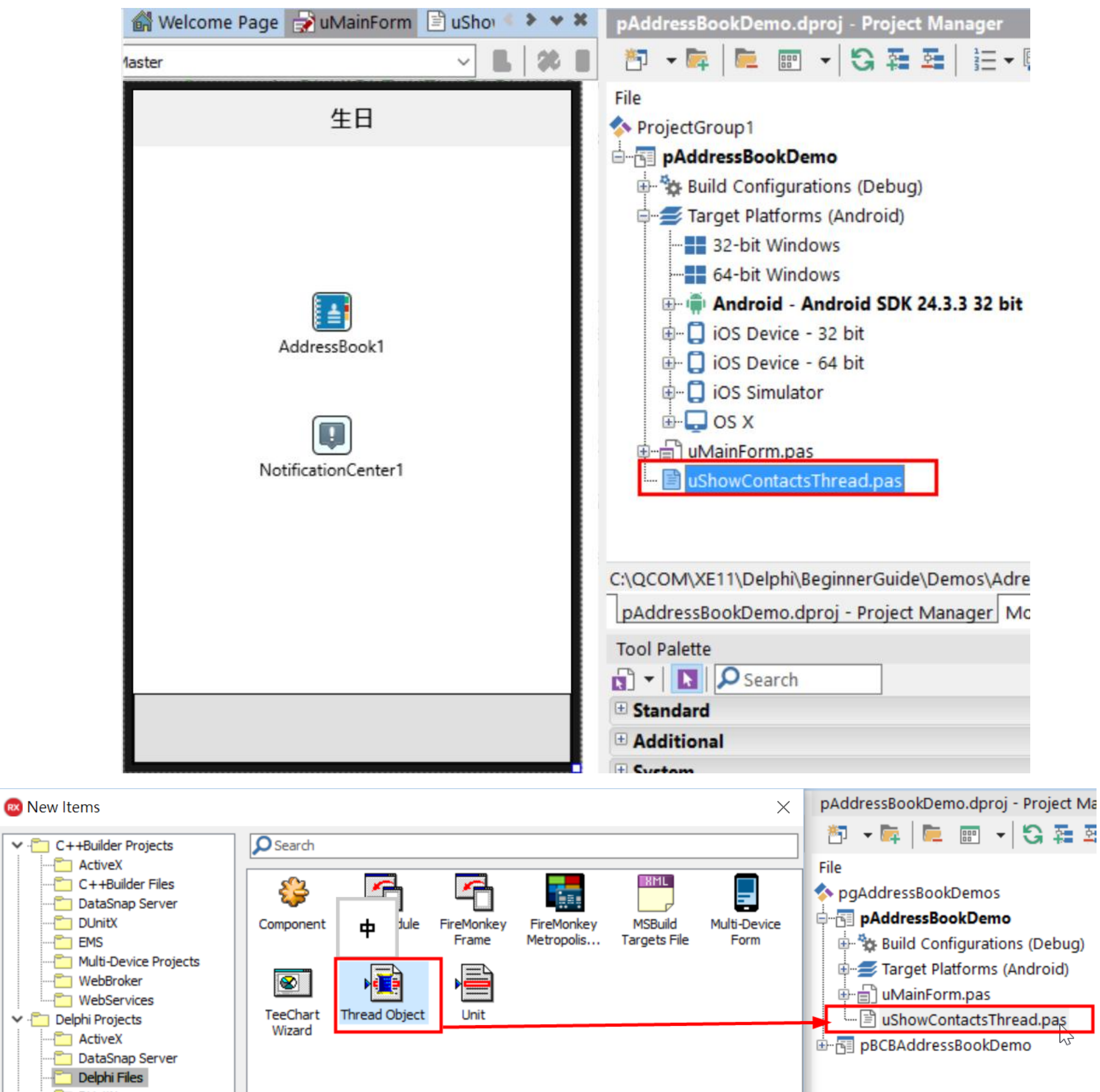
最後一旦取得了 TAddressBookContact 類別物件後，程式師就可以藉由存取它的特性值來取得連絡人資訊，例如連絡人名稱，連絡人 Email 和連絡人生日等重要的資訊：

```
. . .
property DisplayName: string read GetDisplayName;
property EMails: TMultipleStringValues index TContactField.EMails
read GetListStringValue write SetListStringValue;
property Birthday: TDateTime index TContactField.Birthday read
GetDateTimeValue write SetDateTimeValue;
. . .
```

另外在使用 TAddressBook 元件存取連絡人資訊時讀者應具備的一個重要觀念就是應該使用一個獨立的執行緒來存取而不要使用主程序的執行緒，這是因為連絡人資訊可能數量眾多而且手機中的連絡人資訊並不是儲存在 RDBMS 的資料庫中而是使用各自(iOS/Android)的架構儲存。因此在存取大量連絡人資訊時速度可能不夠快而拖累主程序的執行速度，而在 Delphi 中我們正好可使用一個額外的 TThread 類別物件來存取。

現在我們就可以說明如使用 TAddressBook 元件了，請先建立一個 Multi-Device 專案在主表單中加入新的 TAddressBook 元件，接著再建立一個 TThread 類別 TShowContactsThread，如下所示：

版權所有 請勿翻印



首先在主表單的 OnShow 事件中要求存取連絡人資訊的許可權:

```
procedure TfmMainForm.FormShow(Sender: TObject);
begin
  AddressBook1.RequestPermission;
end;
```

接著在 TAddressBook 元件的 OnPermissionRequest 事件中檢查是否取得許可權，如果是的話就呼叫 FillContactsInfo 方法存取所有連絡人資訊:

```
procedure TfmMainForm.AddressBook1PermissionRequest(ASender:
TObject;
  const AMessage: string; const AAccessGranted: Boolean);
begin
  if (AAccessGranted) then
```

```
    FillContactsInfo
  else
  begin
    lblProgressInfo.Text := '用戶不允許存取訊號!';
  end;
end;
```

FillContactsInfo 方法會建立前面建立的 **TTread** 衍生類別 **TShowContactsThread** 的物件並把使用 **TAddressBook** 元件取得的 **TAddressBookContacts** 物件傳遞給此執行緒物件，再啟動執行此獨立的執行緒：

```
procedure TfmMainForm.FillContactsInfo;
begin
  FreeAndNil(FContacts);
  FContacts := AddressBook1.AllContacts;
  if ( (Assigned(FThread)) and (not FThread.Finished)) then
  begin
    FThread.Terminate;
    FThread.WaitFor;
  end;
  FreeAndNil(FThread);

  FThread := TShowContactsThread.Create(FContacts);
  FThread.OnContactLoaded := ContactedLoaded;
  FThread.OnStart := ContactLoadingBegin;
  FThread.OnTerminate := ContactLoadingEnd;
  FThread.Start;
end;
```

版權所有 請勿翻印

TShowContactsThread 類別的建構元會儲存主執行緒傳遞來的 **TAddressBookContacts** 物件：

```
constructor TShowContactsThread.Create(const AllContacts:
TAddressBookContacts);
begin
  inherited Create(True);
  FAllContacts := AllContacts;
end;
```

接著在 Execute()方法中一一取出 TAddressBookContacts 物件每一個代表連絡人的 TAddressBookContact 物件，再呼叫 Synchronize()方法藉由匿 1 方法呼叫 DoContactedLoaded()方法顯示每一個連絡人資訊：

```
procedure TShowContactsThread.Execute;
var
  Contact : TAddressBookContact;
  iIndex : Integer;
  DisplayName : String;
  Photo : TBitmapSurface;
  BirthDay : TDateTime;
begin
  Synchronize(DoStart);
  iIndex := 0;
  for Contact in FAllContacts do
  begin
     DisplayName := Contact.DisplayName;
     Photo := Contact.Photo;
     Birthday := Contact.Birthday;
     try
       Synchronize(procedure
       begin
         DoContactedLoaded(iIndex, Birthday, DisplayName, Photo);
       end);
     finally
       if (Assigned(Photo)) then
         Photo.Free;
     end;
     Inc(iIndex);
  end;
end;
```

版權所有 請勿翻印

最後編譯且執行此範例程式到 iOS 或是 Android 手機中就可以看到如下的執行結果畫面，由於範例程式使用了一個獨立的執行緒存取和顯示連絡人資訊因此執行速度很理想也不會拖累主表單的反應速度：

版權所有 請勿翻印

21 安裝和設定 Linux 開發環境

Delphi RIO 版最主要的功能就是開始支持 Linux 的開發，而且主要是支援 Linux 伺服端的開發工作，例如 WebBroker，DataSnap，RAD Server 等。在 Delphi RIO 版支援 2 個 Linux 版：

- Red Hat 7.x,
- Ubuntu 16.04.x

本節將說明如何安裝和設定 Linux 和 Delphi IDE 的開發環境以便幫助讀者順利開發 Linux 伺服端應用程式。

在開始繼續往下閱讀之前讀者需要完成下列的準備工作：

軟體	說明
Red Hat	Delphi RIO 支持的 Linux 版本，可在此下載安裝：https://developers.redhat.com/blog/2016/03/31/no-cost-rhel-developer-subscription-now-available/
Ubuntu	Delphi RIO 支持的 Linux 版本，可在此下載安裝：

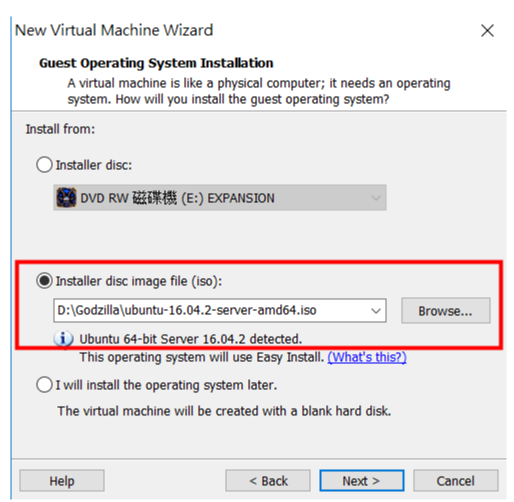
	https://www.ubuntu.com/download/server
FileZilla 或是 WinSCP	傳遞 LinuxPAServer19.0.tar 檔案到 Linux 並在 Linux 中安裝 PAServer，就和 Mac 平臺的 PAServer 一樣，在 Linux 安裝完 PAServer 後 Delphi IDE 就可以把編譯後的 Linux 應用程式自動傳遞和安裝到 Linux 環境 FileZilla: https://filezilla-project.org/download.php?type=client WinSCP: https://winscp.net/eng/download.php
VMWare	由於筆者是把 Linux 執行在虛擬機器中，因此也使用了 VMWare

21-1 安裝 Linux 作業系統

在本小節中將使用 Ubuntu 做為說明。

首先下載 ubuntu-16.04.2-server-amd64.iso 安裝檔，接著筆者使用 VMWare 從此 iso 檔建立 Ubuntu 的虛擬機器，我們只需要在 VMware 中選擇下載的 Ubuntu ISO 檔 VMWare 即可自動辨識出是 Ubuntu 64 作業系統：

版權所有 請勿翻印



用 VMware 安裝 Ubuntu 非常簡單，幾乎是全自動化。在安裝過程中讀者需要設定系統的，在本範例中使用的用戶名稱是 qcom，密碼是 qcom1234。

安裝完 Ubuntu 並使用用戶名稱和密碼登錄後，請在 Ubuntu 的命令列依序執行下麵的設定準備：

先更新您使用的 Ubuntu 作業系統：

```
sudo apt-get upgrade
```

在讀者使用 sudo 命令執行工作時會需要輸入系統管理員的密碼，例如安裝時使用的 qcom1234。

在安裝過程中 Ubuntu 會詢問您是否確定安裝，讀者只需輸入 y 即可。

接著再更新 Ubuntu 的分發檔案：

```
sudo apt-get dist-upgrade
```

接著安裝必要的 Ubuntu 解壓縮套件：

```
sudo apt-get install joe wget p7zip-full curl build-essential zlib1g-dev
libcurl4-gnutls-dev
```

成功完成上述的安裝工作和套件後，就可以執行下面的命令清理安裝時產生的暫存檔：

版權所有 請勿翻印

```
sudo apt-get autoremove
```

再執行下面的命令刪除安裝時產生的暫存檔：

```
sudo apt-get autoclean
```

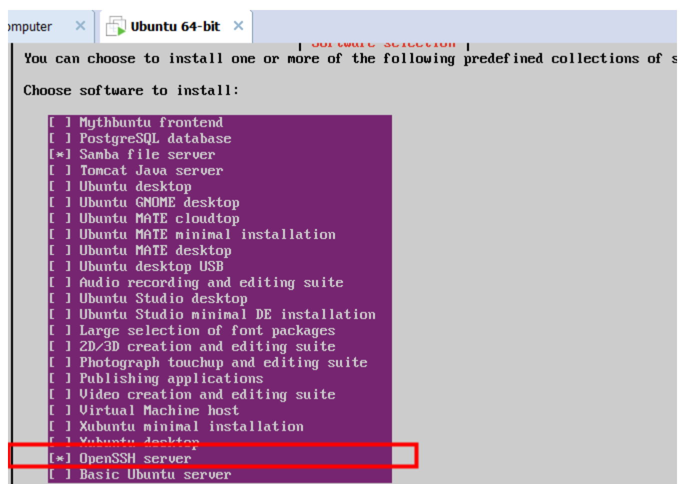
接下來讀者必須安裝 OpeSSH Server 以便把 Windows 中的 LinuxPAServer19.0.tar.gz 傳遞到 Ubuntu，讀者可以直接執行下面的命令安裝 OpenSSH Server：

```
sudo apt-get install openssh-server
```

或是執行：

```
sudo tasksel
```

啟動 Ubuntu 的安裝程式選擇安裝 OpenSSH Server，如下所示：



最後使用下面的命令可在 Ubuntu 中安裝 Apache：

```
sudo apt install apache2
```

現在 Ubuntu 的安裝和設定已經完成，接下來我們需要把 LinuxPAServer19.0.tar 從 Windows 傳遞到 Ubuntu 並且在 Ubuntu 中安裝 PAServer 19.0。首先在 Ubuntu 中執行

```
ifconfig
```

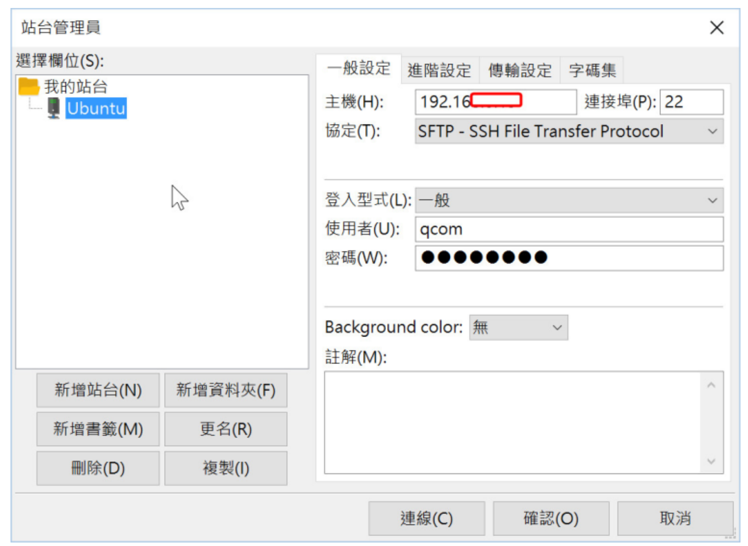
找到您的 Ubuntu 的 IP 地址：

版權所有 請勿翻印

```
qcom@ubuntu:~/PAServer/scratch-dir/Gordon-Ubuntu/LinuxConsole101$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:ff:1c:c7
          inet addr:192.16        Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feff:1cc7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30960 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5138 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:41490998 (41.4 MB)  TX bytes:512585 (512.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
```

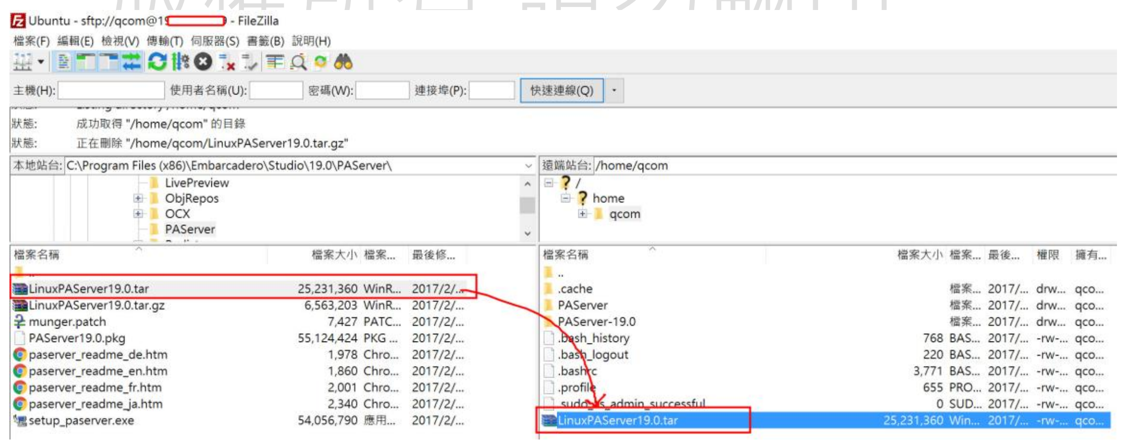
21-2 傳遞 LinuxPAServer19.0.tar 到 Linux

下載 FileZill 並建立和 Unbuntu 連結的資訊，再選擇使用 SFTP，下面的主機就是前面安裝完 Ubuntu 之後的 IP 位址：



成功連結之後就可以把 RAD Studio 安裝目錄中位於 \Embarcadero\Studio\19.0\PAServer\ 目錄下的 LinuxPAServer19.0.tar.gz 拖曳到 Ubuntu 機器中，FileZilla 就會把 LinuxPAServer19.0.tar.gz 傳遞到 Ubuntu，當然讀者也可以把從 LinuxPAServer19.0.tar.gz 壓縮開的 LinuxPAServer19.0.tar 傳到 Ubuntu，如下所示：

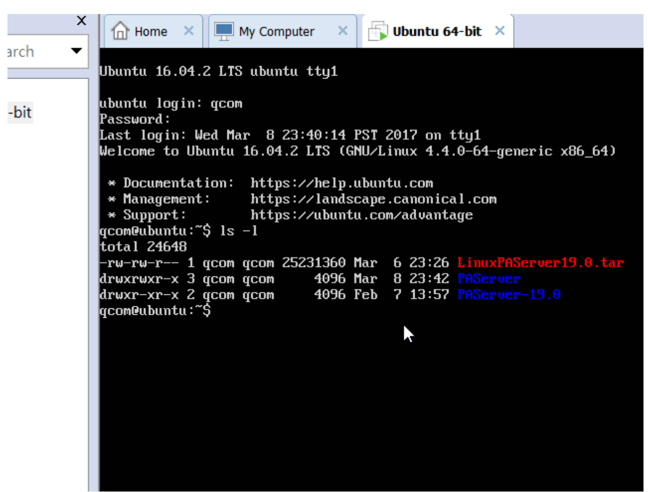
版權所有 請勿翻印



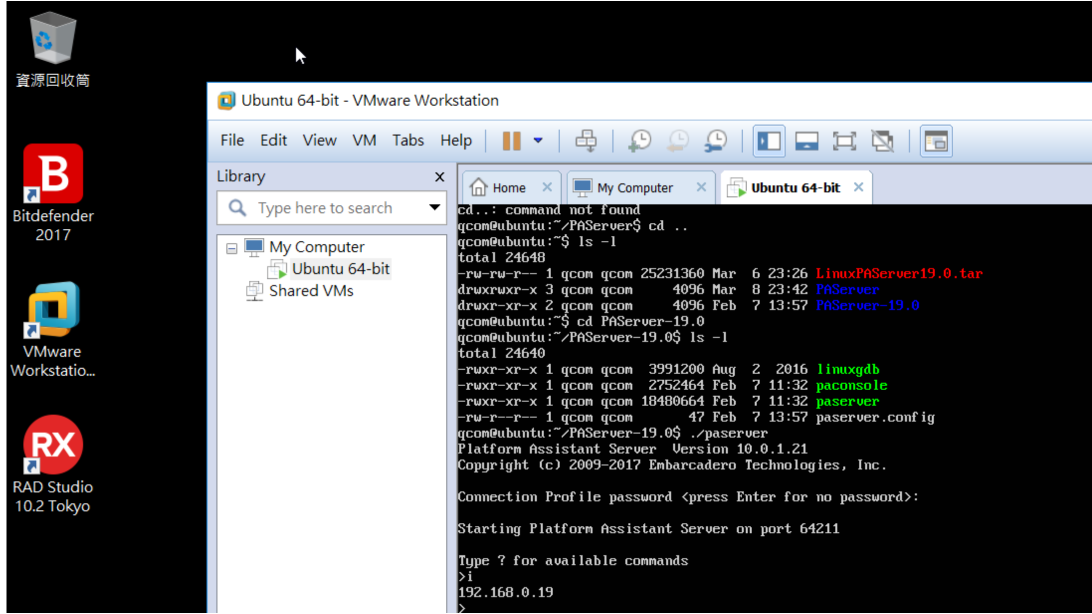
傳遞完成後回到 Ubuntu 並執行下面的命令安裝 PAServer 19.0：

```
tar xvf LinuxPAServer19.0.tar
```

使用 ls –l 即可看到 PAServer-19.0 目錄:



使用 cd PAServer-19.0 到目錄之下再執行./paserver 即可執行 PAServer 就如同前面說明 Mac 的 PAServer 一樣，在 PAServer 中輸入 i 即可回傳 Ubuntu 的 IP 位址:



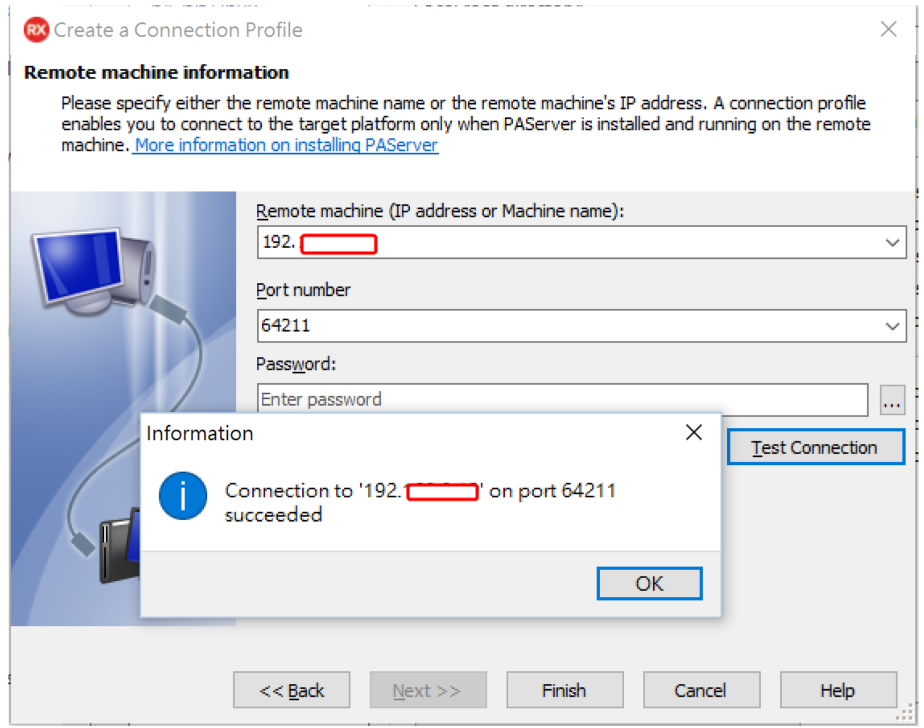
當 PAServer 19.0 成功在 Ubuntu 執行後就可以準備讓 Delphi IDE 連結了。

21-3 設定 Delphi IDE 連結 Linux 作業系統

接下來的工作就是在 IDE 中設定和 Ubuntu 的連結以便從 Ubuntu 中取得 Delphi 編譯 Linux 程式必要的表標頭檔和函式庫。請點選 Tools|Options 選單再於對話盒中左方選擇 SDK Manager 項目再點選下方的 Add...按鈕以便加入 Unbuntu 的 SDK 資訊。

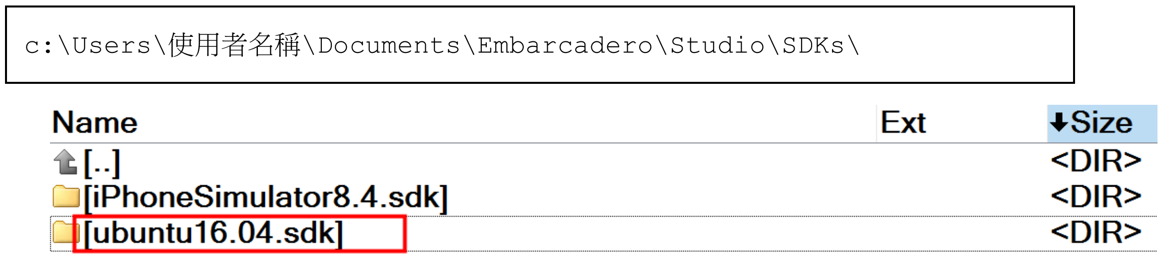
現在先確定 Unbuntu 中已經執行 PAServer：

接著在 Add a New SDK 對話盒中選擇 64-bit Linux，於 Select a profile to connect:欄位選擇 Add New...，再為此 Profile 取一名稱之後，如同前面章節說明的再輸入 Unbuntu 的 IP 位址:

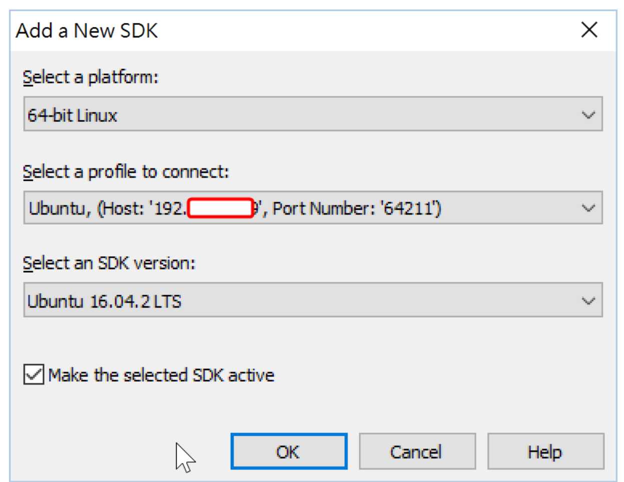


IDE 便會從 Unbuntu 拷貝表標頭檔和函式庫到 Windows 中並存於:

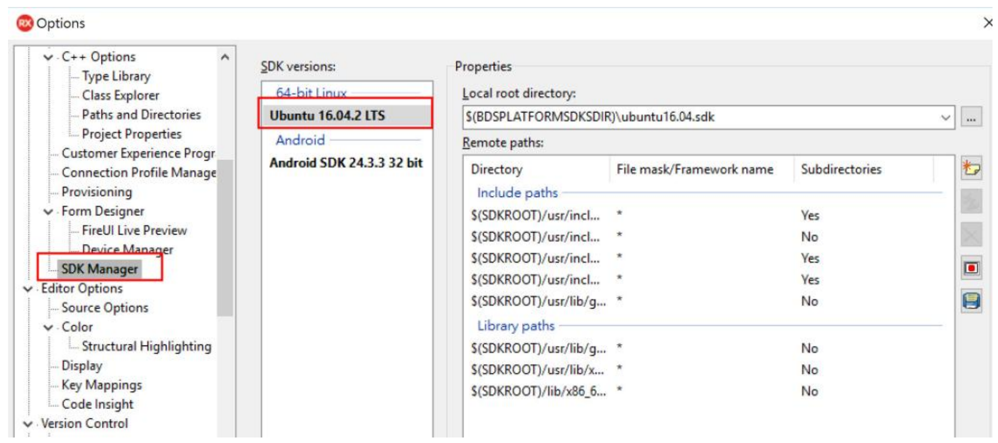
```
c:\Users\使用者名稱\Documents\Embarcadero\Studio\SDKs\
```



而且 IDE 會辨識出讀者使用的 Linux 版本:



最後在 SDK Manager 中便會出現 Ubuntu 的 Profile 資訊:

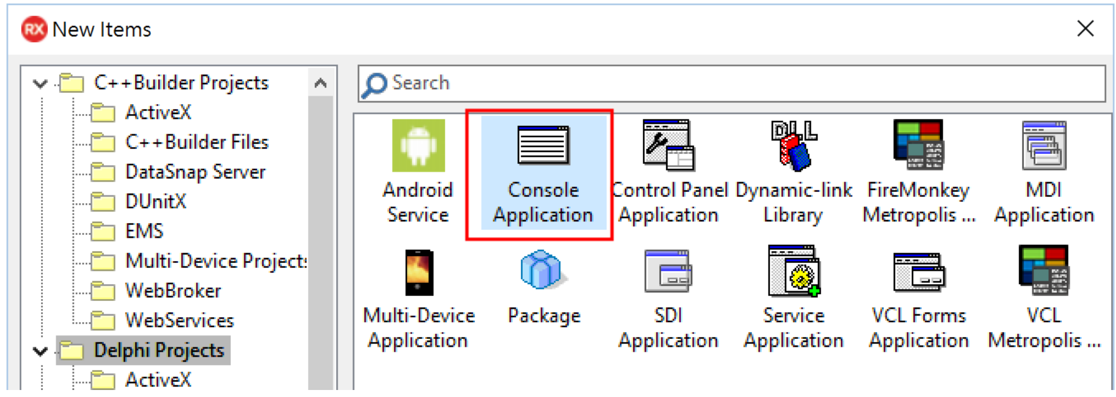


到此就成功設定完成了。

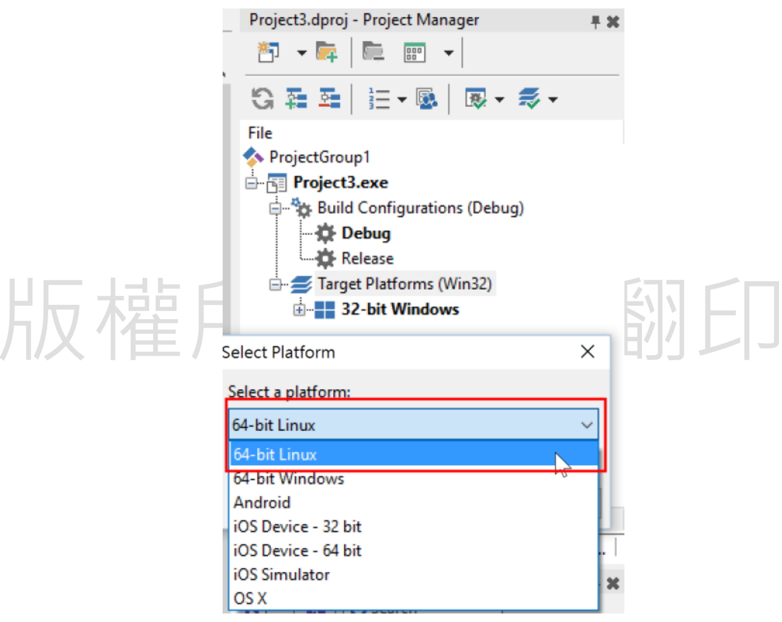
21-4 開發 Delphi Linux 應用程式並部署到 Linux

由於 RIO 版的 Delphi 只支援開發 Linux 伺服器應用程式，例如 DataSnap，Webbroker，RAD Server 套件等，以及文字介面的 Linux 應用程式，因此我們先以如何開發一個文字介面的 Linux 應用程式說明如何開發，部署最終在 Ubuntu 中執行此 Delphi 應用程式。

首先選擇建立 Console Application 項目：

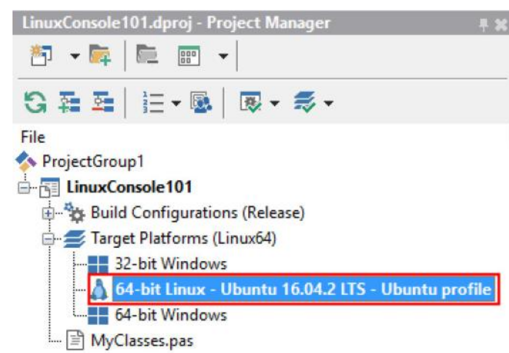


在建立項目之後可到項目管理員中加擊 Target Platforms 就可以看到 64-bit Linux 平臺：

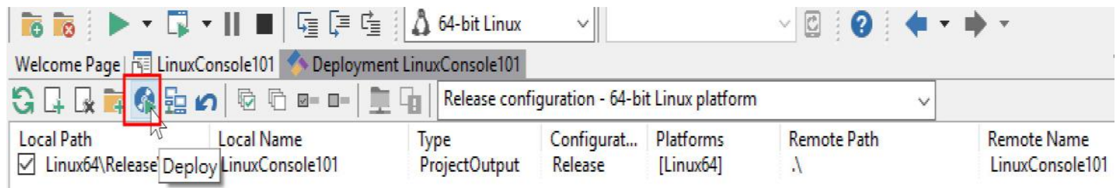


版權 翻印

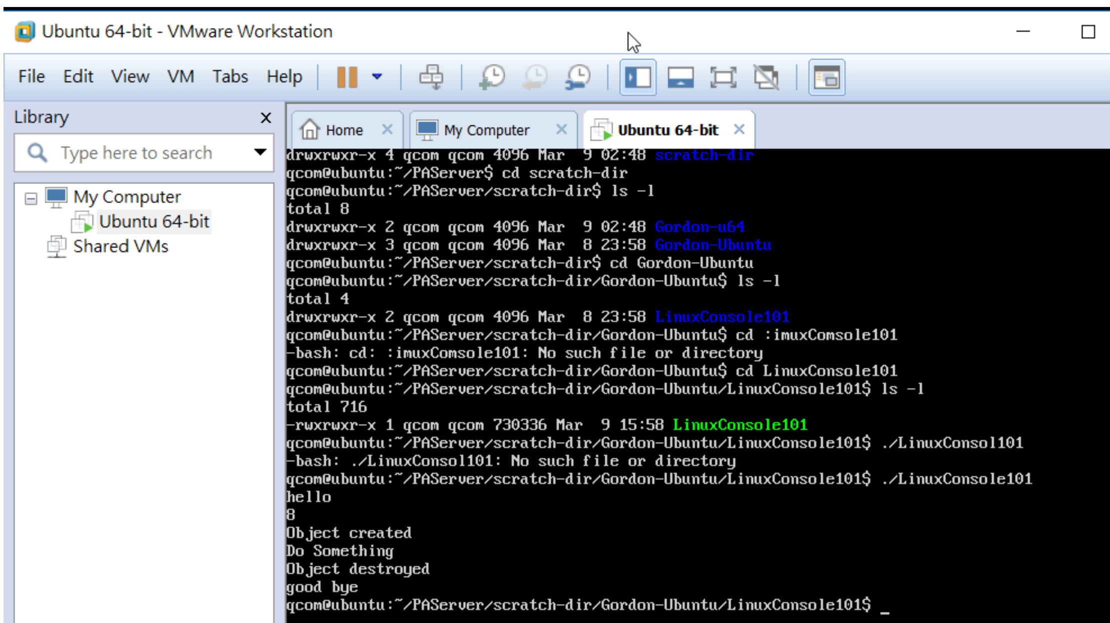
選擇 64-bit Linux 平臺後就可以在項目管理員中看到您的應用程式支援 Ubuntu 了：



從這裡起讀者就可以像開發 Windows 的應用程一樣開發 Linux 的應用程式了，一旦在 Windows 開發完成之後如果要部署到 Ubuntu，就可以點選 Project|Deployment 選單再點選上方的 Deploy 按鈕：

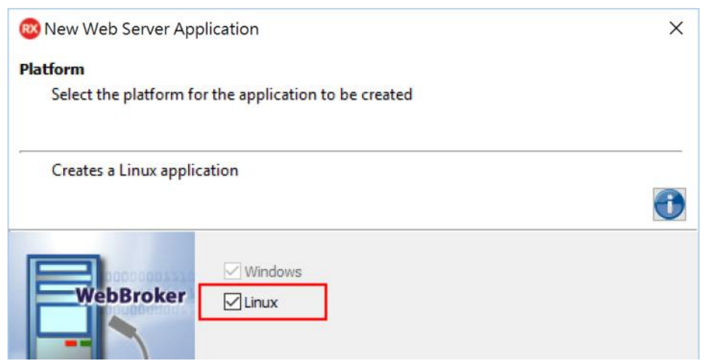


部署到 Ubuntu 的 Delphi 應用程式是部署到 PAServer/scratch-dir/XXXXX-Ubuntu 目錄下，例如前面的範例專案 LinuxConsole101.dproj 就部署在 PAServer/scratch-dir/XXXXX-Ubuntu/LinuxConsole101 目錄下：



要執行部署的 Delphi 應用程式，只需要轉到 Delphi 應用程式名稱目錄下，再使用./ Delphi 應用程式名稱即可，例如要在 Ubuntu 中執行前面的範例專案，只需要執行./ LinuxConsole101 即可。

因此只要是屬於伺服器的開發，例如 DataSnap，Webbroker 和 RAD Server 的開發就和 Windows 開發一樣，例如要開發 Webbroker 應用程式，在 RIO 版中就增加了 Linux 選項：

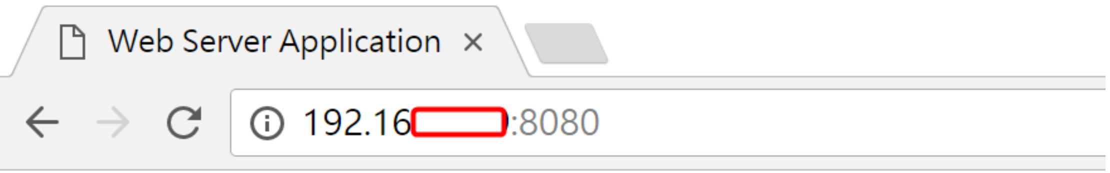


和以前一樣使用 Action Editor 撰寫程式碼，再部署到 Ubuntu 之後，於 Ubuntu 執行 Webbroker 應用程式：

```
procedure TWebModule1.WebModule1DefaultHandlerAction(Sender:
TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled:
Boolean);
begin
  Response.Content :=
    '<html>' +
    '<head><title>Web Server Application</title></head>' +
    '<body>嗨, Delphi For Ubuntu</body>' +
    '</html>';
end;
```

版權所有 請勿翻印

例如下圖就是筆者在 Windows 中使用 Chrome 請求剛才開發並部署到 Ubuntu 的應用程式：



嗨, Delphi For Ubuntu

最後一點提醒的是，如果讀者要安裝 Interbase for Linux 以便使用 RAD Server For Linux 的話，那麼您需要執行下面的步驟以便順利安成 Interbase for Linux：

1. 把 Interbase for Linux 解壓縮並用 FileZilla 傳遞到 Linux

2. 在 Ubuntu 使用下面的指令安裝 tofrodos:

```
sudo apt-get install tofrodos
```

再使用下面的指令把 Windows 的換行字元轉為 Linux 的形式

```
fromdos install_linux64.sh
```

3. 使用

```
sudo tasksel
```

安裝 xubuntu.

4. 重新開機 Ubuntu 進入 Ubuntu GUI 介面，開啟 Terminal Emulator

5. 轉換目錄到 Interbase For Linux 的解壓縮目錄，再使用

```
sudo ./install_linux64.sh
```

才能順利安裝 Interbase For Linux，也請參考下面的 URL：

```
https://community.embarcadero.com/blogs/entry/installing-interbase-xe
7-on-linux
```

本書到此也告一段落了，最後祝您使用 Delphi RIO 開發 iOS/Android App 和 Linux 應用程式愉快，現在您可以試著把前面 iOS 的捏泡泡 iOS App 轉換成 Android 的版本，Have Fun！

embarcadero
授權代理
捷康科技股份有限公司
電話: 02-23650238
傳真: 02-23650196
信箱： sales@qcomgroup.com.tw
http://embarcadero.qcomgroup.com.tw
版 權 所 有 · 請 勿 翻 印