
C++Builder
程式開發手冊
embarcadero

序

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

本書分成 2 大部份，第一部份將說明如何使用開發 iOS 設備上的 App，這部份也將說明如何使用的 IDE。

本書將討論如下的內容:

- 如何安裝和設定 C++Builder for iOS 整合發展環境
- 使用 C++Builder for iOS 整合發展環境開發您的第一個 iOS App
- 使用 C++Builder 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
- 除錯，測試和部署您的 Android App
- 開發一個有趣的 iOS App 吧

➢ 第 2 部份

- 如何安裝和設定 C++Builderfor Android 整合發展環境
- 使用 C++Builder 整合發展環境開發您的第一個 Android App

- 除錯，測試和部署您的 Android App
- 移植有趣的 iOS App 到 Android 中
- 結合 Android App 和雲端功能

在閱讀完本書的內容之後您就可以使用 C++Builder 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 C++Builder 做為學習和開發的工具。

在本書隨後的內容中將以 C++Builderfor iOS 來代表中開發 iOS App 的功能，並以 C++Builderfor Android 代表中開發 Android App 的功能。

目錄

1.安裝和設定 C++Builder for iOS..... 10

1-1 安裝 PAServer..... 11

1-2 C++Builder for iOS 整合發展環境中建立組態 14

2. 進入 C++Builder for iOS 整合發展環境..... 21

2-1 如何建立 iOS App 新專案..... 23

2-2 C++Builder for iOS 專案組成元素..... 26

3 開發您的第 1 個 iOS App..... 31

3-1 使用 MDD 設定..... 39

4. 使用 C++Builder for iOS 整合發展環境..... 41

4-1 移動程式碼區塊..... 55

4-2 儲存/切換桌面設定..... 56

4-3 原始碼格式化..... 63

4-4 SyncEdit..... 74

4-5 待辦清單(To-Do List)..... 77

4-6 程式區塊註解..... 81

4-7 程式碼瀏覽..... 81

4-8 設定和使用書籤..... 82

4-9 歷程管理員(History) 83

4-10 使用 LiveBinding 繫結資料和視覺化元件 86

4-11 實作 SearchEditButton1Click 事件處理函式 88

4-12 開啟多執行程序編譯功能 90

4-13 Visual Assist 功能 93

5 除錯您的 iOS App 98

6 開發和分發 iOS App 到 iOS 設備中 106

確定安裝了 XCode 的命令列工具 106

建立 iOS 設備的遠端組態 107

開發 iPad Mini 範例 App 111

取得 Apple 認證 116

使用部署管理員分發您的 iOS App 127

7 分發複雜的 iOS App 128

8 用 C++Builder 寫個遊戲吧 132

8-1 讓泡泡充滿畫面吧 133

8-2 點選捏破泡泡 137

8-3 加入手勢功能 139

8-4 重新開始遊戲 149

8-5 處理遊戲資訊 150

8-6 把遊戲資訊儲存到資料庫吧 157

8-7 分享遊戲的樂趣吧 169

9.安裝和設定 BCB for Android 開發環境 179

接下來我們就可以開始開發 Android App 了。10.開發 C++Builder for Android App 183

10-1 開發查詢臺北市旅館 App 184

10-2 加入查詢旅館功能 191

10-3 為您的 App 設計個圖像吧 196

10-4 為您的 App 加入啟動畫面吧 198

10-5 為您的 App 加入 Provisioning 資訊吧 200

設定 Build Configurations 為 Release 201

開啟 Target Platforms 設定為 Android 平台並在 Configuration 中選擇 Application Store..... 201

到 Project > Options > Provisioning 頁面填寫必要的資訊..... 202

維護 App 版本資訊 204

11 新功能 206

11-1 MultiView 206

11-2 使用分散式版本控制工具-Git..... 207

11-3 使用 DUNITX 單元測試框架 226

11-3-1 DUNITX 單元測試框架簡介 227

11-3-2 如何成為 DUNITX 測試框架的測試類別 228

規則 1 任何的 C++Builder 類別都可以成為測試類別 228

規則 2 撰寫測試方法 229

規則 3 如何使用 Test Fixture 229

規則 4 使用 Dunitx::Testframework::Assert 類別測試執行結果 230

11-3-3 使用 DUNITX 單元測試框架..... 231

11-4 App Tethering 234
11-4-1 手機端 TetherDBClient 如何工作..... 236
偵測和連結 237
11-4-2 Windows 端 TetherDatabase 如何工作 240
11-4-3 Windows 端和手機如何共同工作..... 241
TetherDBClient 專案端..... 242
TetherDatabase 專案端 242
TetherDBClient 專案端..... 244
TetherDatabase 專案端 244
11-5 藍牙開發 246
使用 TBlueTooth 元件..... 246
12 呼叫 Android 系統功能..... 264
12-1 呼叫 Java 類別程式碼..... 266
步驟 1 把 Java 類別宣告轉成 C++Builder 類別宣告 268
步驟 2 把 C++Builder 類別宣告直接轉成 C/C++的表頭宣告 269
步驟 3 編譯成.O 的檔案並直接連結到最後的 App 中 273
步驟 4 加入 Java 的.jar 檔並部署..... 274
12-2 呼叫 Java API 存取 Android 連絡人資訊 276
12-3 使用 Google GCM 實作推播功能 291
12-3-1 啟動使用 GCM 功能 292
12-3-2 開發 GCM 客戶端 App..... 296
建立 Multi-Device 專案 296
12-3-4 向 DataSnap 伺服器註冊設備 ID..... 305

12-3-5 開發 Windows 客戶端 311
13 物聯網開發 313
13-1 什麼是 Beancon 技術 313
13-2 Beancon 種類 314
13-3 Beancon 資料格式和意義 315
13-4 Beancon 接近狀態 317
13-5 Beancon 設備校正 317
13-6 開發 Beacon App 和物聯網應用架構 318
13-6-1 自動偵測 Beacon 設備 319
13-6-2 開發已知 Beacon 設備 App 326
14 開發有趣的物聯網應用架構 332
14-1 範例資料表 333
14-2 中介 DataSnap 伺服器 334
14-3 移動客戶端 340
14-4 執行 DataSnap 伺服器和客戶端 App 342
15 新的 JSON 類別庫 344
15-1 JSON Reader/Writer 類別 345
15-1-1 Writer 類別 346
15-1-2 Reader 類別 350
15-2 JSON Builder 類別 356
TJSONObjectBuilder 類別 357
TJSONArrayBuilder 類別 360
TJSONIterator 類別 363

15-3 BSON 類別 365

15-3-1 TBsonWriter 365

15-3-2 TBsonReader 類別 370

16 新的 AddressBook 元件() 371

C++Builder for iOS入門指引手冊

本手冊的目的是幫助 C++Builder for iOS 的入門使用者快速學習和瞭解如何使用 C++Builder 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 C++Builder for iOS 整合發展環境中經常會被使用的功能，讓 C++Builder 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 C++Builder for iOS 整合發展環境來開發實際的手機 App。

1.安裝和設定 C++Builder for iOS

在使用 C++Builder 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 C++Builder 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。

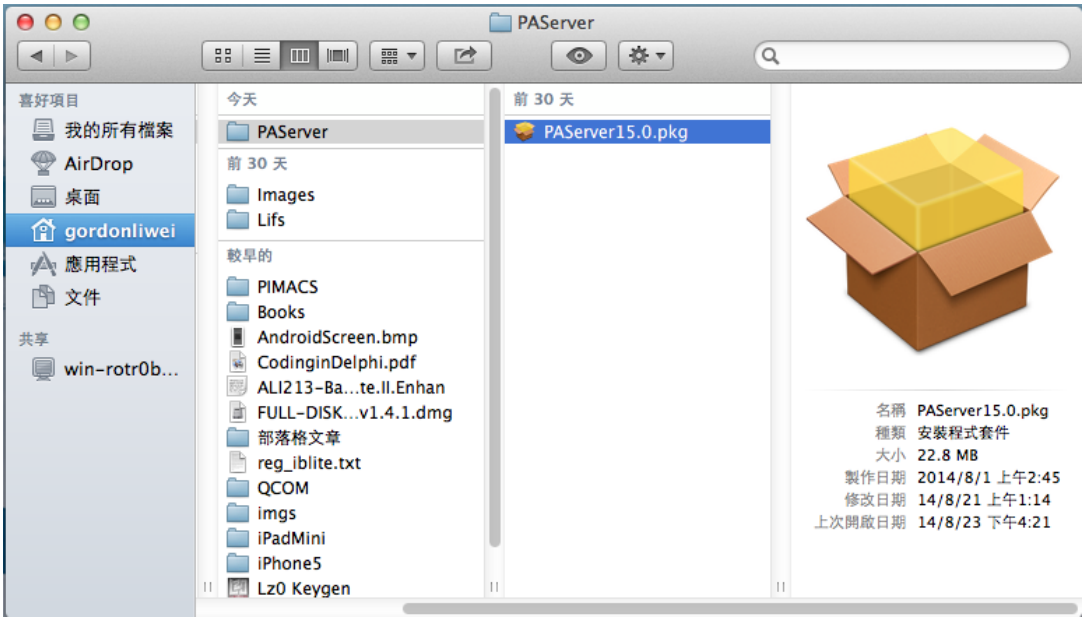
要完成 C++Builder for iOS 的安裝，讀者需要完成下面的步驟:

1. 安裝 C++Builder 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

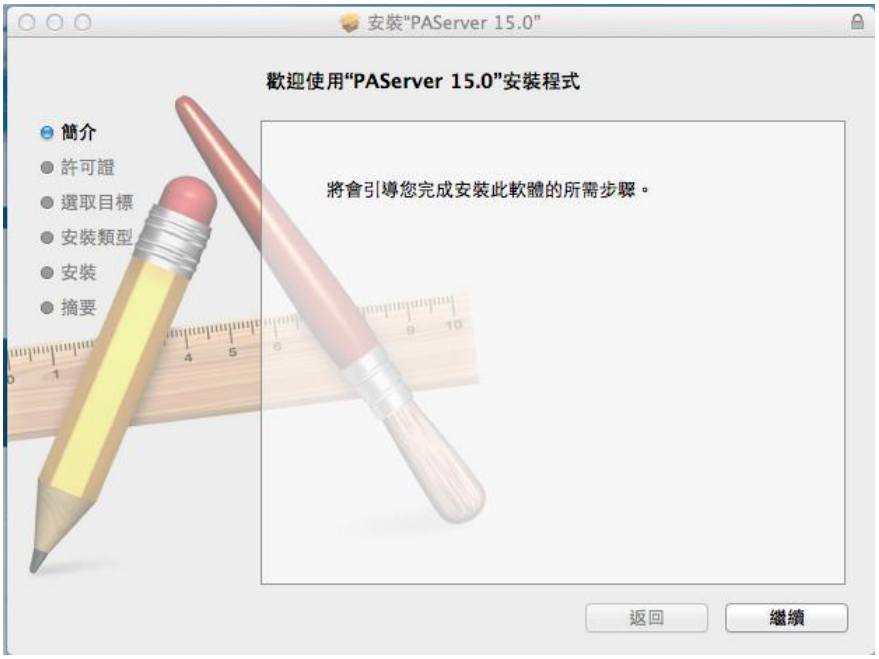
3. 在 C++Builder for iOS 整合發展環境中建立相關的組態，例如建立 iOS 模擬器組態以便除錯 iOS App，建立 iOS Device 組態以便部署和分發 iOS App 到 iPhone 或是 iPad/iPad Mini 設備中。

1-1 安裝 PAServer

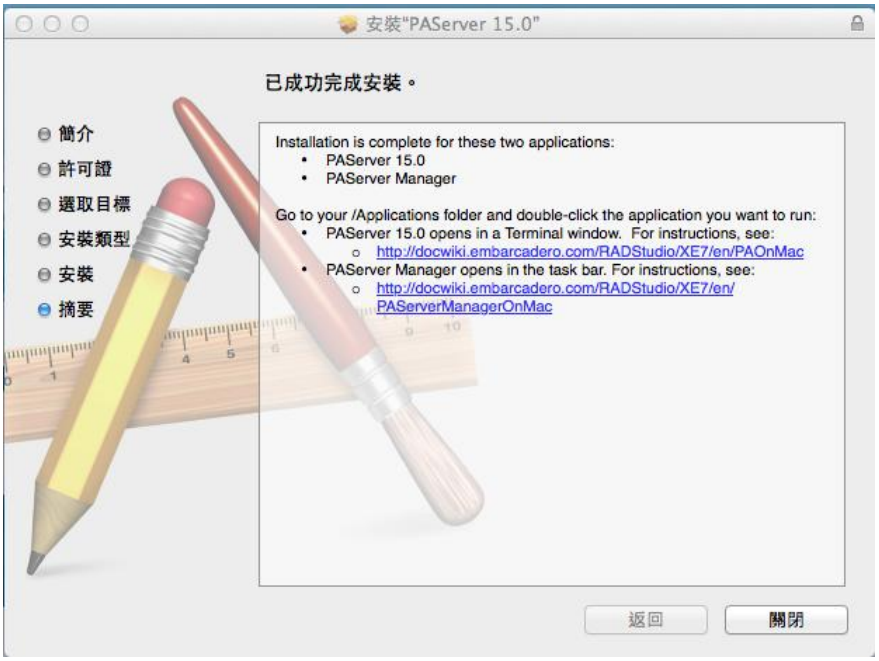
請使用 C++Builder for iOS 的安裝 DVD 安裝 C++Builder for iOS 整合發展環境，在安裝安之後，於 C++Builder for iOS 的安裝目錄下會有一個『PAServer』子目錄，其中的『PAServer15.0.pkg』檔案即包含需要安裝在 Mac OS 主作業系統的 PAServer 應用程式，請把此檔案拷貝到 Mac OS 中。例如下圖顯示筆者把『PAServer15.0.pkg』拷貝到 Mac 的 SharediOS\PAServer 目錄下:



在 Mac 中雙擊『PAServer15.0.pkg』便會開發安裝 PAServer:



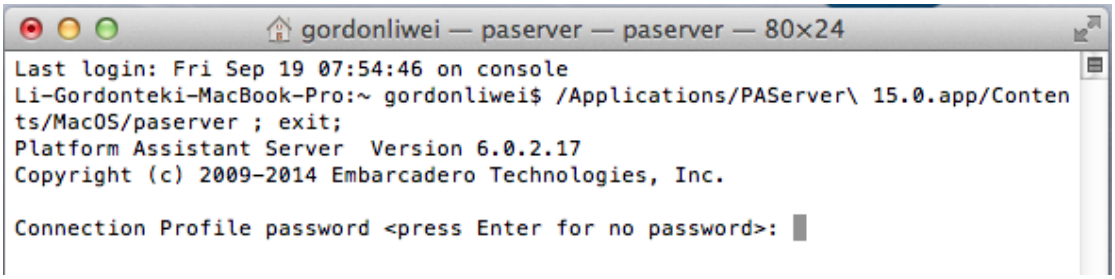
『 PAServer15.0.pkg 』 執 行 完 畢 之 後 會 把 PAServer 安 裝 在 『 /Applications 』目錄中。



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 ，如下所示:

或是在 Mac 中點選螢幕下方的 RAD PAServer 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:



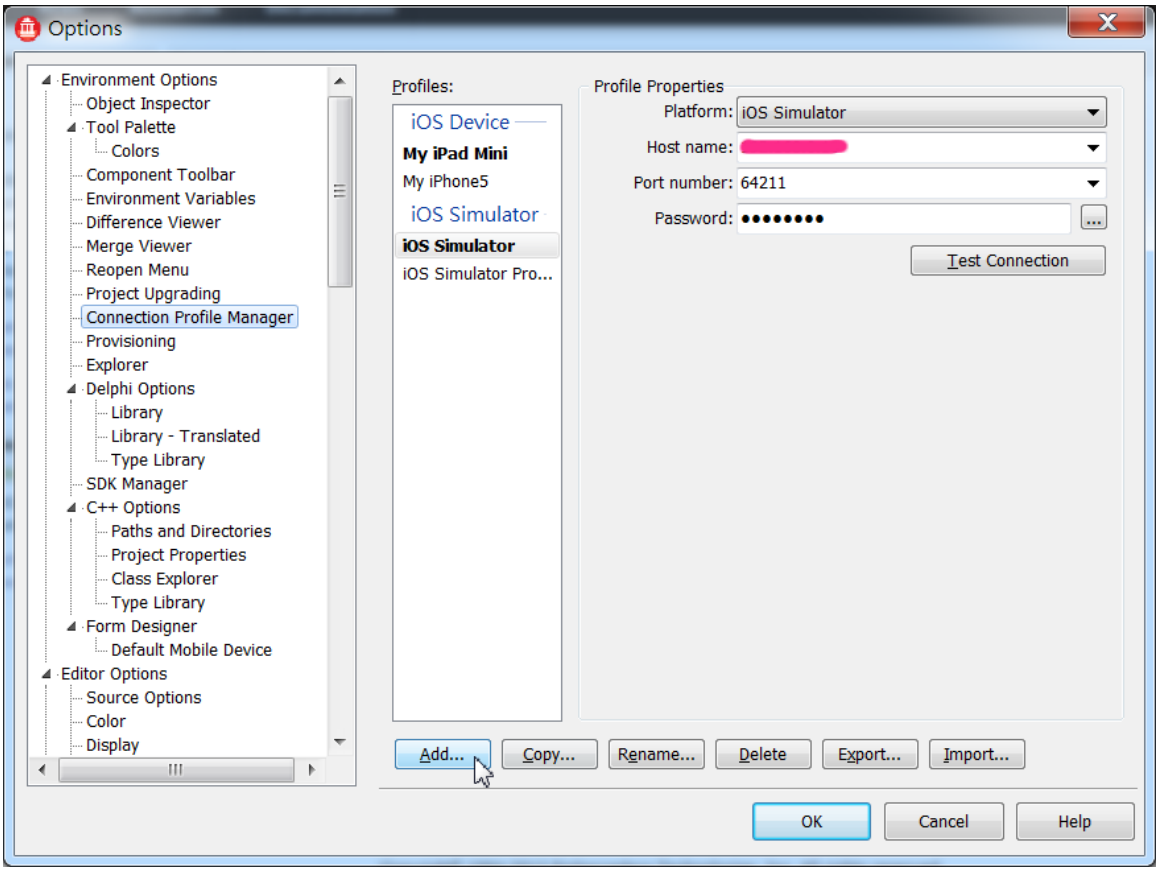
接著 PAServer 就會在執行狀態，例如下圖就顯示了 PAServer 執行在 64211 通信埠，等待連結：

本書在撰寫時是使用 C++Builder for 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 C++Builder for iOS 整合發展環境中建立組態

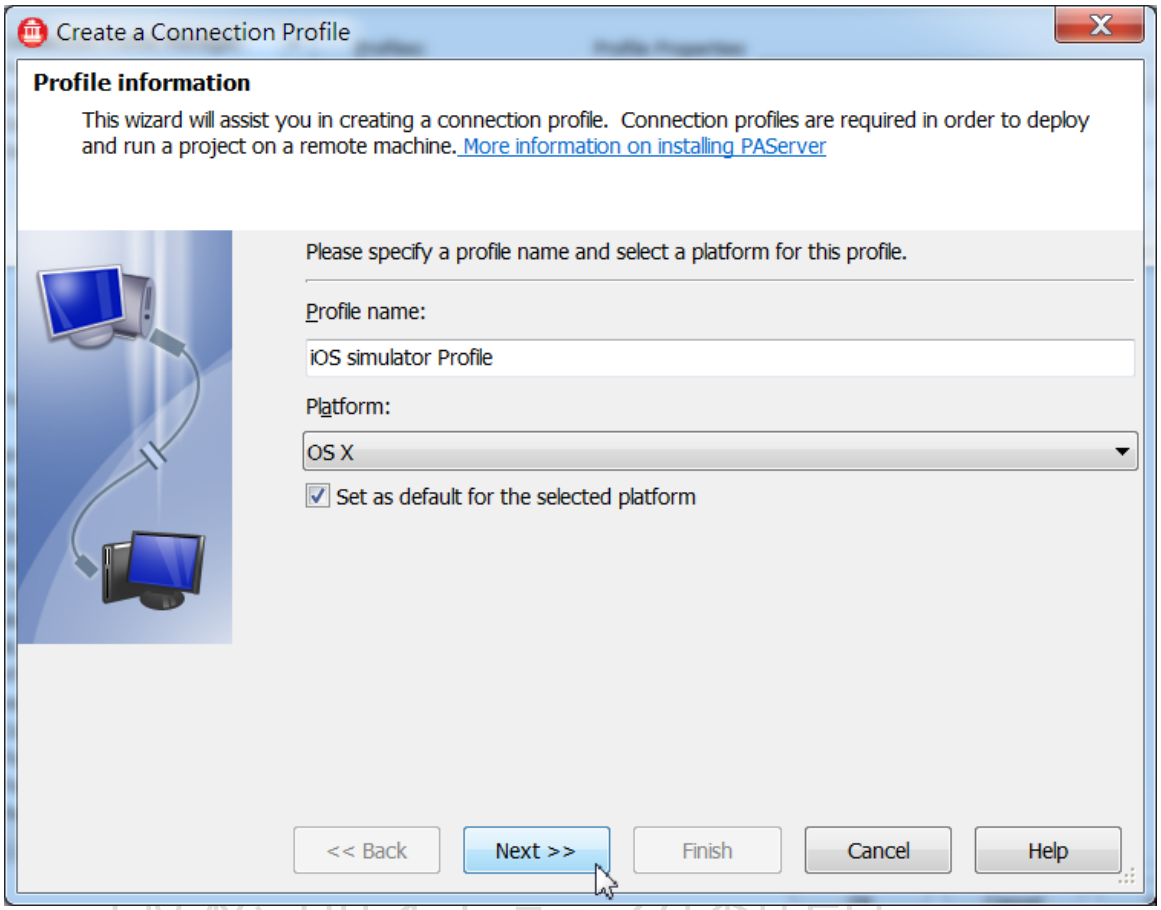
當開發人員在 C++Builder for iOS 中建立 iOS App 專案後，需要指定如何部署和分發此專案的組態。例如如果您需要除錯 iOS App 專案，那麼 C++Builder for iOS 整合發展環境必須把此 iOS App 分發到 Mac OS 中 XCode 的 iOS 模擬器中執行和除錯，因此 C++Builder for iOS 整合發展環境必須知道您的 Mac OS 的相關資訊才能夠正確的把 iOS App 部署到 iOS 模擬器中，因此這就需要開發人員事先設定好組態資訊並且把它指定給 iOS App 專案。

要建立組態資訊，請執行 C++Builder 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 項目中點選左下方的『Add』按鈕以新建組態，如下所示:

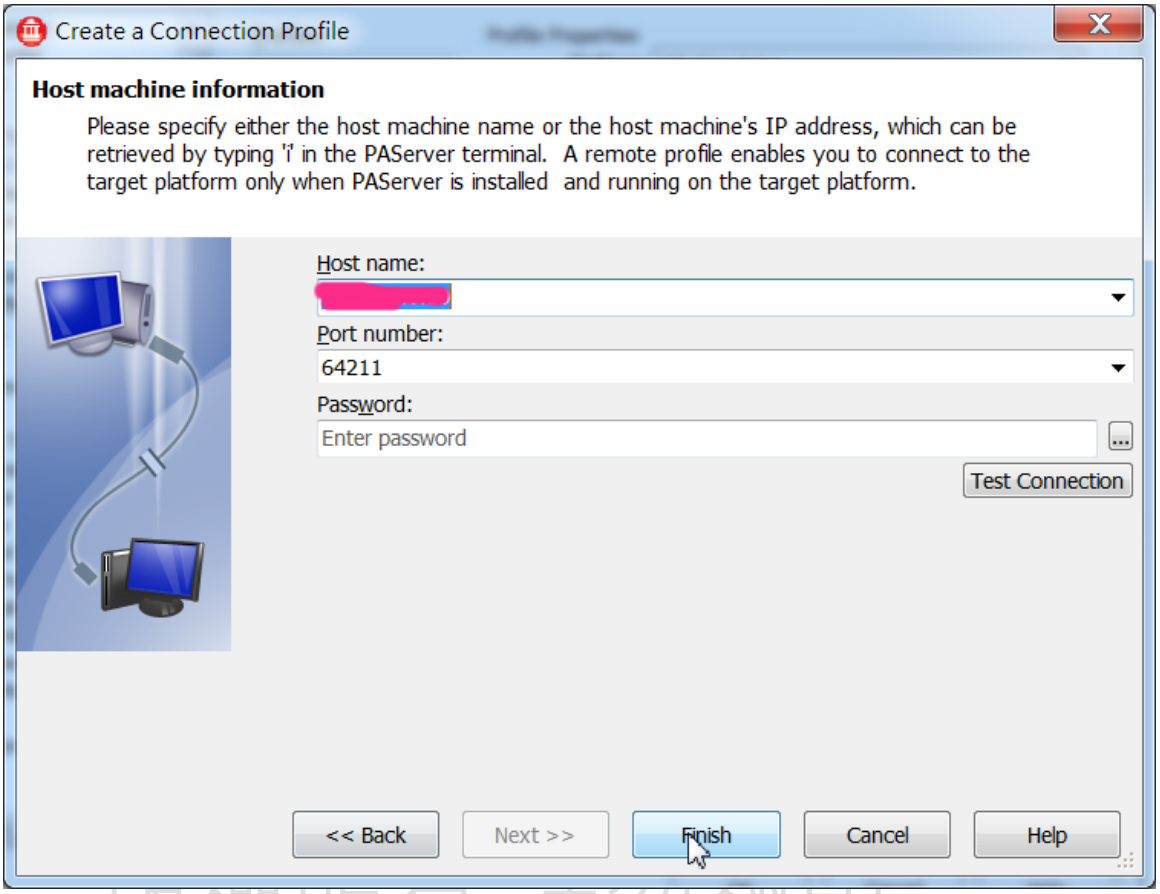


在點選『Add』按鈕後 IDE 會顯示 Create a Remote Profile 對話盒，您需要在此對話盒中輸入您需要在 Profile name 欄位中自行輸入此組態的名稱，接著在 Platform 欄位中選擇此組態需要部署的平台，例如在下圖中筆者為此組態取名為『iOS Simulator Profile』並且選擇部署到 iOS 模擬器中以準備開發和測試 iOS App。在 Platform 欄位的下拉盒中還有其他的平台，例如 iOS Device 平台，讀者可以根據需要選擇欲使用的平台。目前的範例中讓我們使用『OS X』平台，並且勾選下方的『Set as default for the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

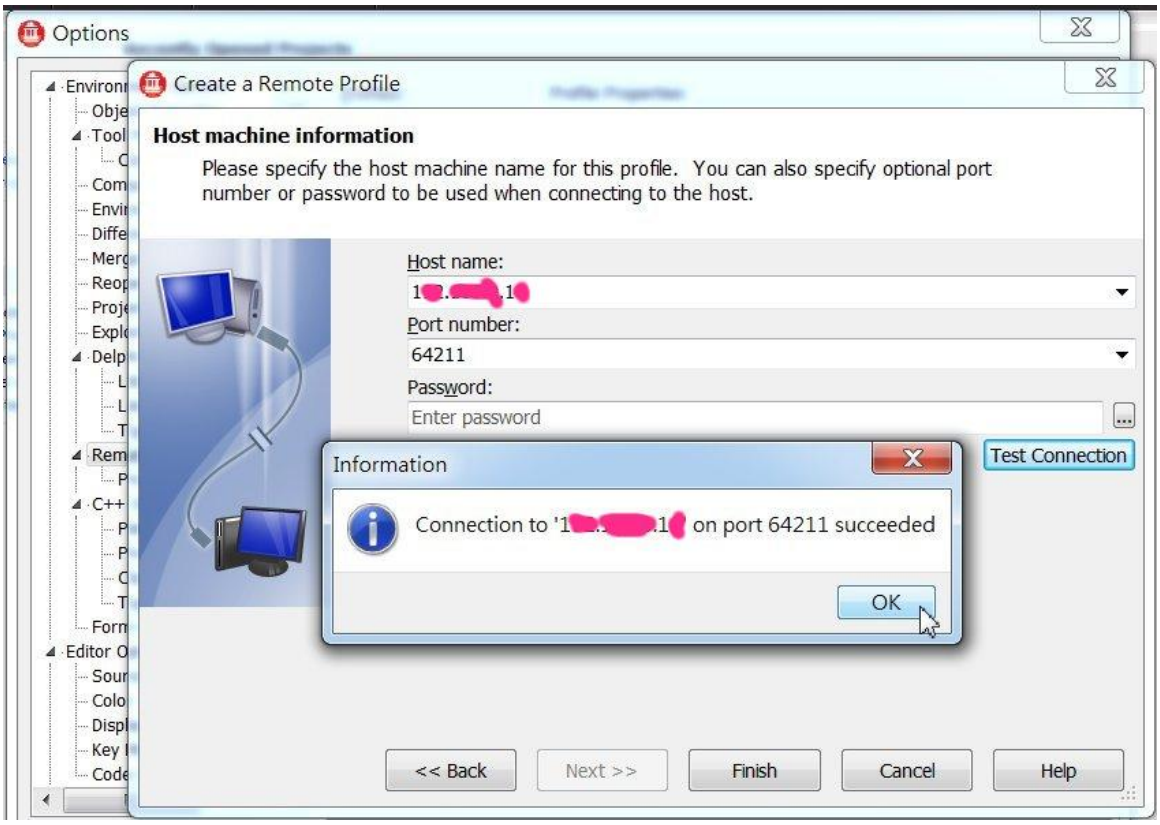


要取得您 Mac OS 的 IP 位址，您可以在您的 Mac OS 中點選螢幕左上方的蘋果圖像，選擇其中的『系統偏好設定』，再點選其中的『網路』圖像，接著 Mac OS 就會顯示如下的網路對話盒，其中就會顯示 IP 位址，例如筆者的 Mac IP 是『1xx.1xx.0.10』，因此在上圖的 Host Name 欄位中就輸入了此 IP。

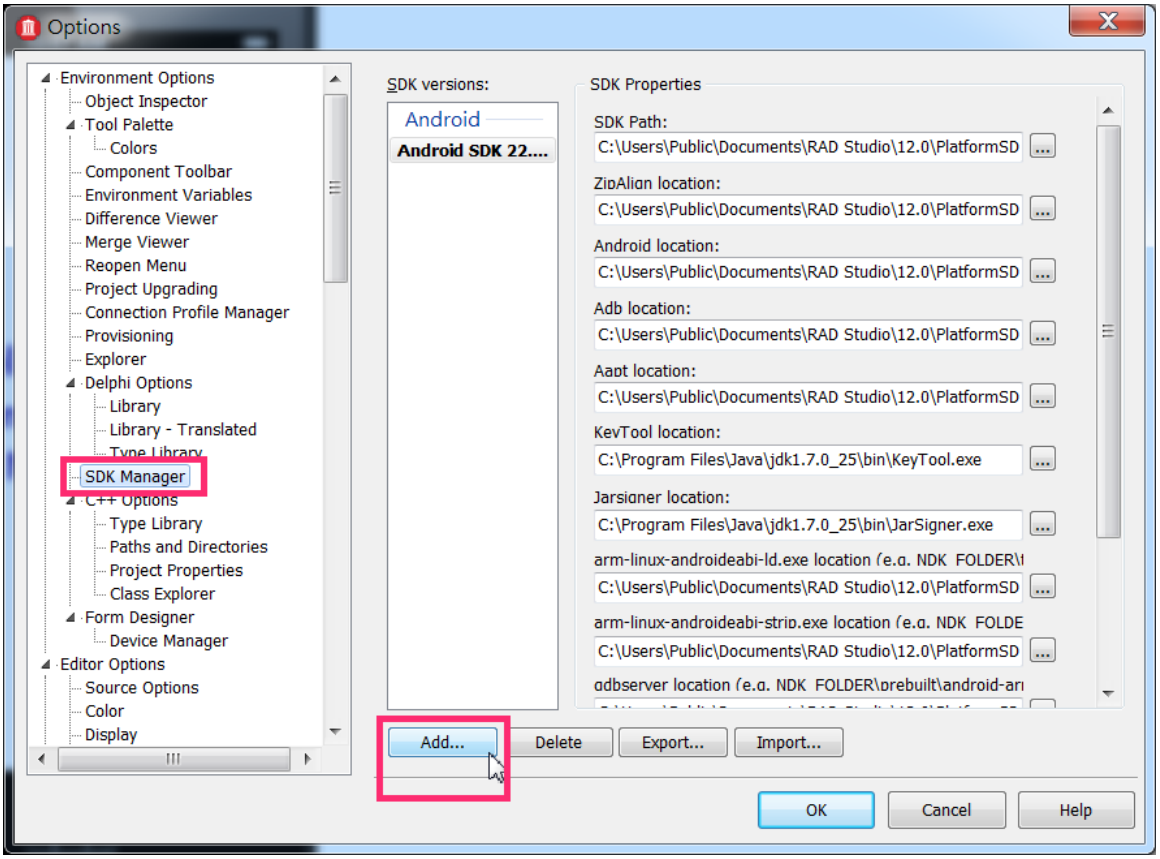
此外由於在 1-1 小節中安裝 PAServer 時 PAServer 使用的通信埠是『64211』而且我們沒有設定連結密碼，因此在上圖中 Port number 欄位中輸入了 64211，Password 欄位則沒有輸入密碼。

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 C++Builder 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。

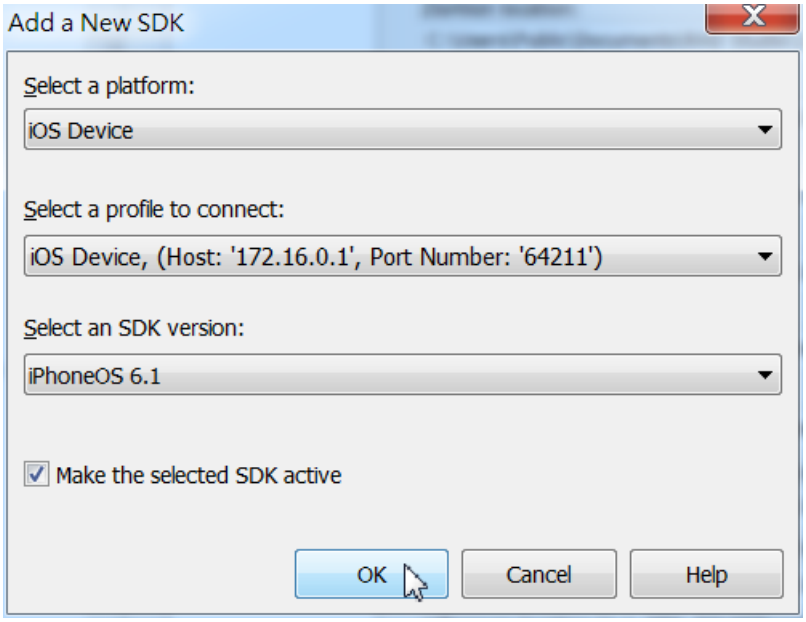
最後請點選『Finish』按鈕以完成建立組態的工作。



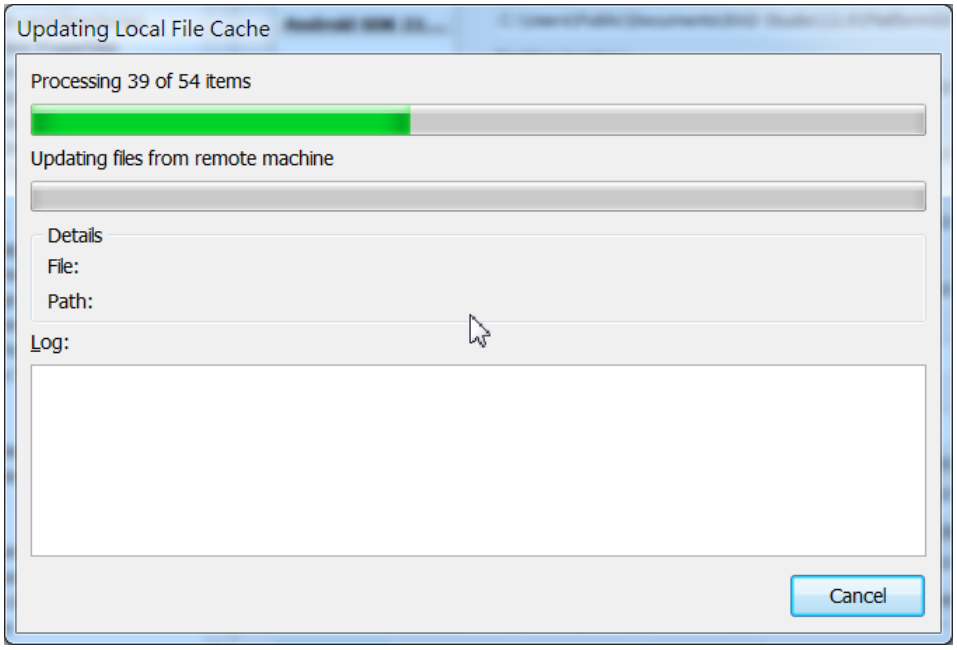
完成建立組態工作之後，最後要進行建立 iOS App 專案需要使用的 iOS SDK 的資訊。請如前述一樣在在 Options 對話盒中的 SDK Manager 選項中點選 Add 按鈕以加入 iOS SDK 資訊，如下面圖形所示：



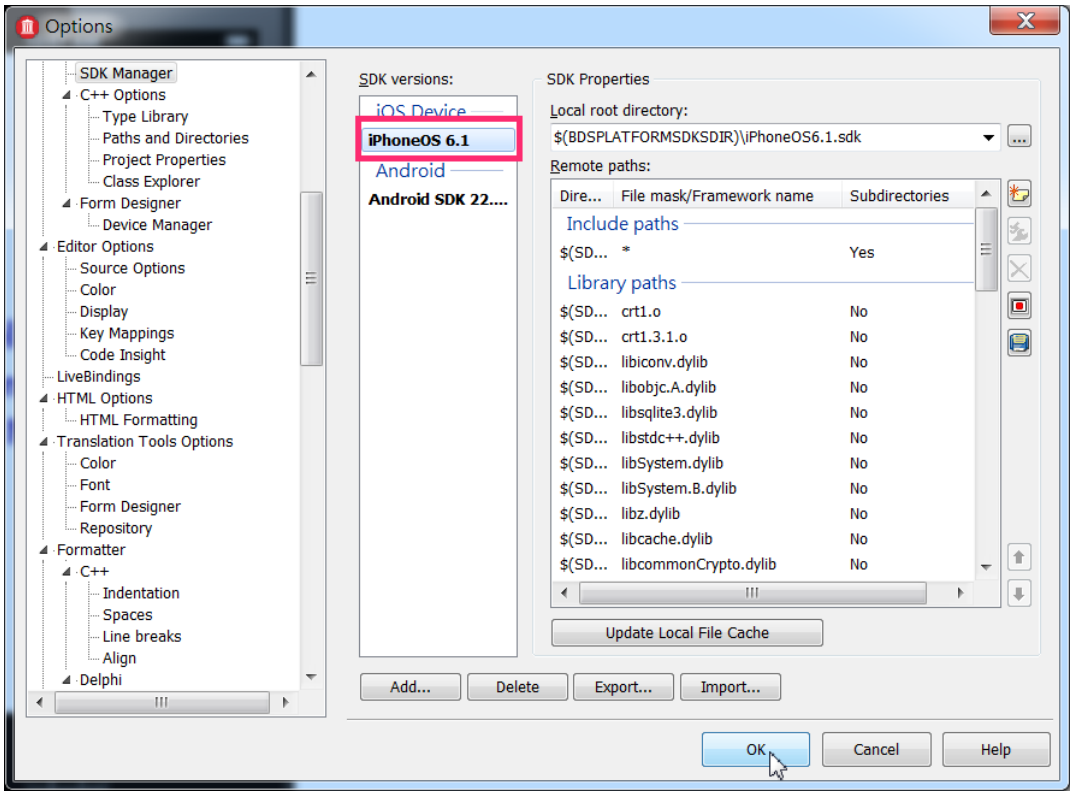
點選了 Add 按鈕之後 C++Builder IDE 會顯示如下的 Add a New SDK 對話盒，請在 Select a platform 下拉盒中選擇 iOS Device，在 Select a profile to connect 下拉盒中選擇前面步驟建立的組態名稱，最後在 Select an SDK version 下拉盒中會出現你使用的 iOS SDK 版本，請從下拉盒中選擇您使用的 SDK 版本，如下面圖形所示：



點選上面對話盒中的 OK 按鈕後 C++Builder 便會自動設定 iOS APP 專案需要使用的 iOS SDK 相關資訊：



在上面的步驟完成之後您就可以在 SDK Manager 選項中看到出現了 iOS Device 項目，在其中則會出現您的 iOS SDK 的設定資訊，如下面圖形所示：

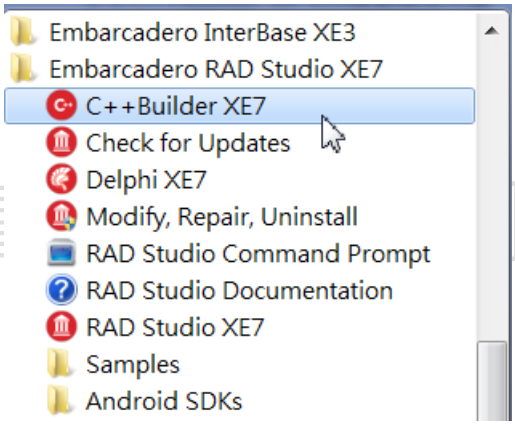


現在我們就可以準備進入 C++Builder for iOS 整合發展環境來開發 iOS App 了。

> C++Builder for iOS 只支援直接在 iOS 設備中開發 App，並不支援 iOS Simulator，因此在前面設定的步驟中是直接設定 iOS 設備的相關組態資訊，本書稍後討論的內容和範例也都是直接在 iOS 設備中測試和執行的，在未來的 C++Builder 版本才會支援 iOS Simulator。

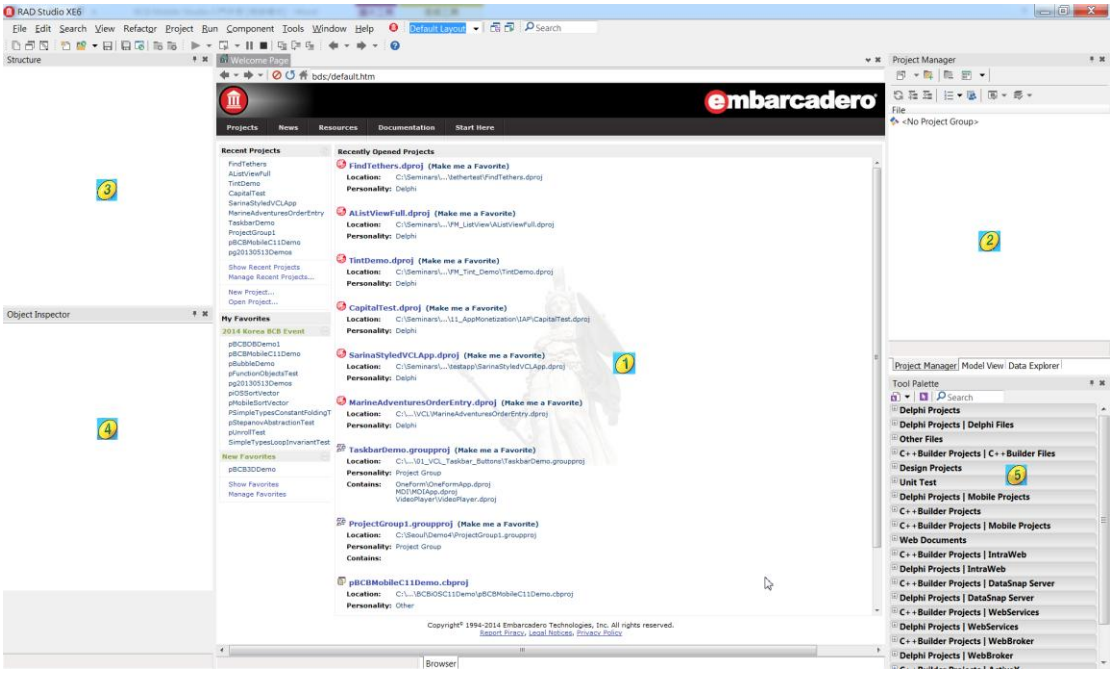
2. 進入 C++Builder for iOS 整合發展環境

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



點選 Embarcadero RAD Studio 或是 C++Builder 圖像執行 C++Builder for iOS

一進入 C++Builder for iOS 整合發展環境，您會看到類似如下的畫面，C++Builder for iOS 整合發展環境除了功能表和工具列之外(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:

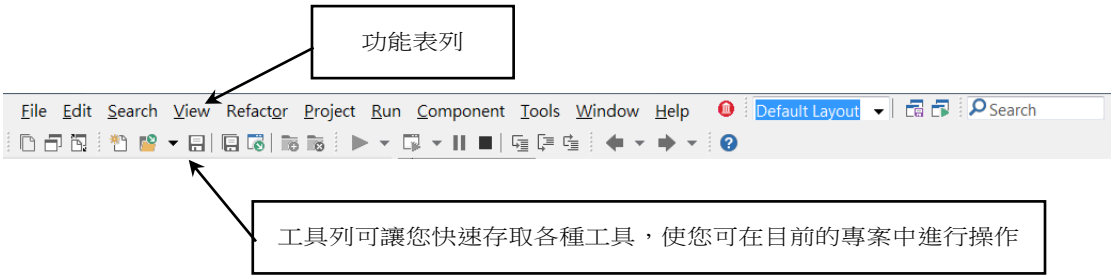


C++Builder for iOS 整合發展環境進入畫面

下面的表格說明了這成 5 大區域的功能:

區域編號	說明
	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。
	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
	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
	物件檢視器(Object Inspector)，在專案設計時期可檢視，設定元件的特性值，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
	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:



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 C++Builder for iOS 專案。

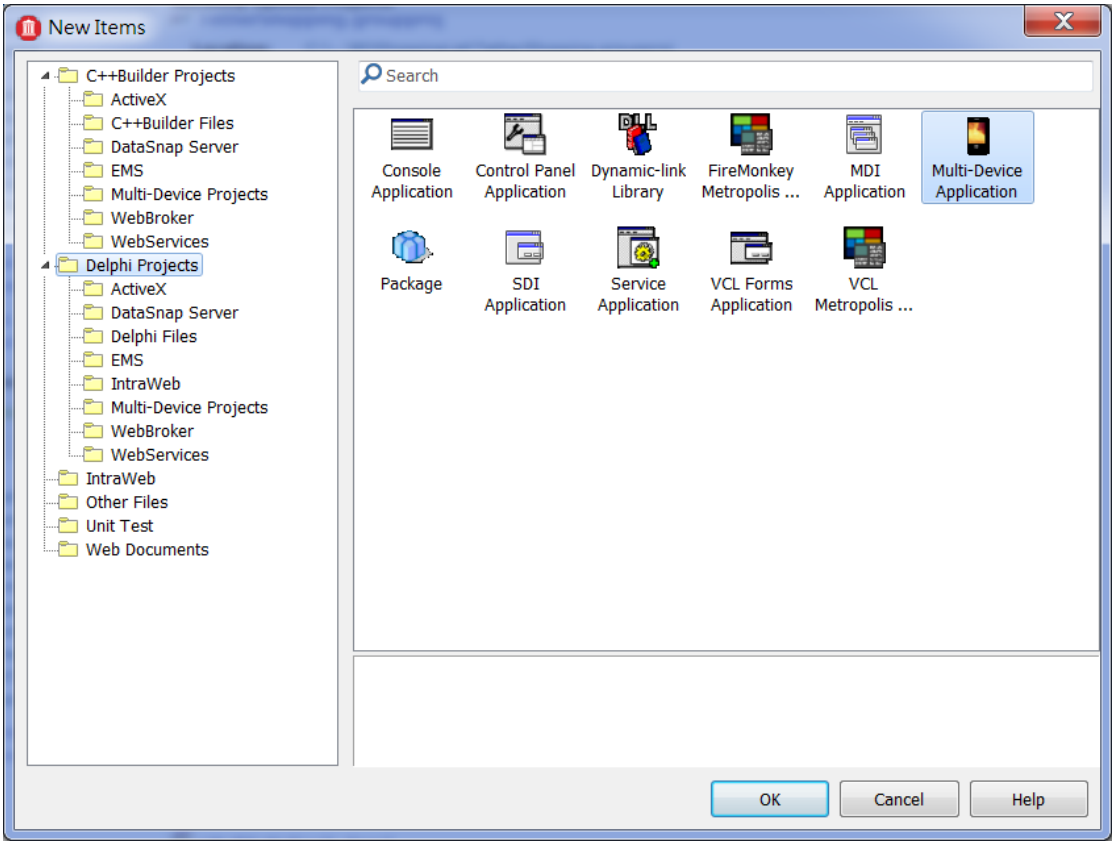
2. 建立新專案

要使用 C++Builder 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

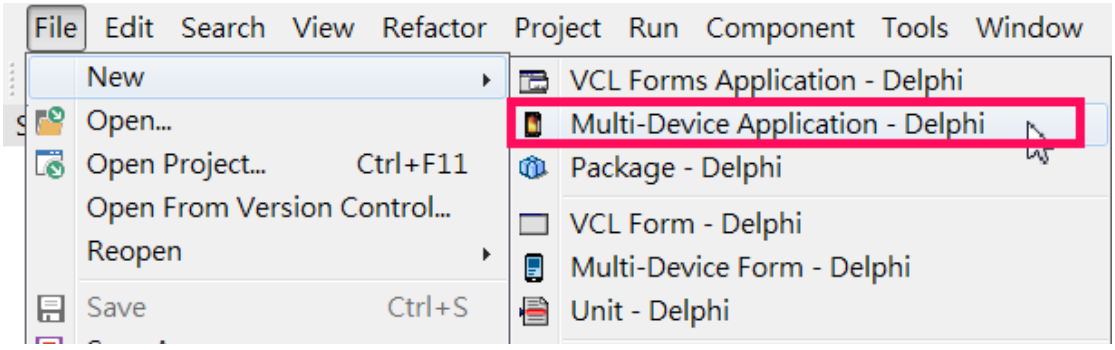
在 C++Builder for iOS 整合發展環境中有許多方法可以建立新專案，首先您可以點選工具列中的『New items』快捷鍵，它位於工具列從左方開始的第 4 個快捷鍵:

點選了工具列中的『New items』快捷鍵之後，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 C++Builder Projects 項目中選擇『Multi-Device Application』圖像以建立 iOS App 專案，如下圖所示:



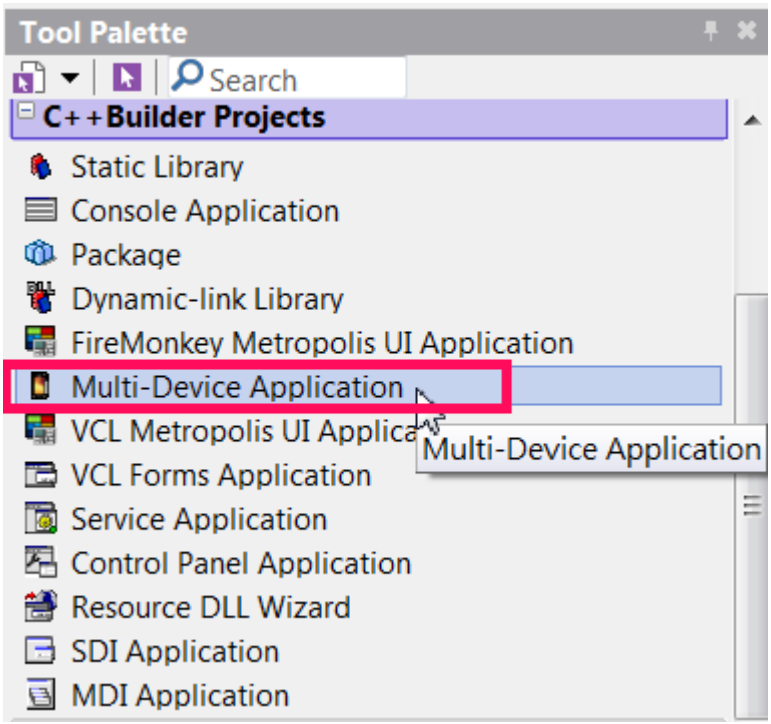
在 New Items 對話盒中選擇要建立的專案型態

第二種建立專案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專案型態，如下所示:



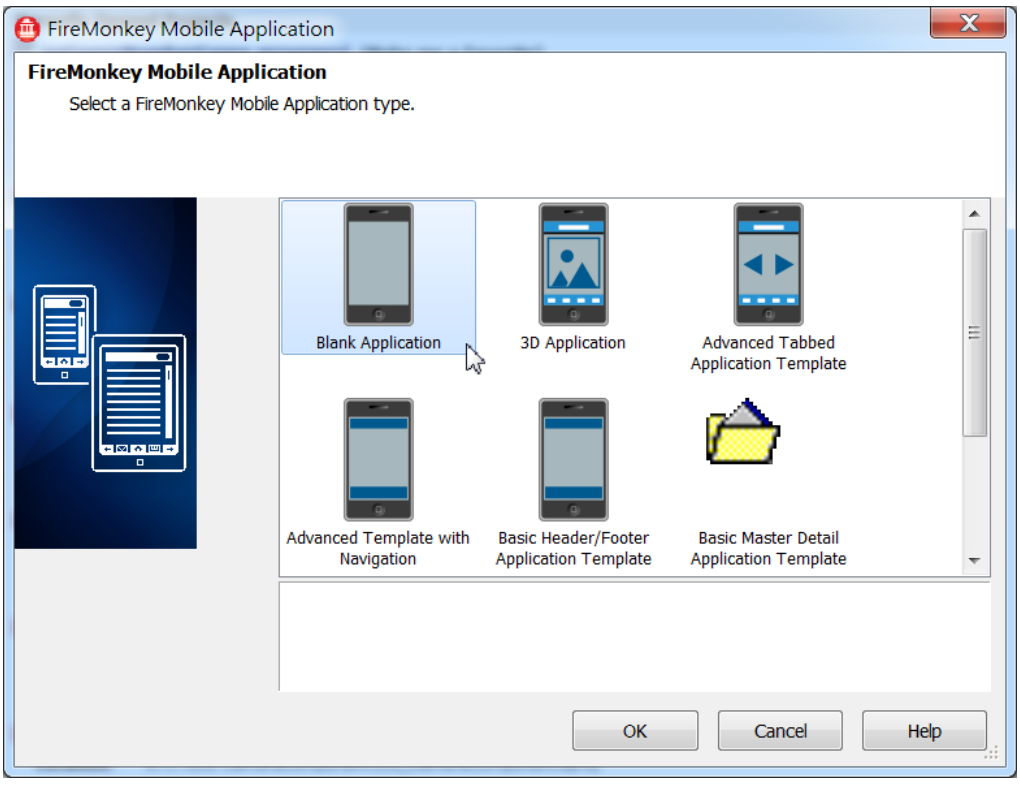
選擇功能表中的 File|New 選項並且選擇要建立的專案型態

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專案型態，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『FireMonkey Mobile Application』專案，C++Builder 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:



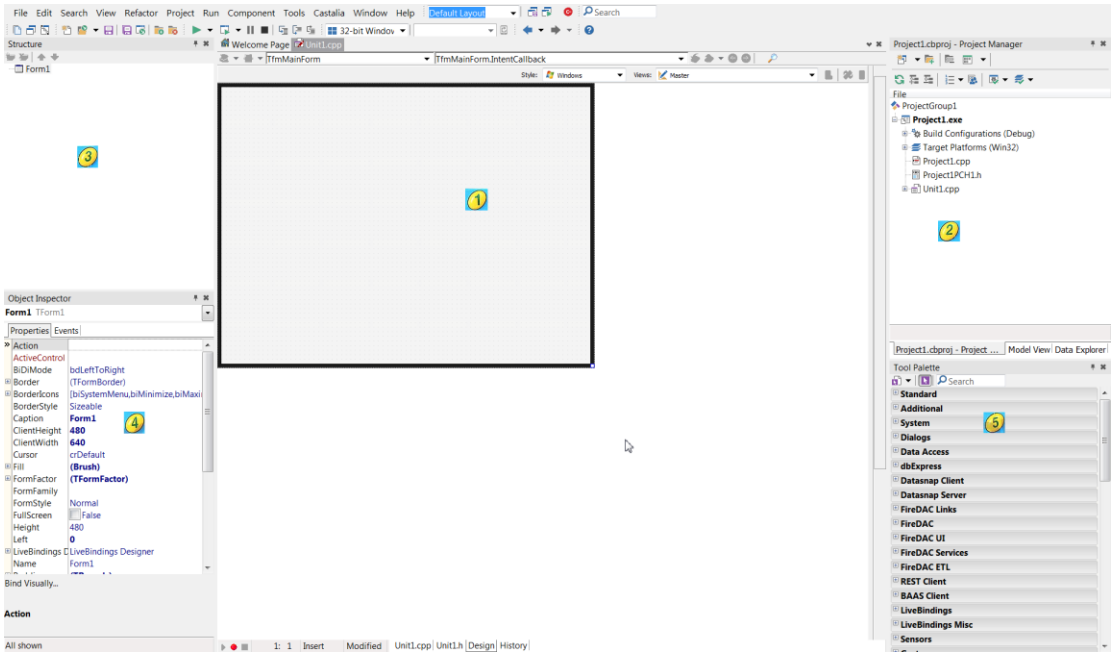
下面的表格說明了這麼專案的意義:

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 專案	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 C++Builder for iOS 專案組成元素

現在 C++Builder 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:

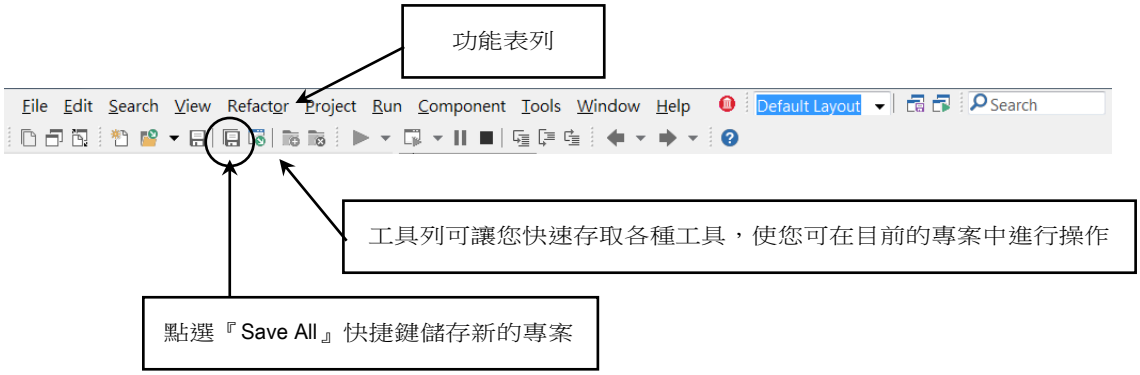


整合發展環境建立 FireMonkey 專案並且開啟專案主表單，準備讓您開始設計使用者介面

區域編號	說明
	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
	專案管理員顯示專案中所有的檔案和相關的設定
	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)

	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
	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

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』快捷鍵儲存新的專案:

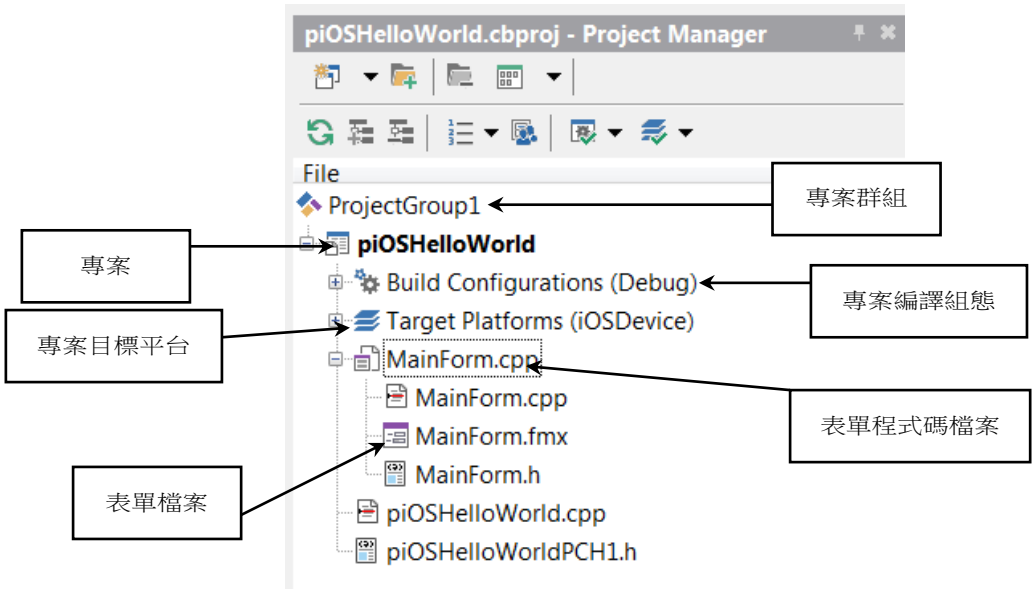


或是同時按下『Shift+Control+S』3 個按鍵儲存專案，當儲存專案時整合發展環境會詢問以什麼名稱儲存表單和專案，請以『MainForm』儲存主表單，以『piOSHelloWorld』儲存專案。

> 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:\MyDemos，如果您直接儲存專案而沒有選擇特定的目錄，那麼 C++Builder IDE 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

C++Builder 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。

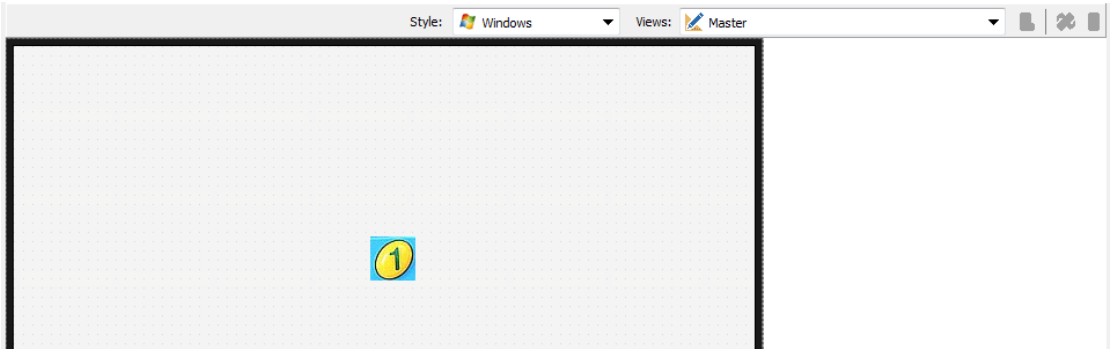
除了表單和程式碼檔案之外，專案中尚有其他的元素，這些元素都會顯示在專案管理員中。其中的『專案編譯組態』可設定目前專案是使用『除錯』模式，『部署』模式或是『客製化』模式。而『專案目標平台』則可設定專案編譯的目標平台，FireMonkey 專案可支援 Win32，Win64，Mac OSX 或是 iOS 平台，由於我們是使用 C++Builder for iOS 開發，而且在 1-2 小節中我們設定『iOS Simulator』為內定平台，因此在專案中的 Target Platforms 節點中可以看到 iOS Simulator 被設定為目前專案使用的平台:



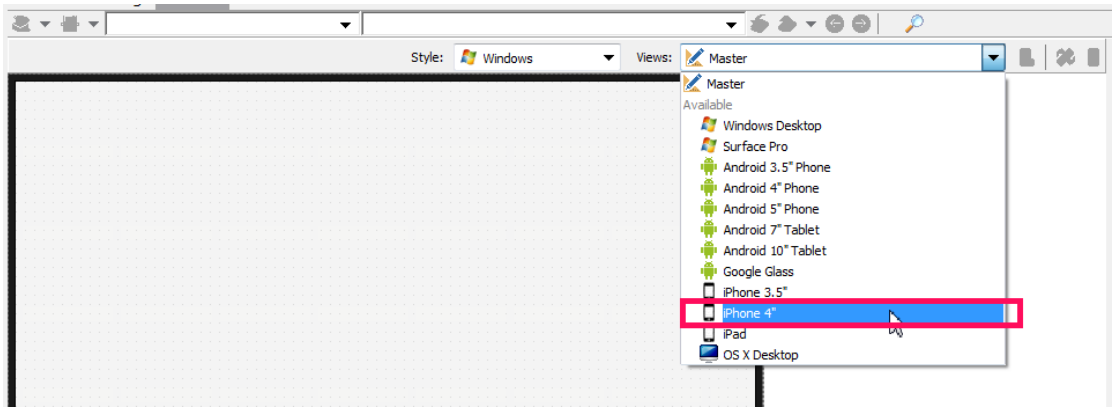
專案管理員

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 來設計不同平台的 UI，當開發人員在 C++Builder 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。

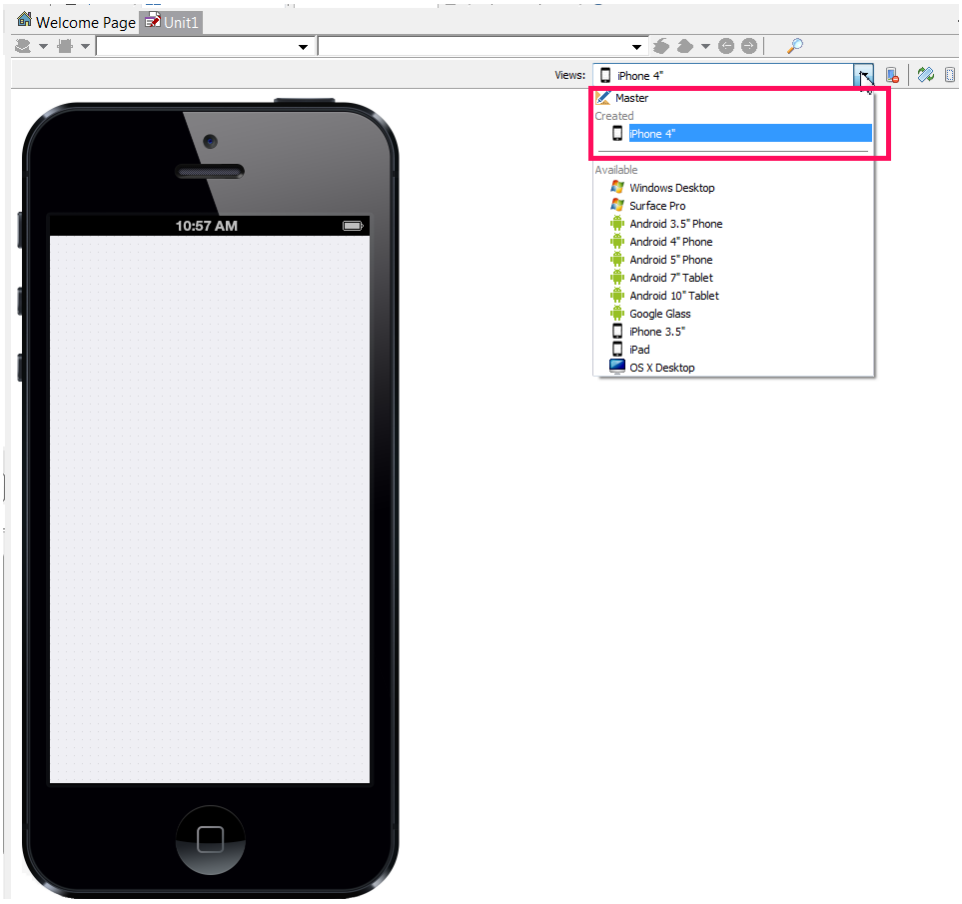
開發人員在 Master View 上加入的任何元件都會自動繼承到其他子的 View 之中。當然開發人員也可以在子 View 中直接進行 UI 設計，



現在就讓我們在此專案中建立一個 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：



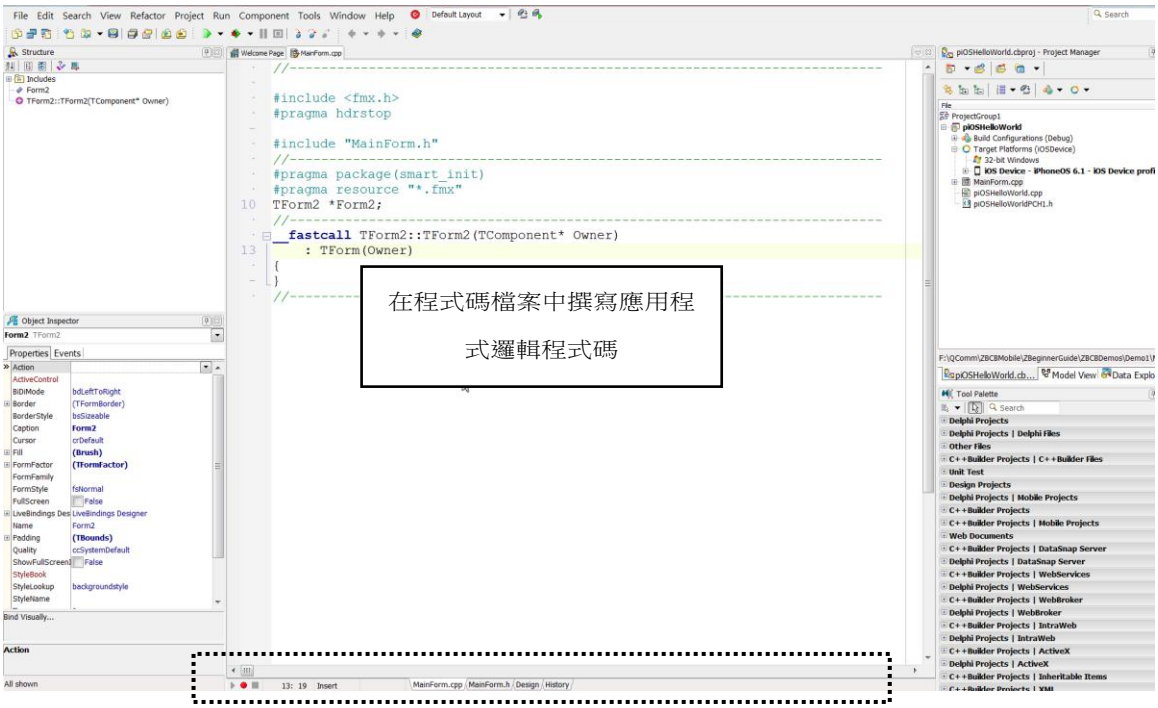
點選了 iPhone 4 子 View 之後 IDE 便會建立一個 iPhone 的設計表單如下所示並且在 IDE 右上方的 Views 下拉盒中可以看到現在專案有 2 個 View：Master View 和 iPhone 4：



當專案建立之後，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:

當專案建立之後，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。



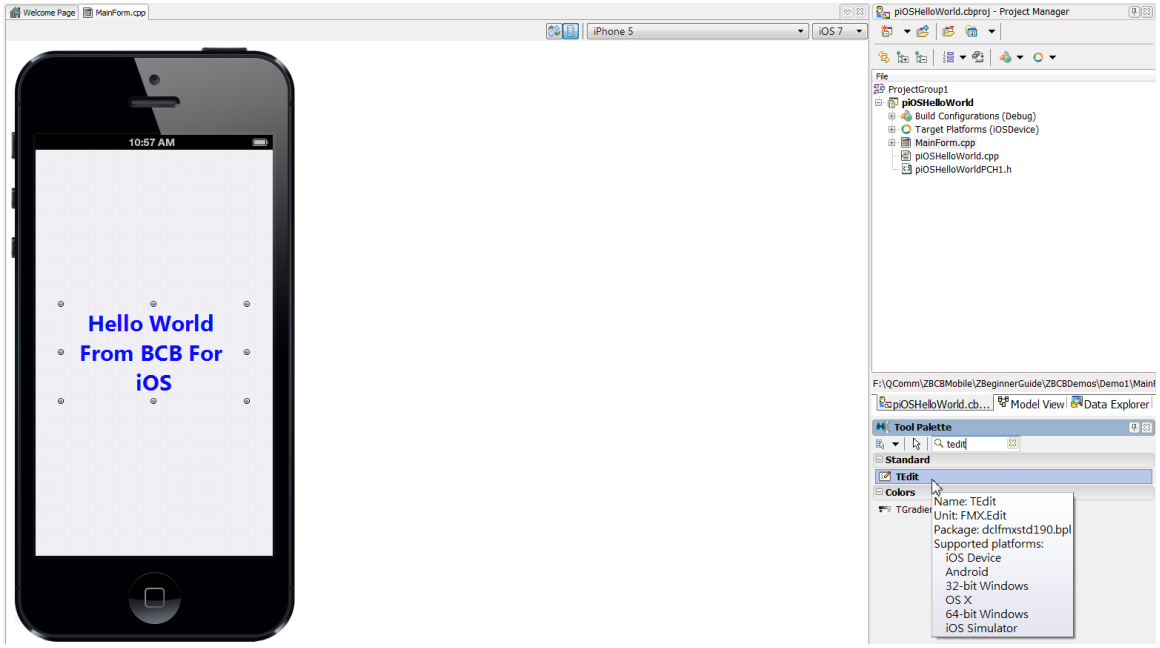
或是按下『F12』鍵切換表單和程式碼頁次，或是點選工具列中的『Toggle Form/Unit(F12)』快捷鍵來切換:

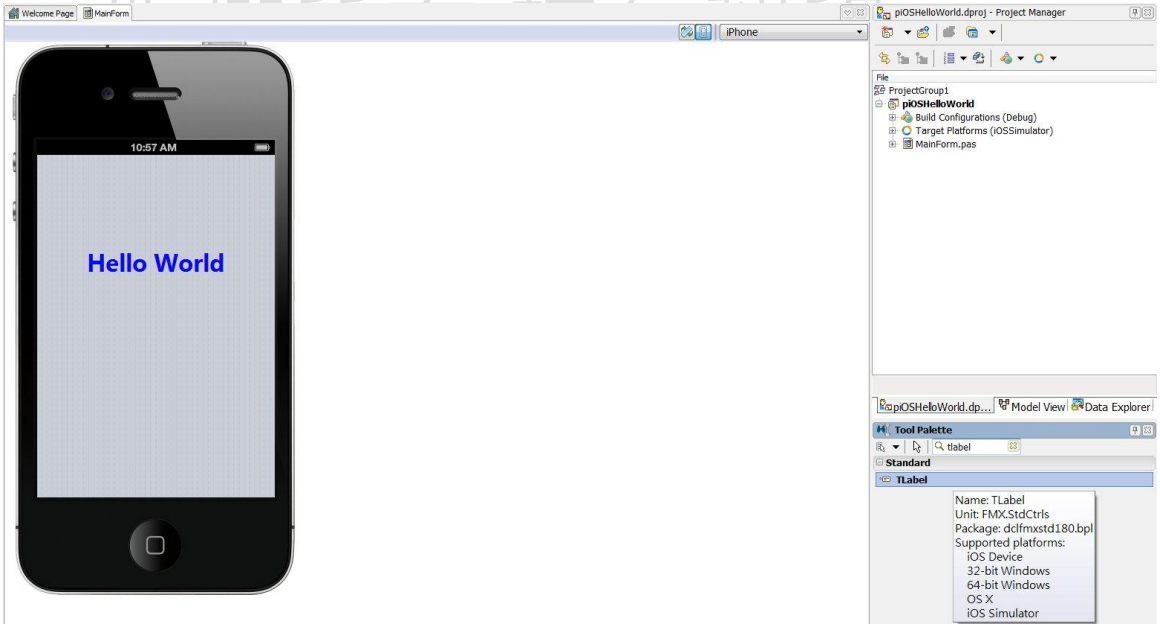
現在我們就可以開始開發您的 iOS 應用程式了。

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 C++Builder 的筆者來說，XCode 實在過於繁瑣，讓我們看看 C++Builder 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 C++Builder 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:

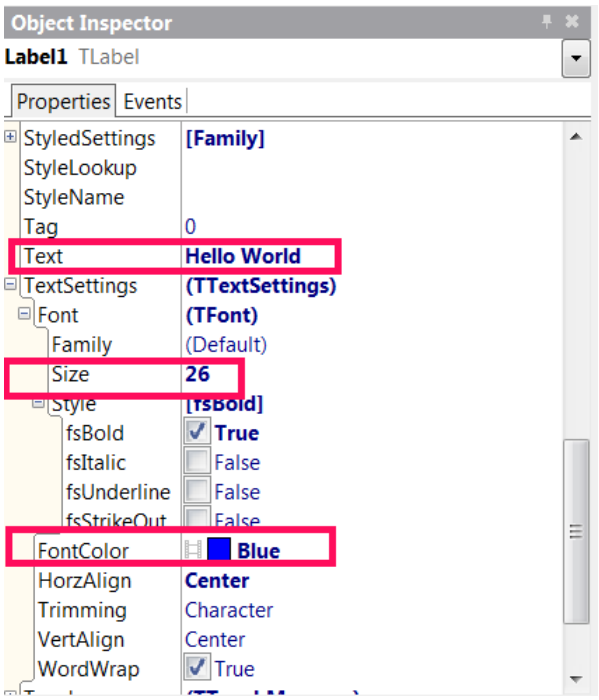




接著到 IDE 左下方的物件檢視器中設定 TLabel 如下的特性值:

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

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，C++Builder for iOS 就會編譯，並且分發『piOSHelloWorld』專案到您的 iOS 設備中執行了，例如下圖就是『piOSHelloWorld』專案在筆者的 iPhone 5 手機中執行的畫面(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 C++Builder 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：

然後使用滑鼠雙擊 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:

```
void __fastcall TForm2::Button1Click(TObject *Sender)
{
  ShowMessage("歡迎使用 C++Builder for iOS!");
}
```

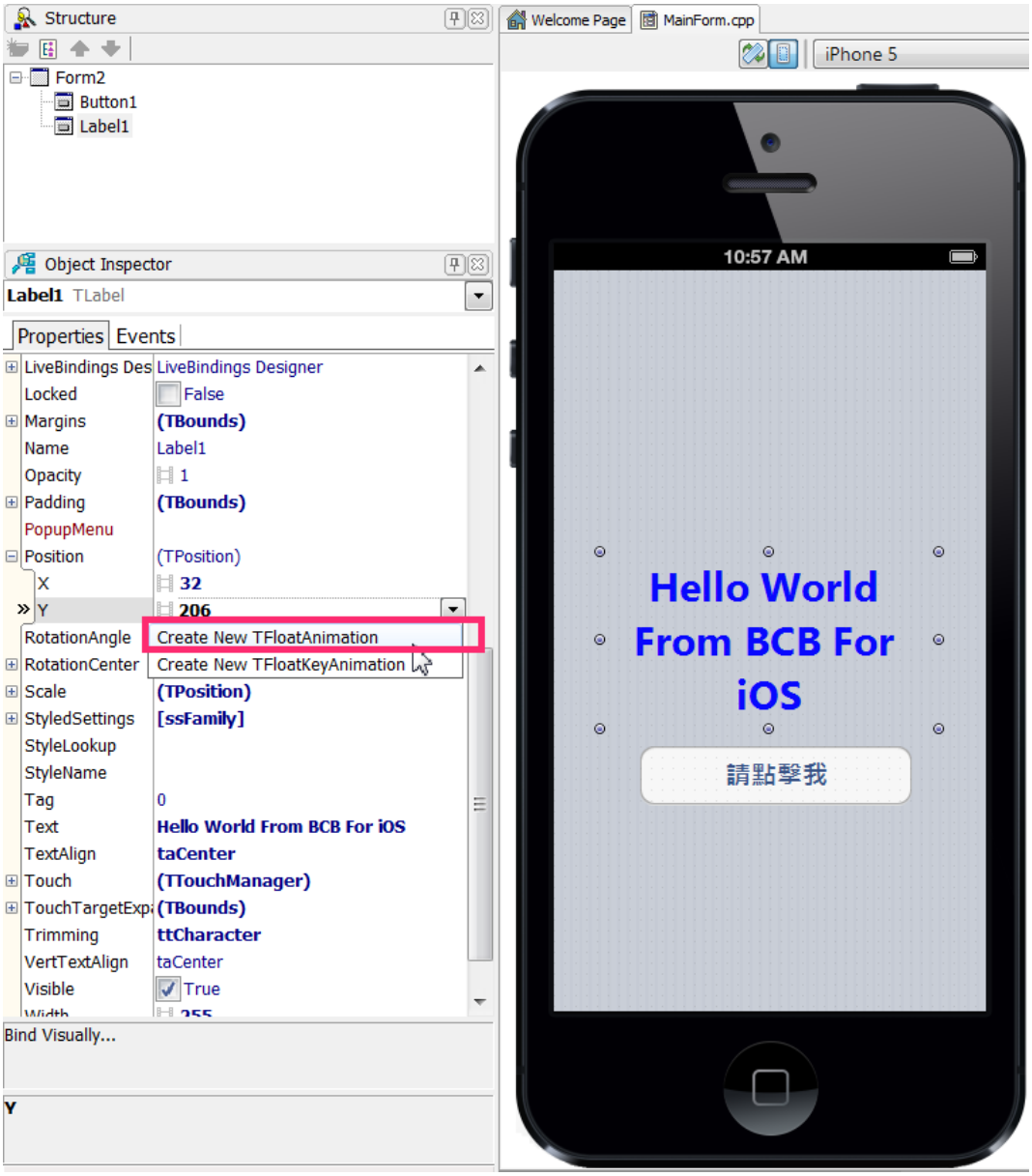
在稍後本書會說明，這段程式碼稱為事件處理函式，在其中我們呼叫 ShowMessage 函式顯示訊息，ShowMessage 函式是 C++Builder for iOS 的執行時期函式館中的函式。

現在請再次按下 F9 執行『piOSHelloWorld』專案，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：

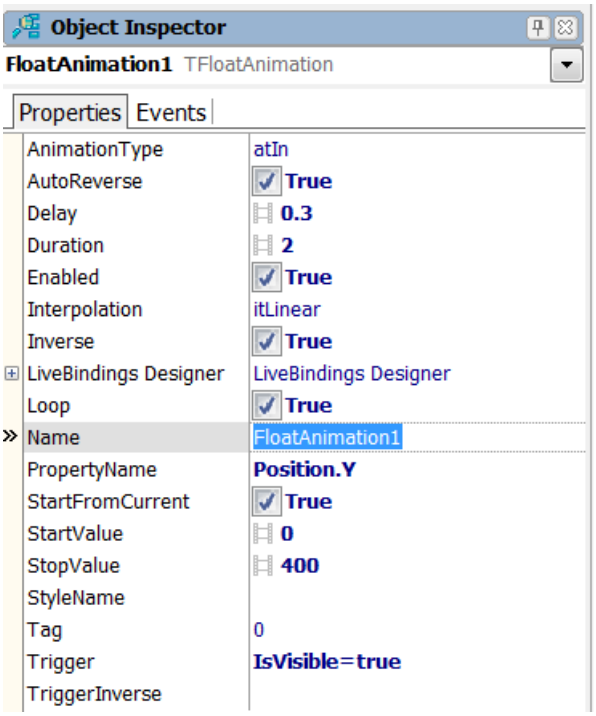
讀者可以看到使用 C++Builder for iOS 撰寫 iOS 程式碼並且連結視覺化元件是多麼的容易。在我們離開第一個 C++Builder 開發的 iOS App 之前，最後再讓我們為『Hello World From BCB For iOS』這個文字加入動畫的功能。

讓我們在執行 piOSHelloWorld App 把『』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，如此反覆不停的移動。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 C++Builder 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功能表，請點選其中的『Create New TFloatAnimation』以便為 TLabel 元件的 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:

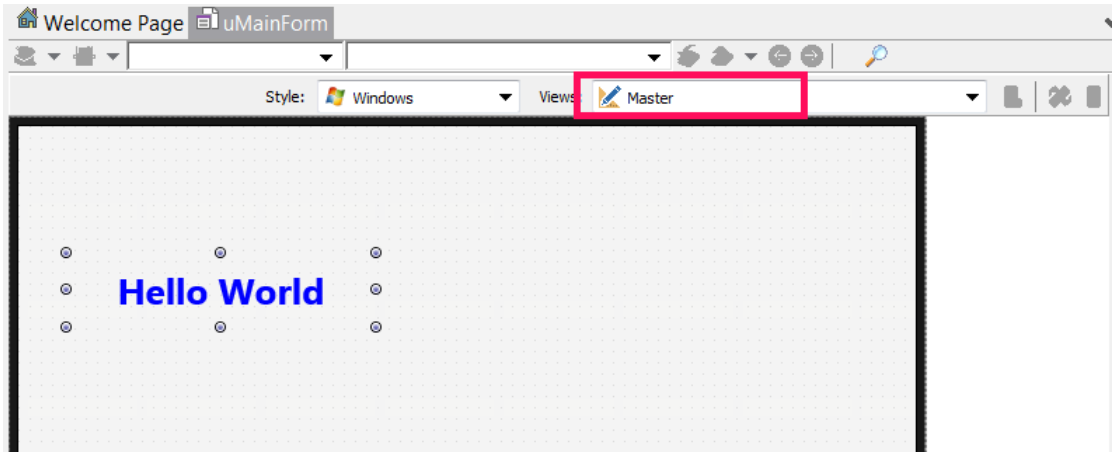
特性	特性值	說明
AutoReverse	True	當動畫完成時自動以反方相再執行一次
Delay	0.3	每一動畫動作之間延遲 0.3 秒
Duration	2	此動畫效果一共執行 2 秒
Enabled	True	啟動動畫功能
Loop	True	不斷重覆執行此動畫功能
StartFromCurrent	True	從目前 TLabel 元件的 Y 軸位置開始動畫功能
StopValue	400	動畫功能到達 TLabel 元件的 Y 軸位置 400 的地方時就完成
Trigger	IsVisible=true	當 TLabel 元件顯示時就觸發執行動畫功能

現在請再次按下 F9 執行『piOSHelloWorld』專案我們，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：

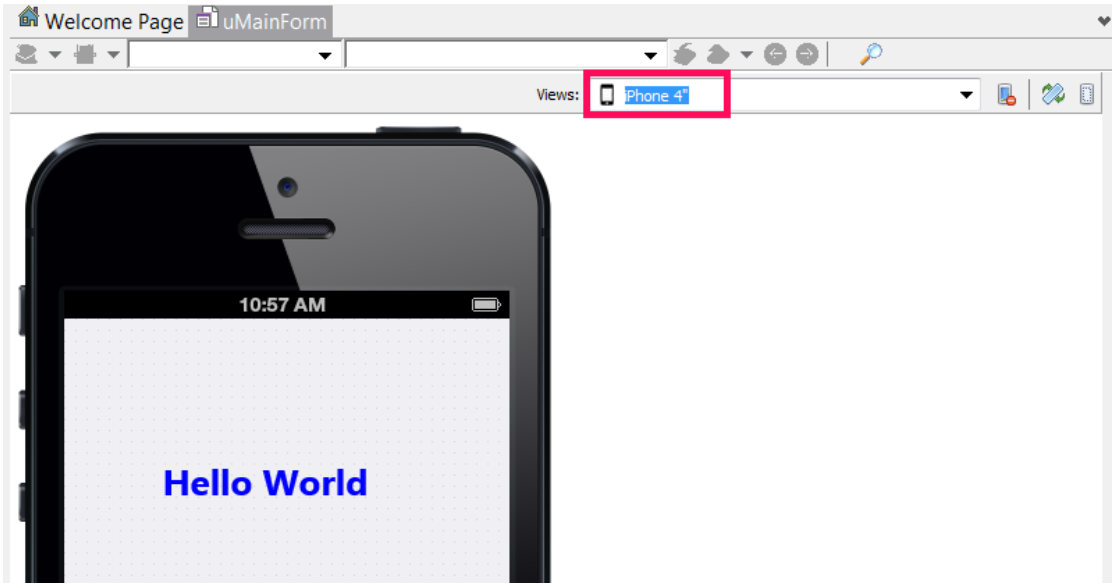
OK，現在我們完成了第一個 iOS App，這個具備動態視覺化效果的 Hello World iOS App 使用 C++Builder for iOS 開發起來不但比其他 iOS 開發工具更簡單，提供的功能也遠遠超過了只是靜態的顯示 Hello World 文字。從這個簡單的範例就可以證明 C++Builder for iOS 比其他 iOS 開發工具更強大，更易於使用，也具體更高的生產力。

3-1 使用 MDD 設定

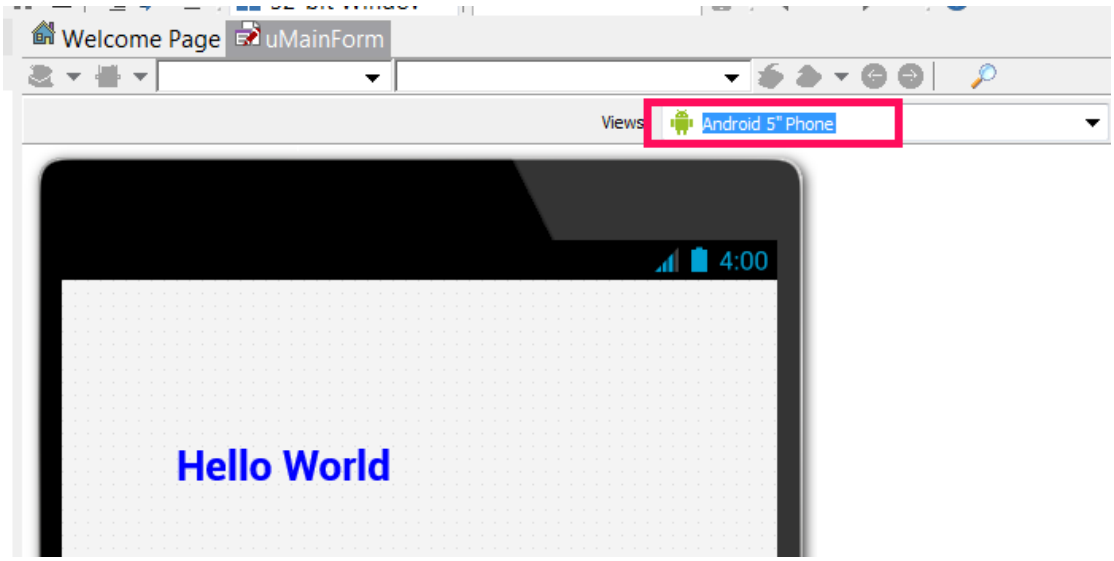
在前面的說明中我們是直接在 iPhone 的子 View 中進行 UI 設計，當然您也可以在 Master View 中進行共用設計再到特定的子 View 中進行調整，例如我們可以先在 Master View 中加入 Hello World 的 TLabel 元件：



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：

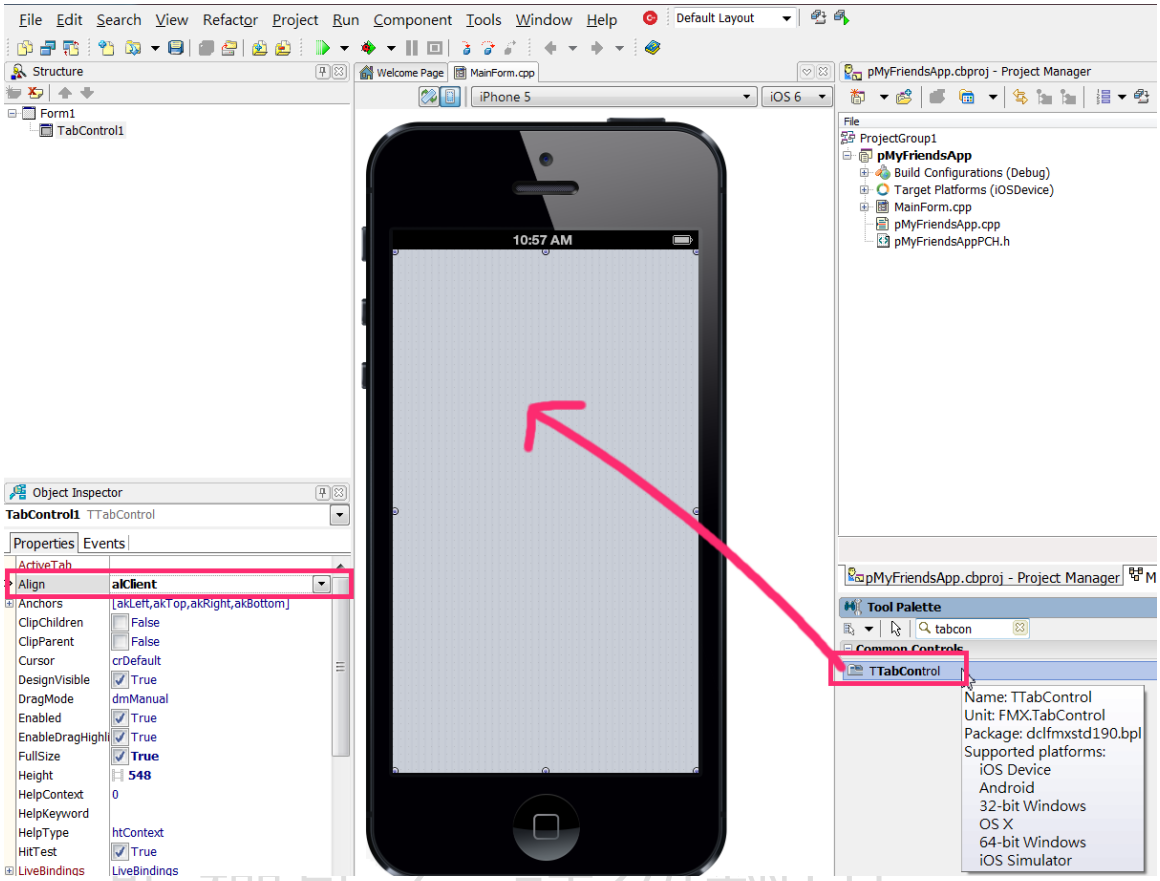


接下來讀者將開始學習如何有效的使用 C++Builder for iOS IDE 來開發 iOS App，我們將使用 FireMonkey For Mobile 框架開發個人通訊錄 App，在開發的過程中讀者將學習許多 C++Builder for iOS 的使用技巧以及 FireMonkey For Mobile 框架的元件。

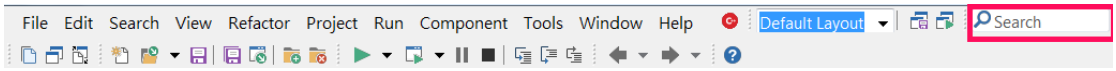
4. 使用 C++Builder for iOS 整合發展環境

請執行 C++Builder for iOS IDE，建立一個『HD FireMonkey Mobile Application』專案，並且以『pMyFriendsApp』為名稱儲存此 iOS App。

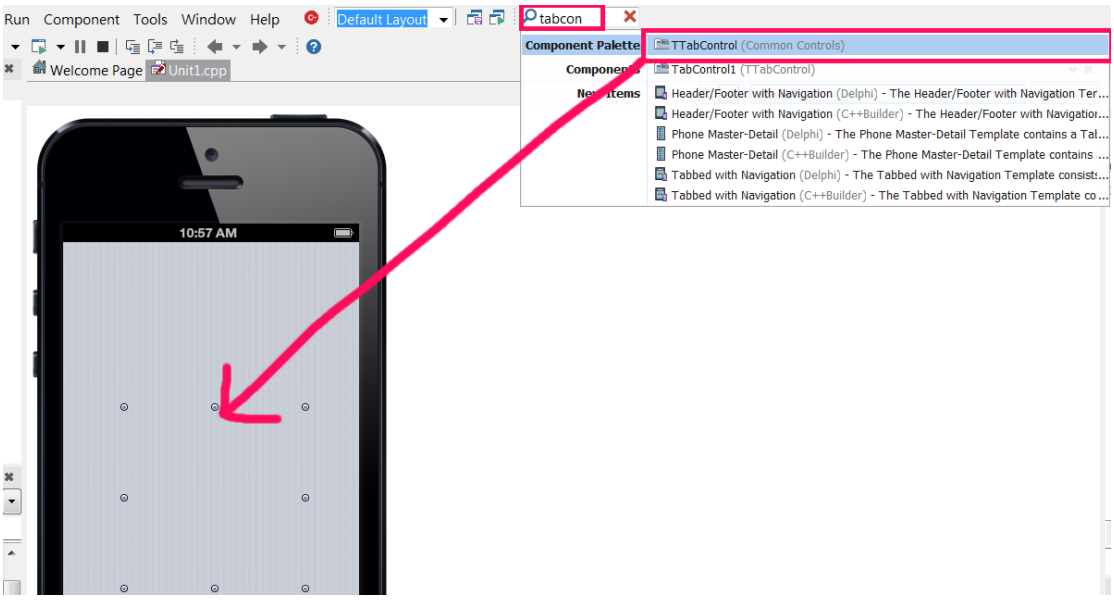
在 IDE 左下方的工具盤中搜尋 TTabControl 元件，拖曳到 iPhone 主表單中，在物件檢視器中設定 TTabControl 元件的 Align 特性值為『alClient』以便讓 TTabControl 元件佔據整個使用者顯示區域，如下所示：



由於 FireMonkey For Mobile 框架提供的元件非常的多，因此您在 IDE 右上方的 Search 控制項中搜尋任何 C++Builder 相關的物件，如下所示：

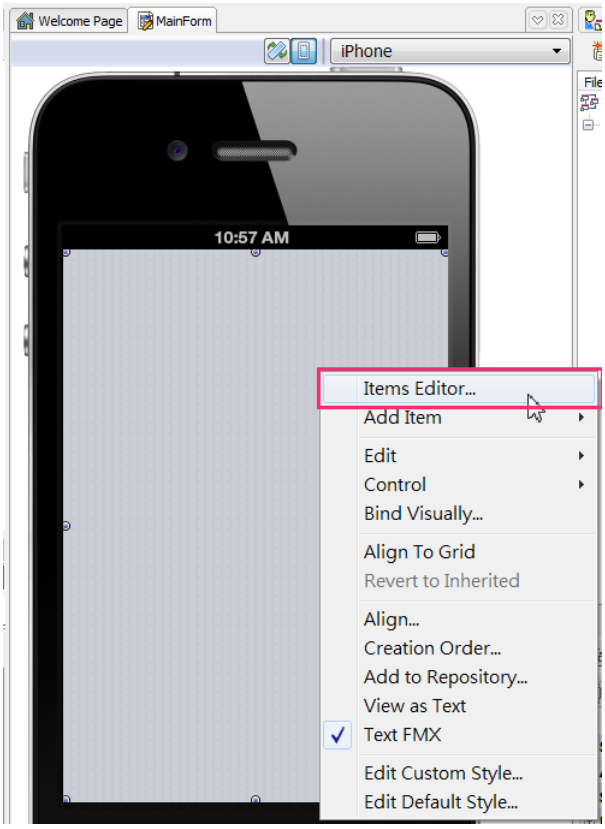


您可以在 IDE 右上方的 Search 控制項中輸入 TTabControl 即可找到 TTabControl 元件，接著使用滑鼠雙擊 TTabControl 即可自動把 TTabControl 加入表單中。或是您要搜尋的元件的部份名稱，例如『tabcont』，『bcontrol』等都可以找到符合輸入名稱的元件。

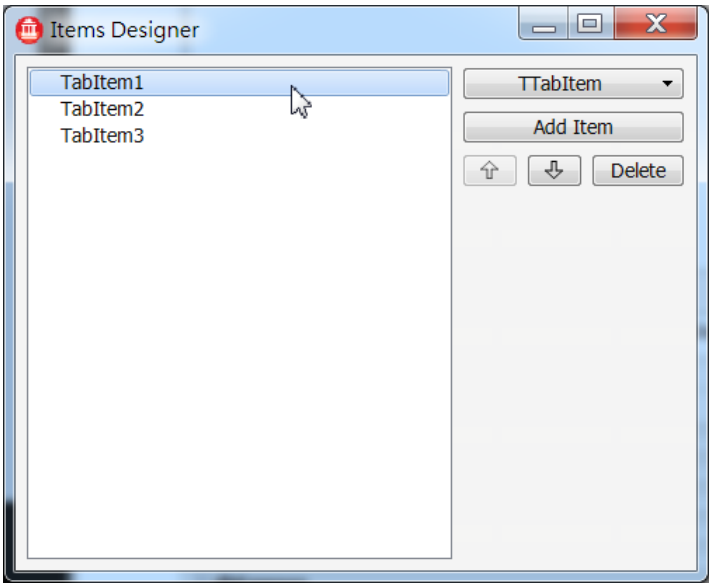


> 請注意『IDE Insight』會根據您現在使用的模式來搜尋內容，例如如果您是在程式碼編輯器中於 IDE 右上方的 Search 控制項中搜尋 TTabControl，那麼便搜尋不到，因為 TTabControl 元件無法加入到程式碼之中。

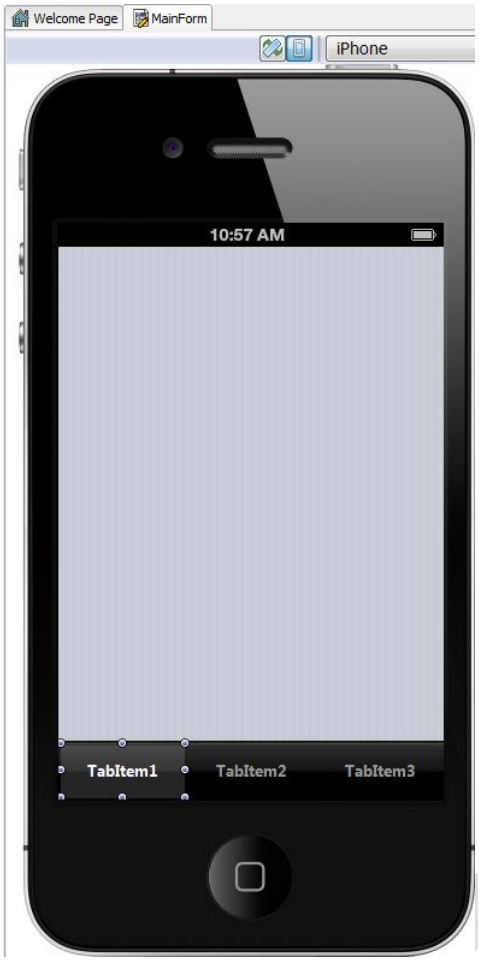
現在讓我們在這個 TTabControl 元件中加入數個頁面，請點選表單設計家中的 TTabControl 元件並且點選滑鼠右鍵，此時 IDE 便會顯示一個快顯功能表，請選擇其中的『Items Editor…』選項以啟動選項設計者對話盒:



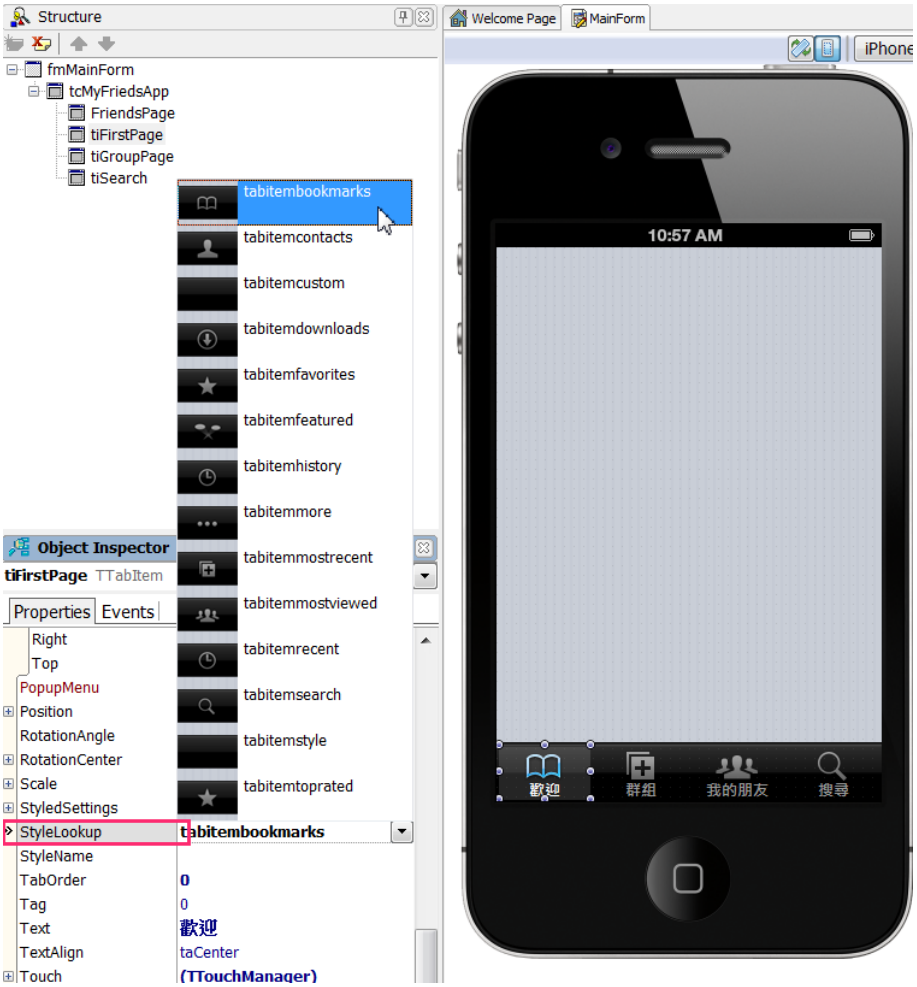
接著點選對話盒中的『Add Item』按鈕在 TTabControl 元件中加入四個 TTabItem 子元件，如下所示:



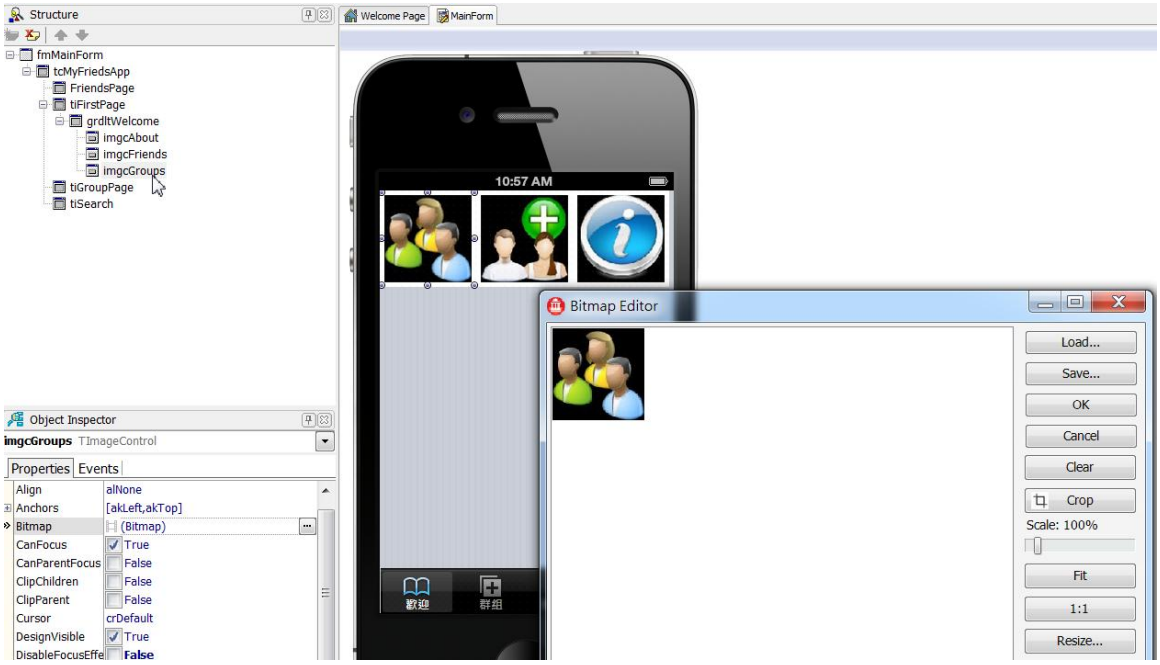
接著在物件檢視器中設定 TTabControl 元件的 TabPosition 特性值為『tpBottom』，此時表單設計者便類似如下所示:



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:



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雙擊它的 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:

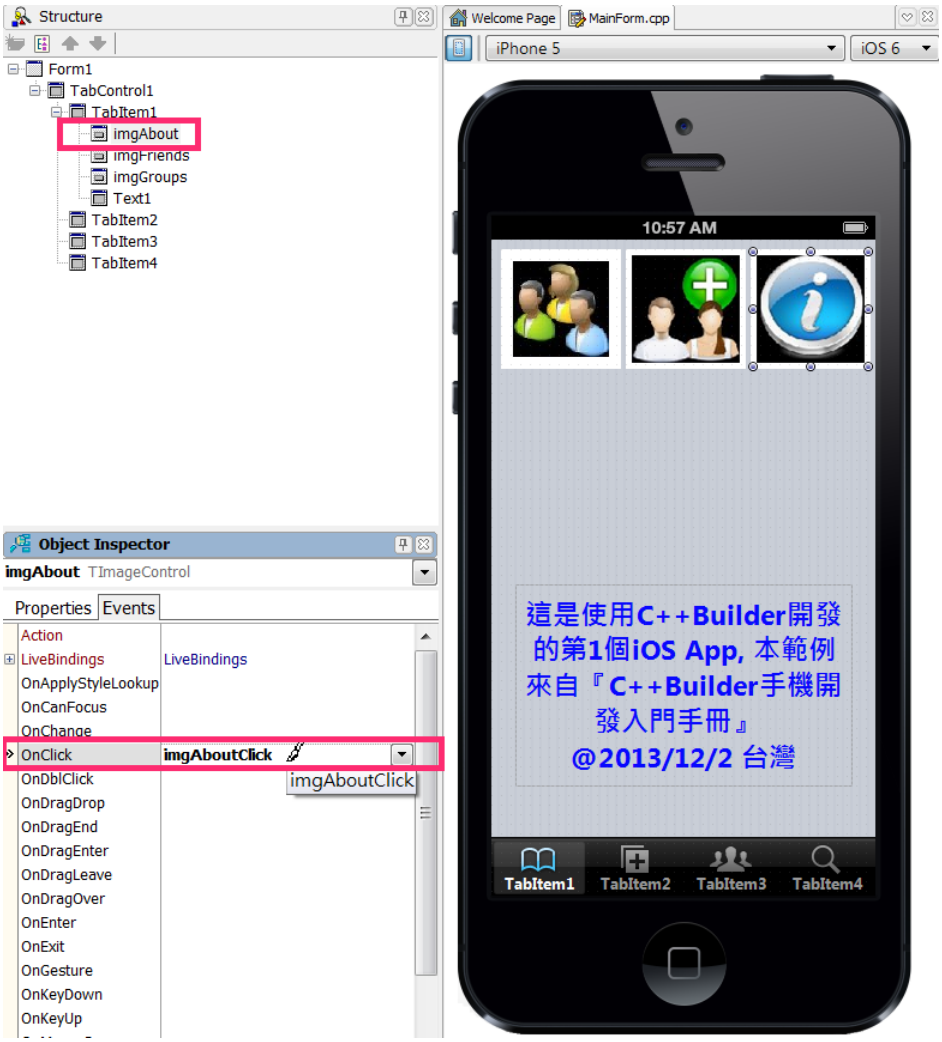


現在讓我們為這個 FireMonkey App 加入一些有趣的功能，這個功能就是當這個 FireMonkey App 執行時，如果使用者點選上圖中最右方的圖像時就動態顯示一個此 App 的資訊文字，這個動態文字會從畫面下方往上出現，接著在一定的時間之後這個資訊文字就會消失。

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 頁次中雙擊它的 OnClick 項目以自動產生 OnClick 事件處理函式，如下所示:

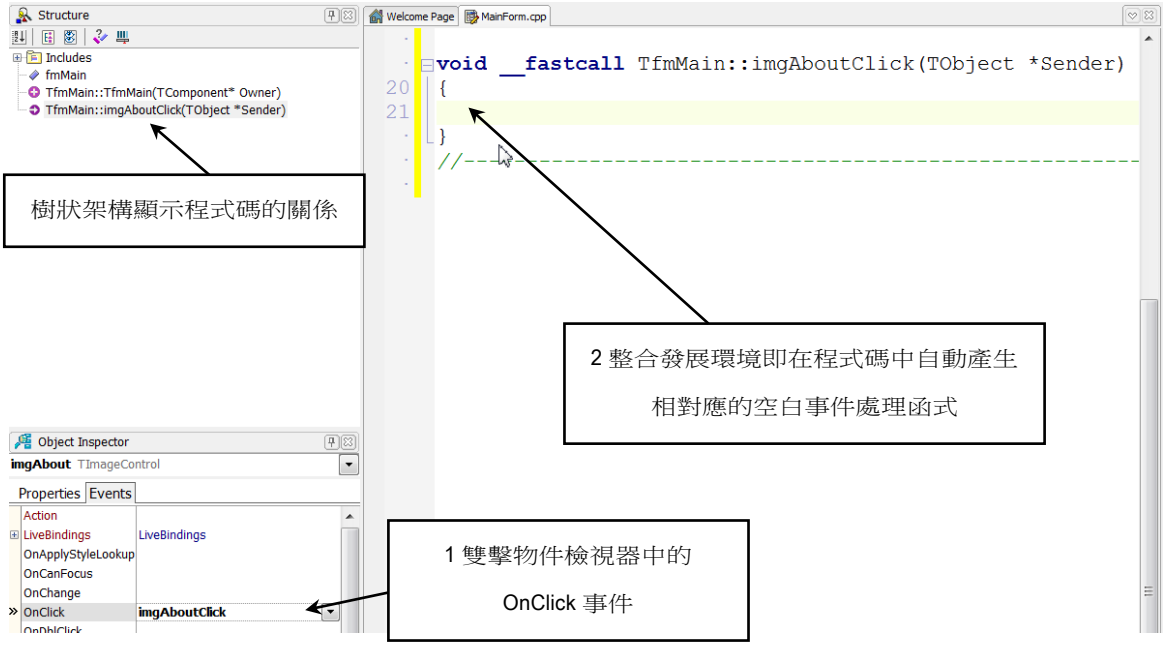
現在我們開始需要撰寫一些程式碼了，請點選 TTabControl 的第一個頁面中最右方的圖像元件，接著點選物件檢視器中的『Events』頁次，並且雙擊其中的 OnClick 事件，如下所示:



點選表單，在物件檢視器的『Events』頁次雙擊事件以自動產生事件處理函式

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:



雙擊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』頁次雙擊 OnActivate 以自動產生主表單的 OnActivate 事件處理函式，如下所示:

主表單的 OnActivate 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。

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

```
txtWelcome->Position->Y = this->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    void __fastcall TfmMain::FormActivate(TObject *Sender)
002    {
003    tcMyFriedsApp->ActiveTab = tiFirstPage;
004    SetupWelcomeText();
005    }
006
//---------------------------------------------------------------------
007    void TfmMain::SetupWelcomeText()
008    {
```

```
009    txtWelcome->Visible = false;
010    txtWelcome->Position->Y = this->Height + 10;
011    }
```

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:

```
    void __fastcall AnimateFloat(const System::UnicodeString
APropertyName, const float NewValue, float Duration = 2.000000E-01f,
TAnimationType AType = (TAnimationType)(0x0), TInterpolationType
AInterpolation = (TInterpolationType)(0x0));
```

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。

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

```
((TControl *) (txtWelcome->Parent))->Height - txtWelcome->Height + 10
```

它代表 TText 元件的新垂直位置，也就是 TTabControl 元件第一個頁面的高度減去 TText 元件的高度，再加上 10 個像素的高度。

最後的參數值 2 代表整個動態效果的時間會維持 2 秒鐘。

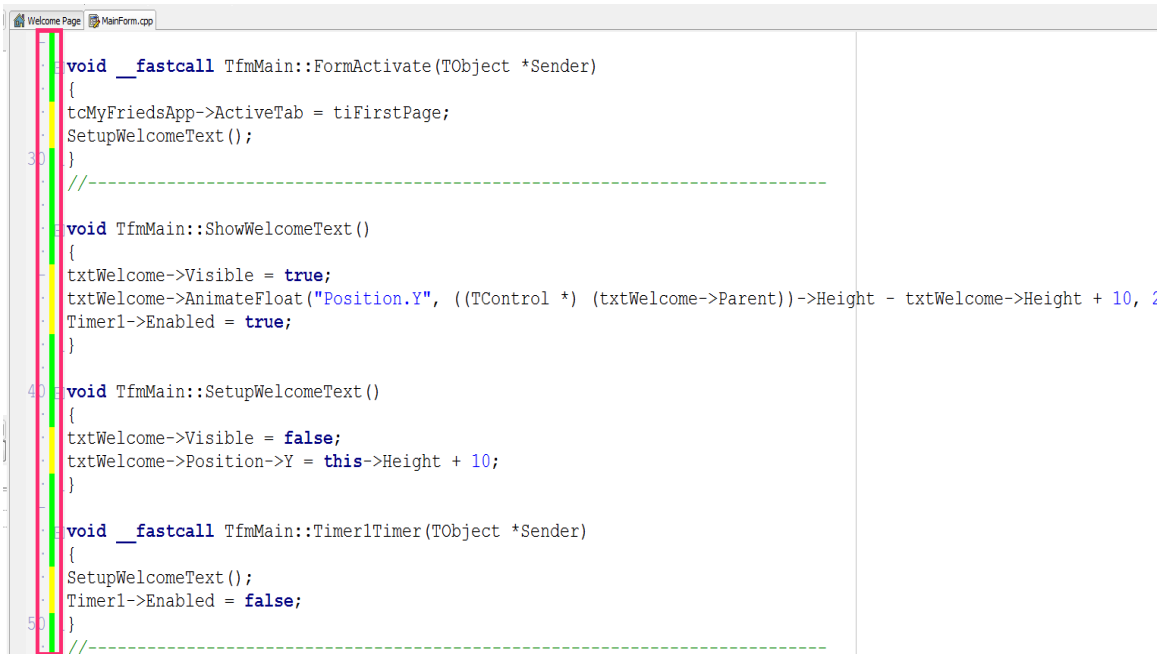
```
001    void TfmMain::ShowWelcomeText()
002    {
003    txtWelcome->Visible = true;
004    txtWelcome->AnimateFloat("Position.Y", ((TControl *)
(txtWelcome->Parent))->Height - txtWelcome->Height + 10, 2);
005    Timer1->Enabled = true;
006    }
```

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    void __fastcall TfmMain::Timer1Timer(TObject *Sender)
002    {
003    txtWelcome->AnimateFloat("Position.Y", this->Height + 10, 2);
004    Timer1->Enabled = false;
005    }
```

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 行再次呼叫 AnimateFloat()方法把 TText 元件回復成初始化的位置，004 行再停止 TTimer 元件的運作。

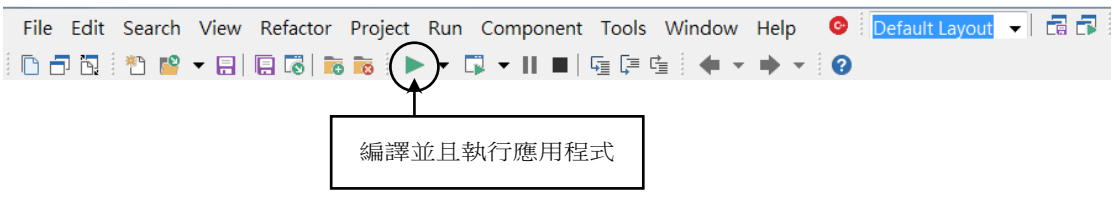
現在您的程式碼頁次應該看起來類似如下所示:



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』快捷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 的設備中看到下面的 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 C++Builder For 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 C++Builder For 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:

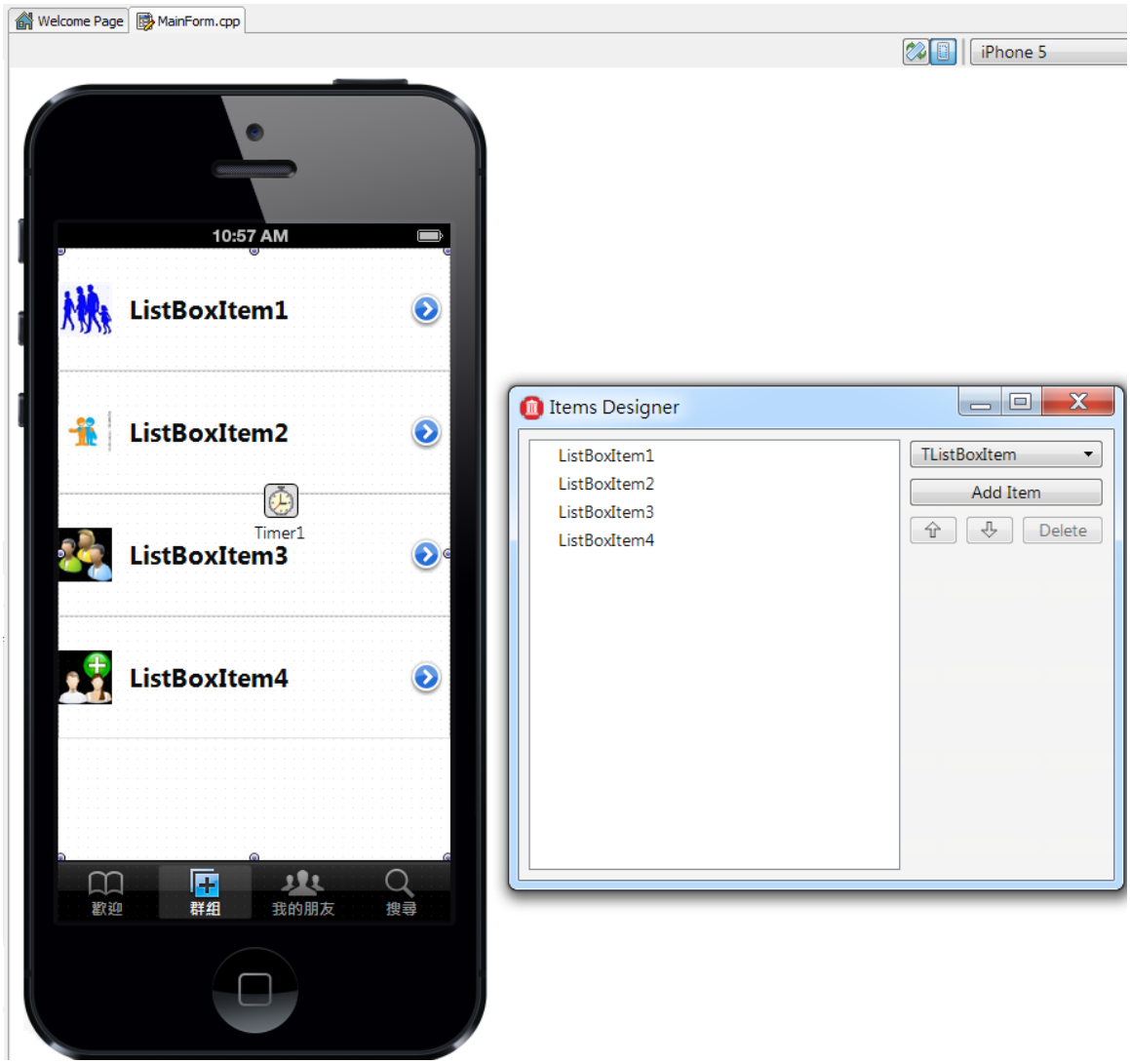
當然您也可以把程式區塊往左移動，您只要按下『Ctrl+Shift+U』就可以把整個程式區塊往左方移動。

同時按下 Ctrl+Shift+I 按鍵把程式區塊往右移動

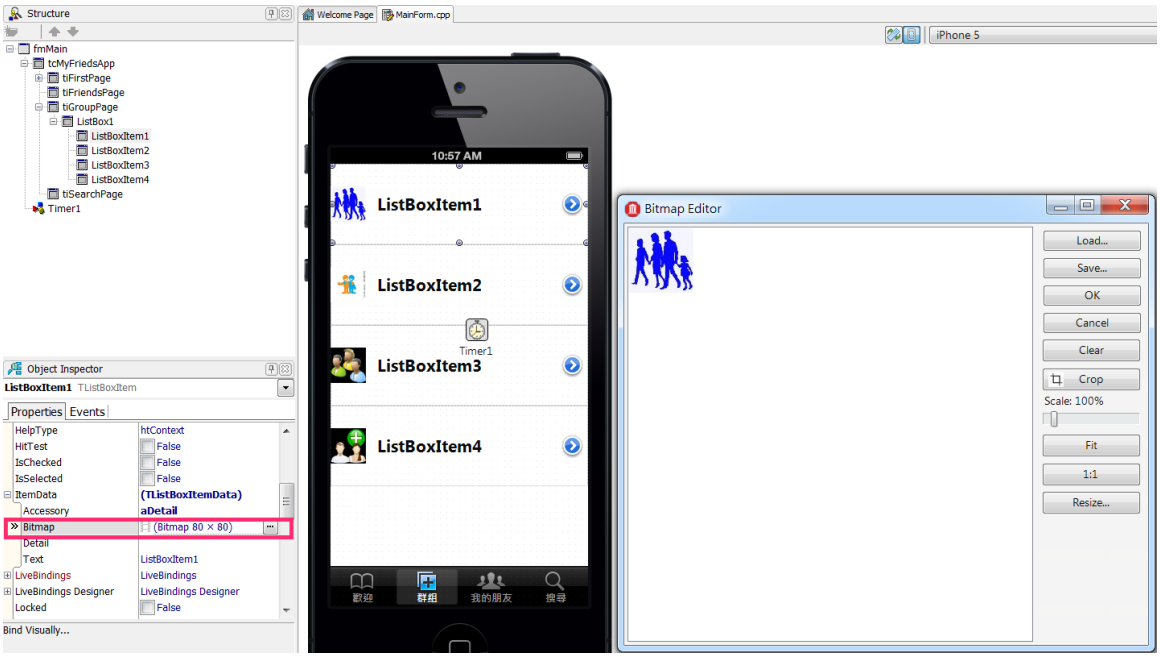
現在讓我們繼續開發這個範例應用程式，讓它更有趣一點。

4-2 儲存/切換桌面設定

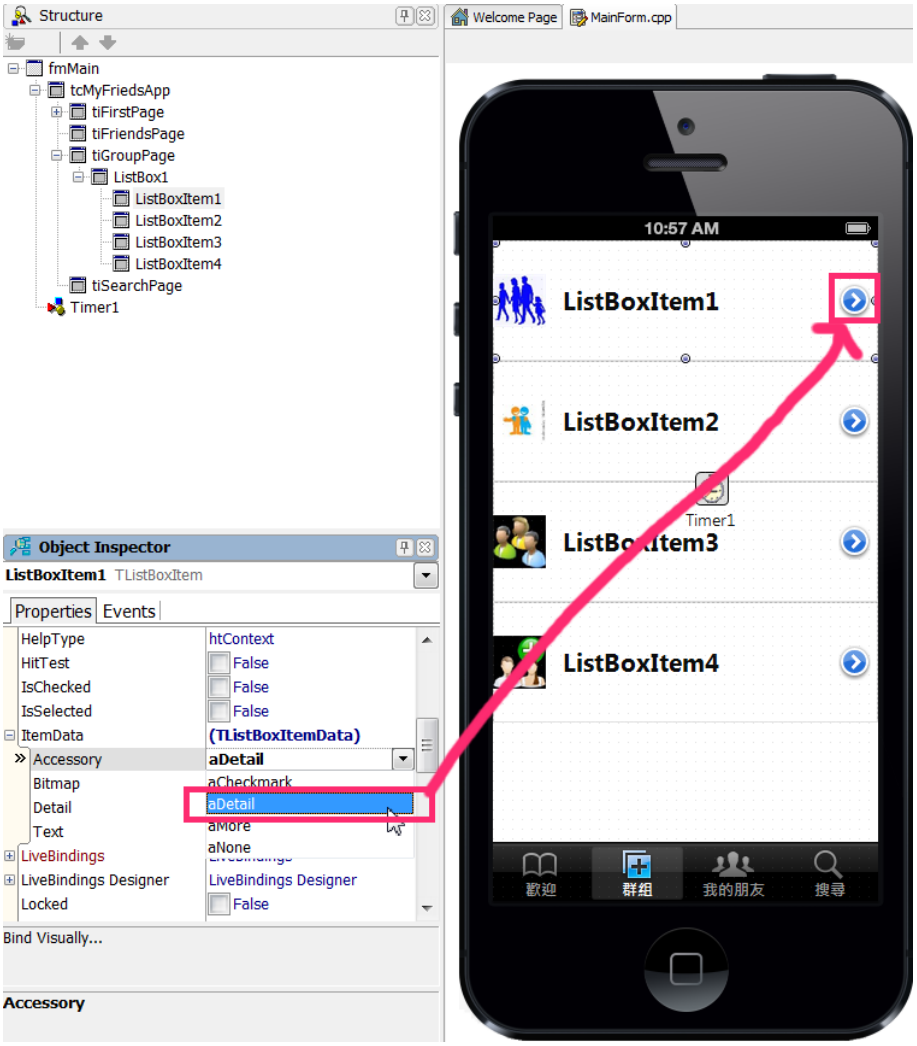
回到範例 HD Mobile FireMonkey 應用程式，點選主表單中的『群組』按鈕以開啟 TTabControl 元件的第 2 個頁面，從工具盤中拖曳 TListBox 元件到其中，設定 TListBox 的 Align 特性值為 alClient，接著點選滑鼠右鍵開啟 TListBox 的快顯功能表，從功能表中選擇『Items Editor…』選項以開啟 TListBox 的項目設計家，從右上方的下拉按鈕中選擇『TMetropolisUIListBoxItem』項目，再點選 4 次『Add Item』按鈕以便在 TListBox 元件中加入 4 個 TMetropolisUIListBoxItem 物件，如下所示:



接著關閉項目設計家對話盒，然後在主表單中選擇 TListBox 中的 TListBoxItem 物件，然後在物件檢視器點選它的 ItemData 下的 Bitmap 子特性，從下拉功能表中選擇 Edit…選項以開啟它的 Bitmap Editor 對話盒，點選其中的 Load 按鈕以載入圖像，再點選 OK 之後此圖像就會顯示在 TListBoxItem 物件中，如下所示，讀者可以如法炮製為 TListBox 中的每一個 TListBoxItem 物件都載入圖像。



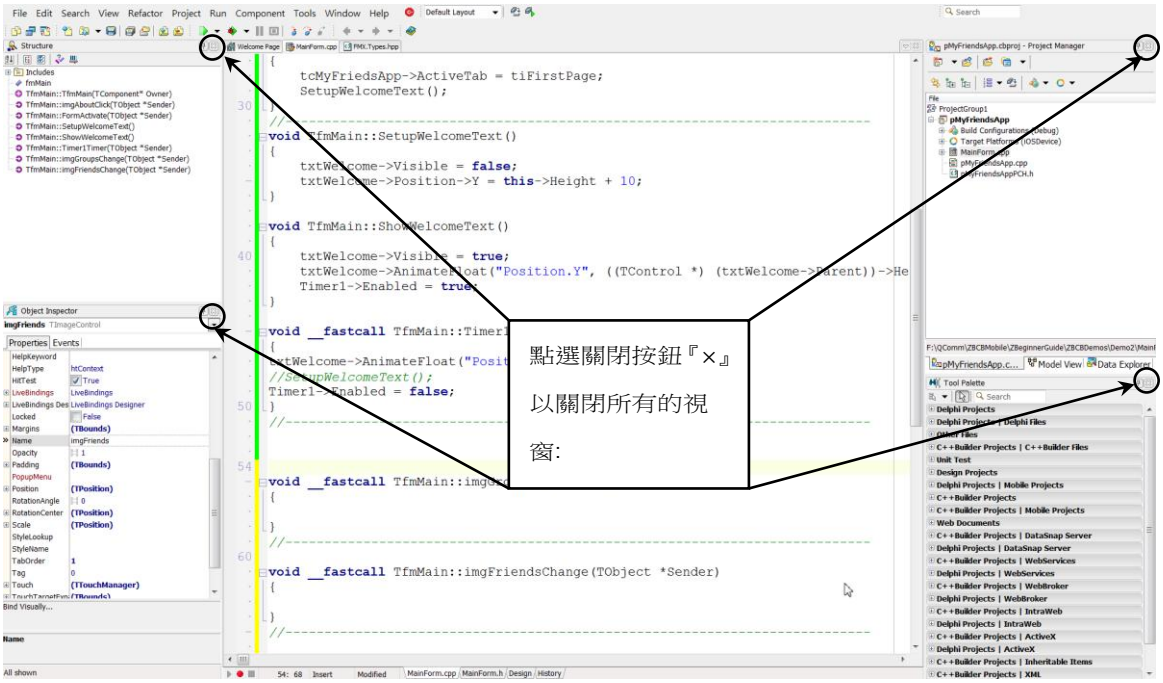
再接著讓我們為每一個 TListBoxItem 物件再加入一個指示符號以代表點選當執行此 App 時如果點選 TListBoxItem 物件的話，就可以顯示更為詳細的資訊。因此請點選 TListBoxItem 物件，在物件檢視器中點選它的 ItemData 特性左方的『+』號以展開 ItemData 的子特性，在其中有一個 Accessory 子特性，請點選它，從下拉盒中再選擇『aDetail』特性值，如下所示。如此一來我們就可以在 TListBoxItem 物件最右方加入一個『  』符號:



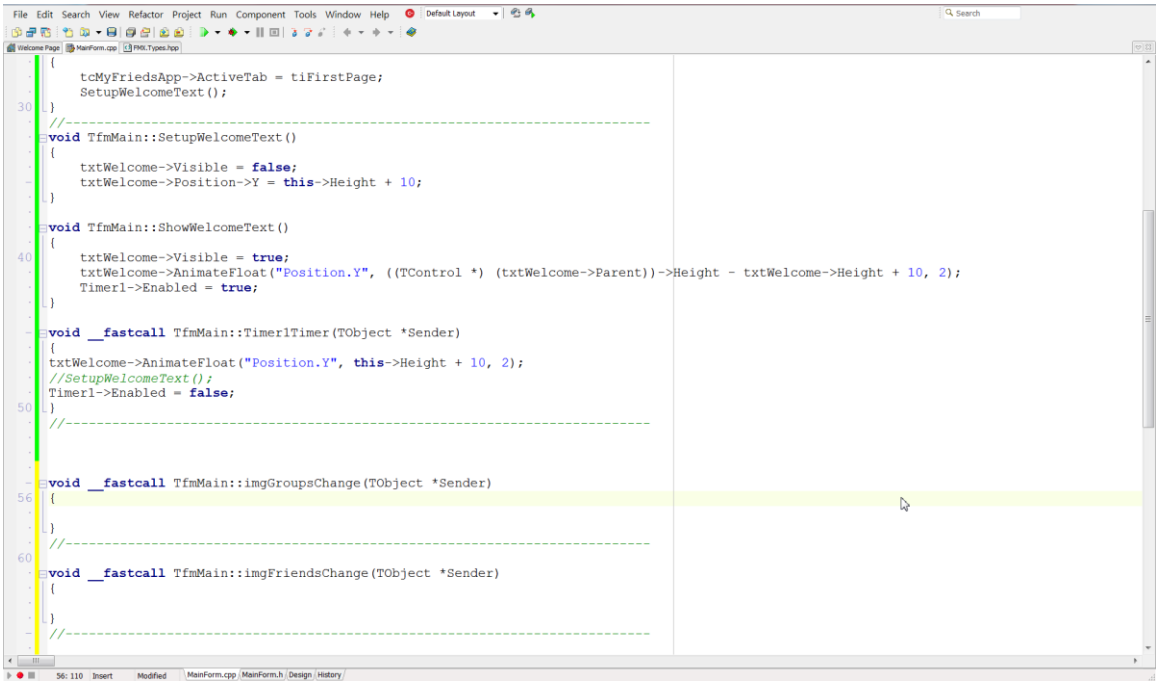
回到 TTabControl 元件的第一個頁面，為最左方和中間的圖像在『Events』頁次分別建立它們的 OnClick 事件處理函式:

```
void __fastcall TfmMain::imgGroupsChange(TObject *Sender)
{


}
//---------------------------------------------------------------------------
void __fastcall TfmMain::imgFriendsChange(TObject *Sender)
{


}
```

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，接著如下圖所示單擊『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：

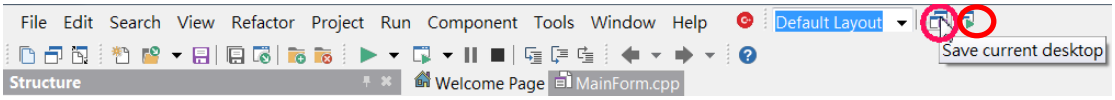


現在應該如下圖只剩下編輯器視窗:



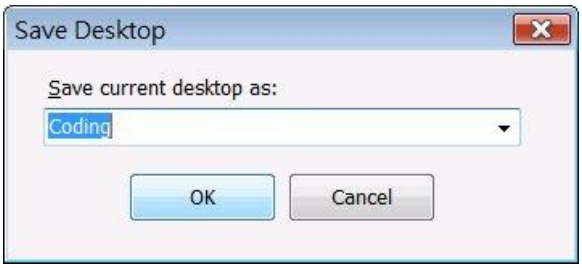
當您在專心撰寫程式碼時，可能不希望看到其他不相關的視窗，因此可以使用上面說明的方法只使用編輯器來撰寫程式碼，但每次都需要關閉 4 個視窗非常的麻煩，因此您可以把現在只使用編輯器來撰寫程式碼的桌面組態儲存起來，那麼當下次您只需要編輯器時，就只要選擇這個組態就可以把整合發展環境回復成現在的組態。

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



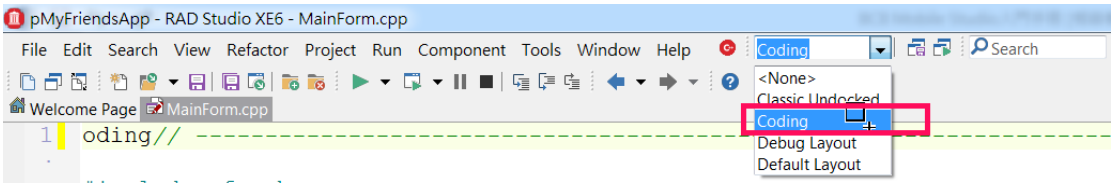
儲存目前桌面的設定組態

接著 C++Builder 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於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

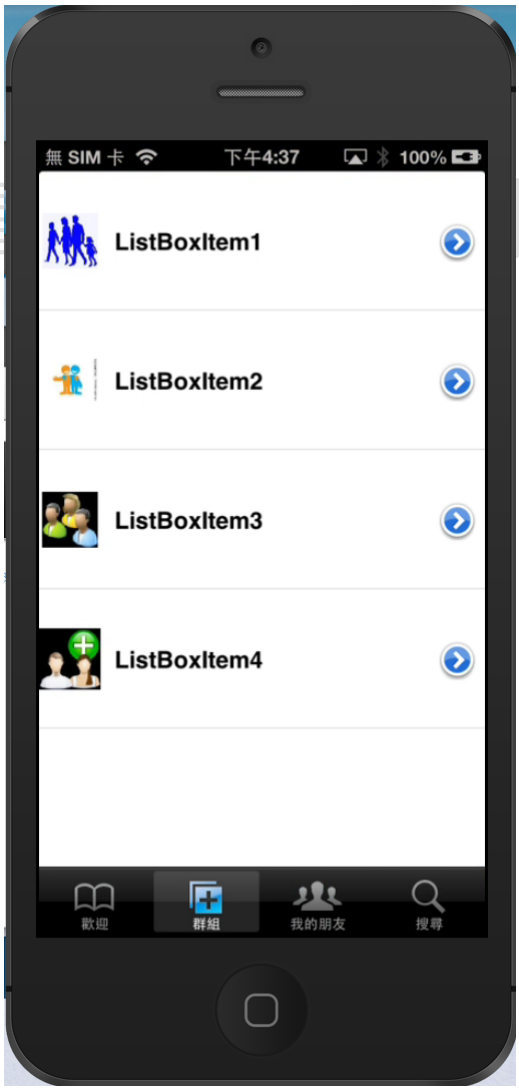
請試著在此下拉盒中選擇不同的組態，例如先選擇『Default Layout』，您會發現此時整合發展環境回復到一開始 5 個不同的視窗都顯示的組態，再選擇您儲存的『Coding』組態，那麼整合發展環境又會回復到只開啟編輯器的組態了。下面的表格說明了下拉盒中不同組態的意義:

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
<Classic Undocked>	使用 C++Builder 5~7 經典的桌面組態
<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

在在程式碼編輯器中為前面建立的兩個 OnClick 事件處理函式實作如下的程式碼，在這兩個 OnClick 事件處理函式中我們只是切換目前顯示的頁面:

```
void __fastcall TfmMain::imgFriendsClick(TObject *Sender)
{
  tcMyFriedsApp->ActiveTab = tiFriendsPage;
}

void __fastcall TfmMain::imgGroupsClick(TObject *Sender)
{
  tcMyFriedsApp->ActiveTab = tiGroupPage;
}
```

在 IDE 中執行此範例 App，它就會執行在 iOS 設備中，如果我們點選第一個頁面中不同的圖像就可以看到範例 App 可以顯示不同的內容，類似如下所示:

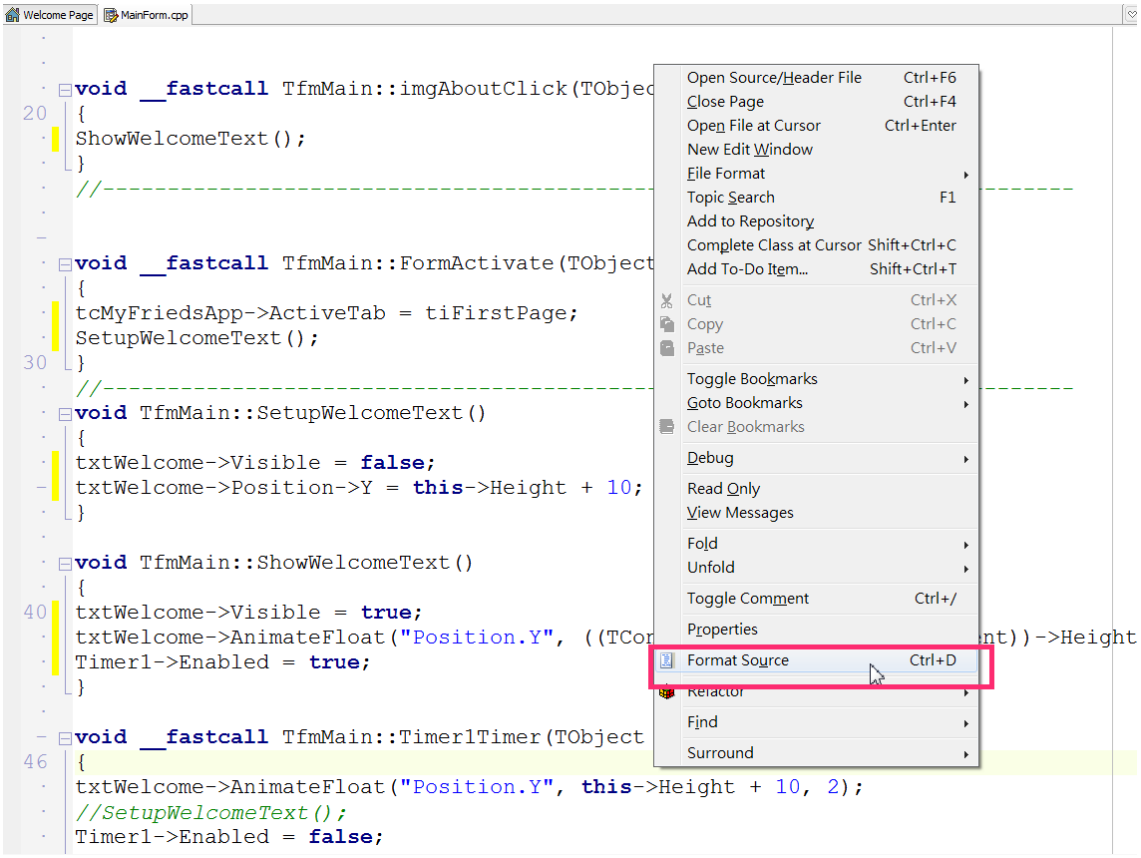


4-3 原始碼格式化

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。

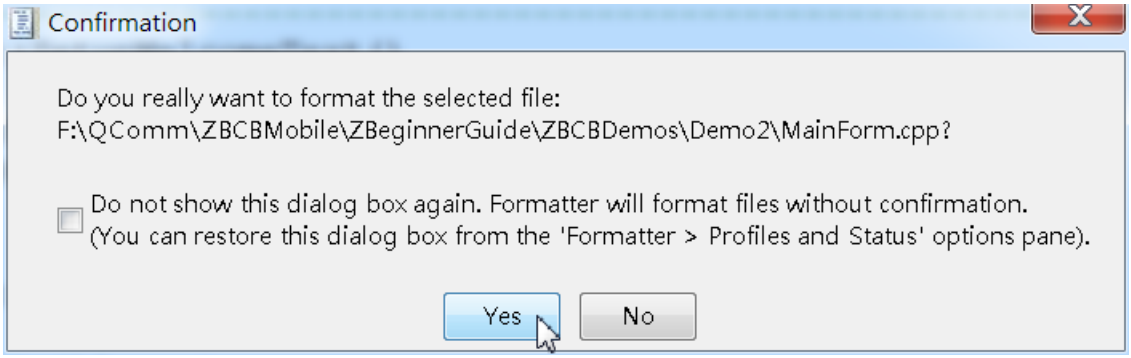
```
void __fastcall TfmMain::imgAboutClick(TObject *Sender)
{
ShowWelcomeText();
}
//---------------------------------------------------------------------------

void __fastcall TfmMain::FormActivate(TObject *Sender)
{
tcMyFriedsApp->ActiveTab = tiFirstPage;
SetupWelcomeText();
}
//---------------------------------------------------------------------------
void TfmMain::SetupWelcomeText()
{
txtWelcome->Visible = false;
txtWelcome->Position->Y = this->Height + 10;
}

void TfmMain::ShowWelcomeText()
{
txtWelcome->Visible = true;
txtWelcome->AnimateFloat("Position.Y", ((TControl *) (txtWelcome->Parent))->Height
Timer1->Enabled = true;
}
```

請回到編輯器中剛才實作 imgcAboutClick 的程序中，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

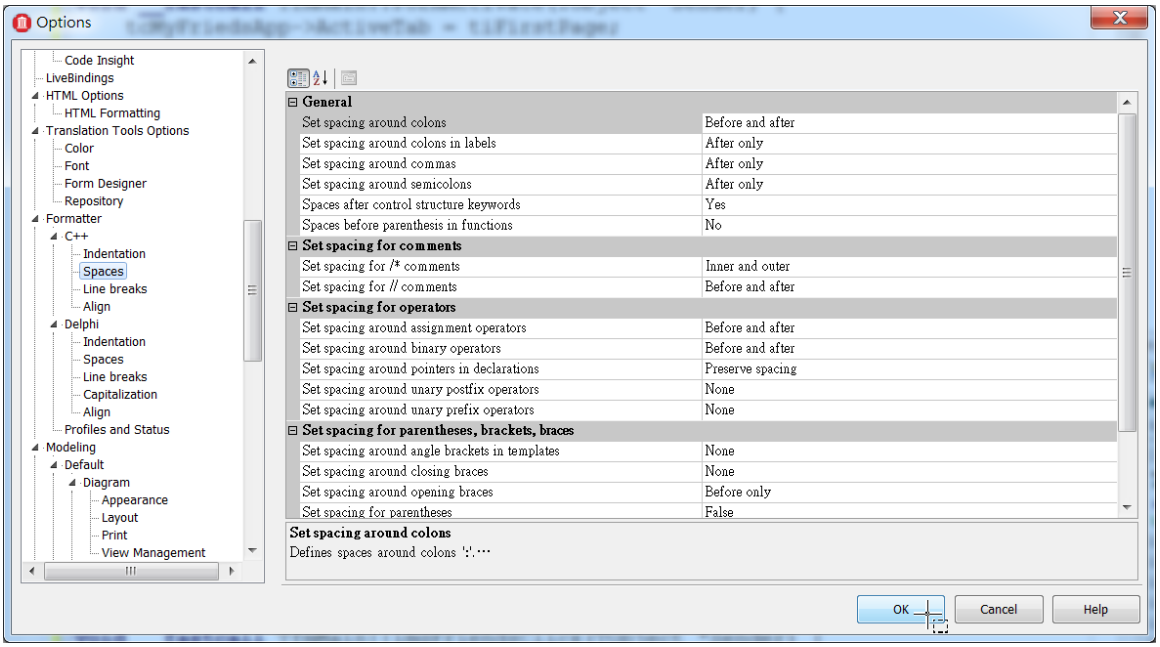
接著 IDE 會顯示下面的對話盒詢問您是否確定要格式化原始程式碼，請點選『Yes』按鈕，或是順便勾選對話盒中的勾選盒，避免每次要格式化原始程式碼是 IDE 都會詢問您。



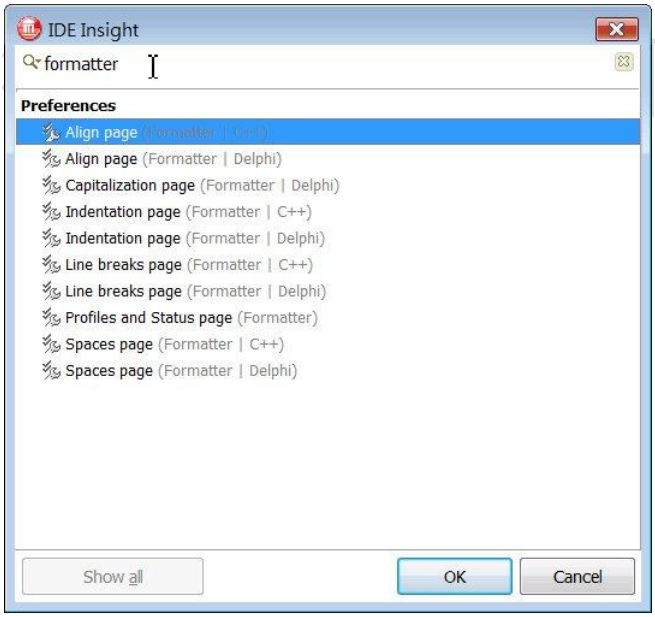
點選『Yes』按鈕之後，IDE 便會開始格式化您的原始程式碼，下圖就是格式化之後的結果，您可以看到程式碼風格好多了，也更容易明瞭，當然您也可以養成良好的習慣，在撰寫程式碼時就使用類似的風格。

```
void __fastcall TfmMain::imgAboutClick(TObject *Sender) {
    ShowWelcomeText();
}
// ---------------------------------------------------------------------------

void __fastcall TfmMain::FormActivate(TObject *Sender) {
    tcMyFriedsApp->ActiveTab = tiFirstPage;
    SetupWelcomeText();
}

// ---------------------------------------------------------------------------
void TfmMain::SetupWelcomeText() {
    txtWelcome->Visible = false;
    txtWelcome->Position->Y = this->Height + 10;
}

void TfmMain::ShowWelcomeText() {
    txtWelcome->Visible = true;
    txtWelcome->AnimateFloat("Position.Y",
        ((TControl *)(txtWelcome->Parent))->Height - txtWelcome->Height +
        10, 2);
    Timer1->Enabled = true;
}

void __fastcall TfmMain::Timer1Timer(TObject *Sender) {
    txtWelcome->AnimateFloat("Position.Y", this->Height + 10, 2);
    // SetupWelcomeText();
    Timer1->Enabled = false;
}
// ---------------------------------------------------------------------------

void __fastcall TfmMain::imgFriendsClick(TObject *Sender) {
    tcMyFriedsApp->ActiveTab = tiFriendsPage;
```


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 | C++Builder 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:



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:



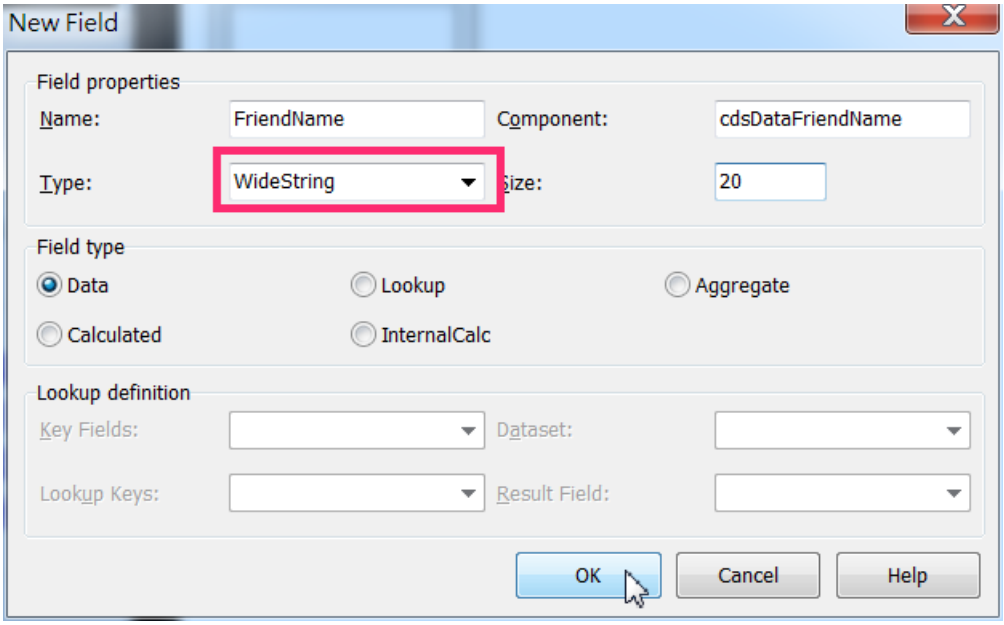
現在您就可以試著改變一些設定然後再次重新格式化您的原始程式碼，看看改變這些設定之後有什麼效果了。

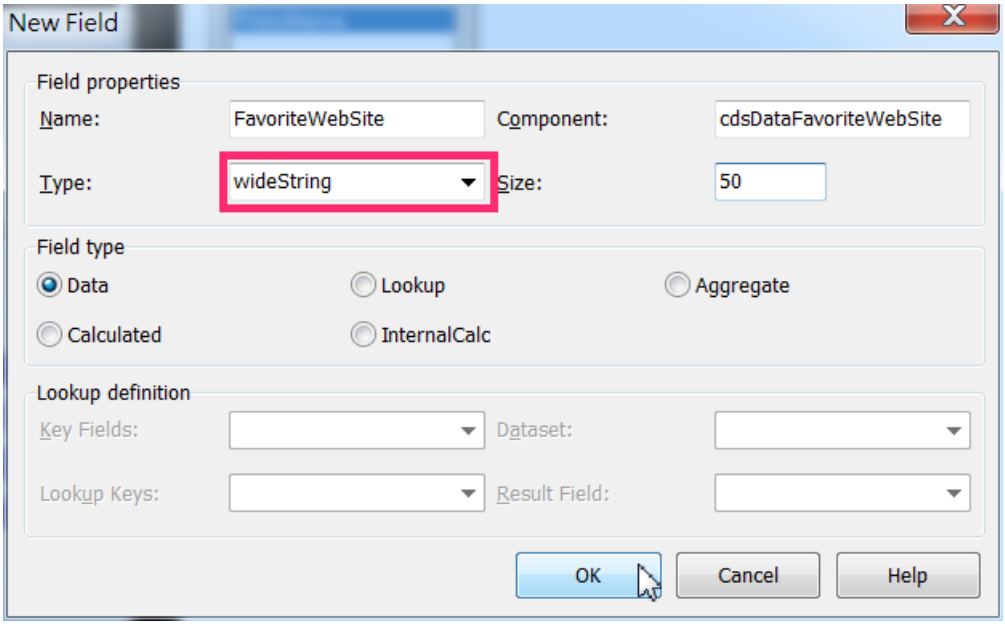
讓我們為這個範例 iOS 加入一些更為複雜和有趣的功能，那就是開始加入存取資料的能力。如果您是使用其他 iOS 開發工具而需要為 iOS App 加入資料處理的功能的話，您需要花費許多的時間去撰寫大量的程式碼，但使用 C++Builder for iOS 卻非常的簡單。

首先請在主表單中加入 TClientDataSet 設定它的 Name 特性值為 cdsData，加入 TDataSource 元件，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:





請注意，在 iOS 平台中如果要使用中交資料的話，那麼必須使用 WideString 欄位型態。

現在 cdsData 元件就擁有 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():


```
void TfmMainForm::CreateGroupDataSet()
{
  TIndexDef *pIndex;

  pIndex = cdsMyData->IndexDefs->AddIndexDef();
  pIndex->Fields = "FriendName";
  pIndex->Name = "idxFriendName";

  cdsMyData->CreateDataSet();
  cdsMyData->Active = true;
}
void TdmDemoApp::FillGroupDataSet()
{
  cdsMyData->Insert();
  cdsMyData->FieldByName("FriendName")->Value = “Jackson Wang”;
  cdsMyData->FieldByName("FavoriteWebSite")->Value =
“www.youtube.com”;
  cdsMyData->Post();
```

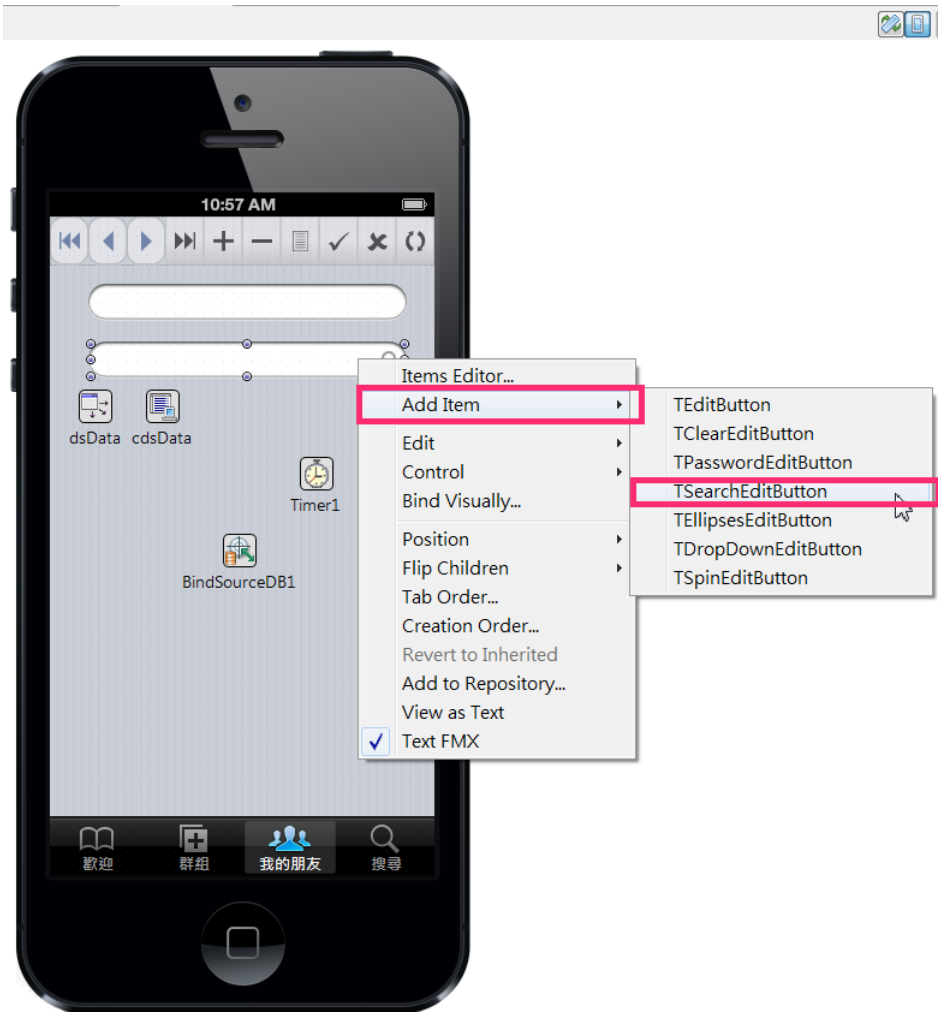
```
  cdsMyData->Insert();
  cdsMyData->FieldByName("FriendName")->Value = "Hua Lee";
  cdsMyData->FieldByName("FavoriteWebSite")->Value =
"www.embarcadero.com";
  cdsMyData->Post();

  cdsMyData->Insert();
  cdsMyData->FieldByName("FriendName")->Value = "DongHseng Cheng";
  cdsMyData->FieldByName("FavoriteWebSite")->Value = "www.msn.com.tw";
  cdsMyData->Post();

  cdsMyData->Insert();
  cdsMyData->FieldByName("FriendName")->Value = "YuHei Chu";
  cdsMyData->FieldByName("FavoriteWebSite")->Value =
"C++Builder.ktop.com.tw";
  cdsMyData->Post();

  cdsMyData->Insert();
  cdsMyData->FieldByName("FriendName")->Value = "老夫子";
  cdsMyData->FieldByName("FavoriteWebSite")->Value =
"www.google.com.tw";
  cdsMyData->Post();
}
```

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。

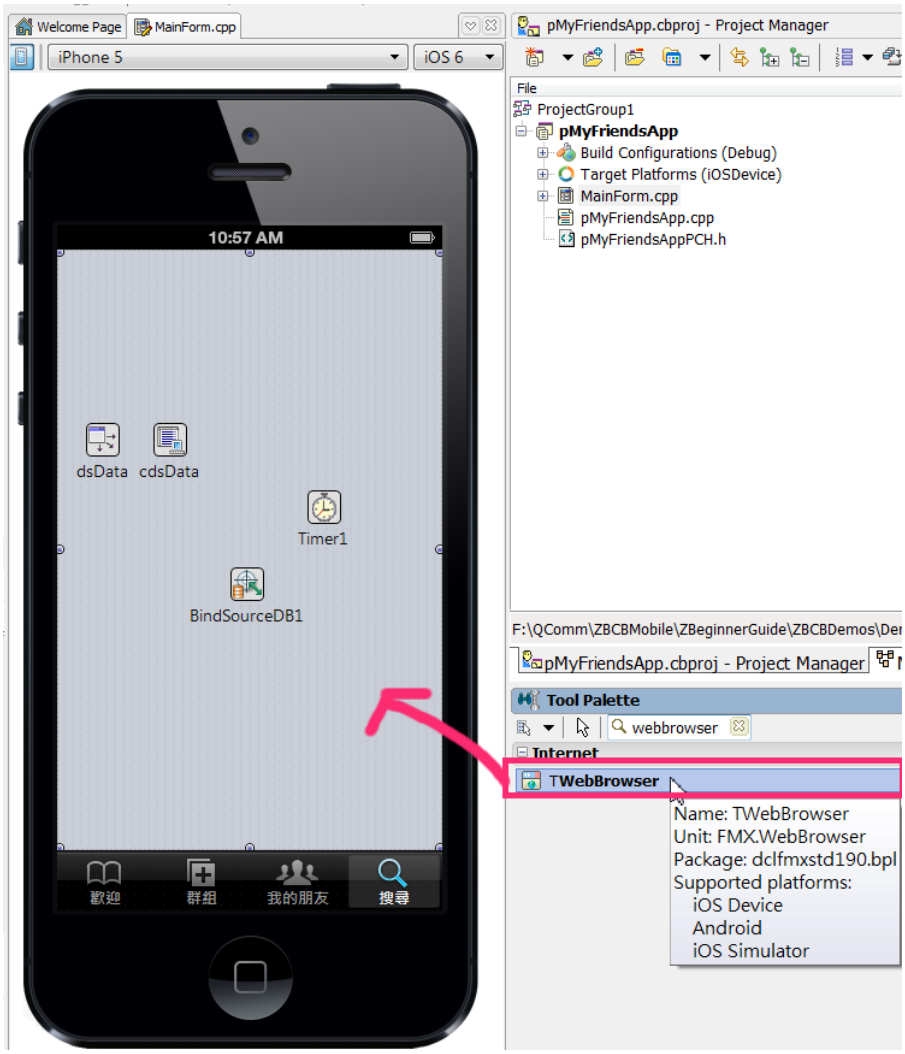
```
void __fastcall TfmMain::FormCreate(TObject *Sender)
{
  CreateGroupDataSet();
  FillGroupDataSet();
}
```

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

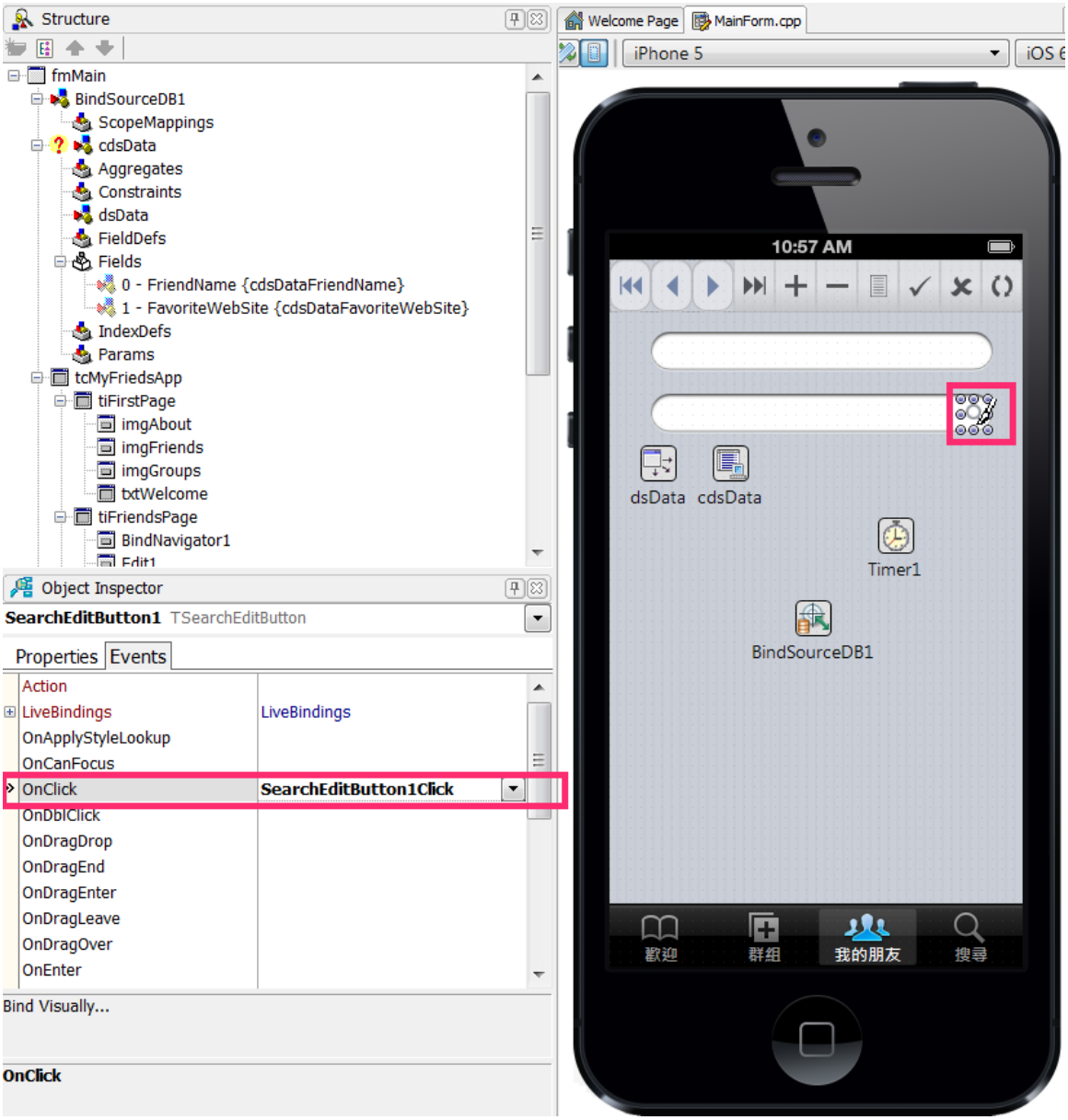
再讓我們為剛才加入的第二個 TEdit 元件加入一個搜尋按鈕，請點選第二個 TEdit 元件，再點選滑鼠右鍵，從快顯功能表中選擇『Add Item』選項，再從其中選擇『TSearchEditButton』，如下所示:



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 元件的 Align 特性值為 alClient，如下所示:



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




```
void __fastcall TfmMain::SearchEditButton1Click(TObject *Sender)
{
  //
}
```

接下來我們就可以開始試著繫結 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

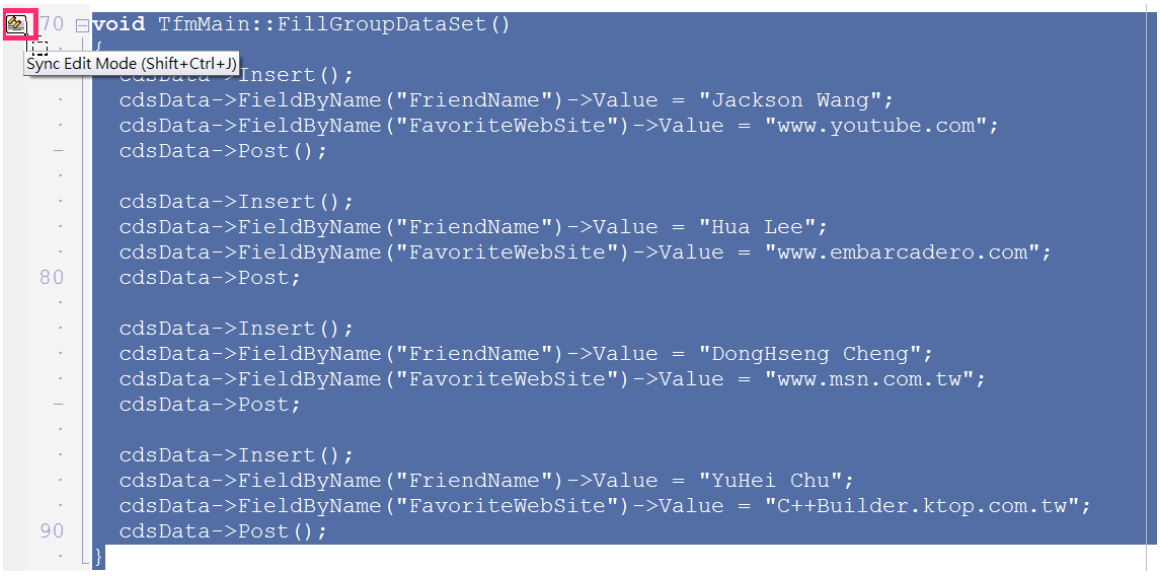
在繼續改善程式碼之前，您需要學習一個非常有用的編輯器技巧，那就是所謂的 SyncEdit。

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:

```
void TfmMain::FillGroupDataSet()
{
  cdsData->Insert();
  cdsData->FieldByName("FriendName")->Value = "Jackson Wang";
  cdsData->FieldByName("FavoriteWebSite")->Value = "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:



選擇程式碼並且啟動 SyncEdit 功能

現在請使用滑鼠雙擊編輯器中圖像，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:

```
70  void TfmMain::FillGroupDataSet()
    {
72    cdsData->Insert();
      cdsData->FieldByName("FriendName")->Value = "Jackson Wang";
      cdsData->FieldByName("FavoriteWebSite")->Value = "www.youtube.com";
      cdsData->Post();

      cdsData->Insert();
      cdsData->FieldByName("FriendName")->Value = "Hua Lee";
      cdsData->FieldByName("FavoriteWebSite")->Value = "www.embarcadero.com";
80    cdsData->Post;

      cdsData->Insert();
      cdsData->FieldByName("FriendName")->Value = "DongHseng Cheng";
      cdsData->FieldByName("FavoriteWebSite")->Value = "www.msn.com.tw";
      cdsData->Post;

      cdsData->Insert();
      cdsData->FieldByName("FriendName")->Value = "YuHei Chu";
      cdsData->FieldByName("FavoriteWebSite")->Value = "C++Builder.ktop.com.tw";
90    cdsData->Post();
    }
```

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:

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

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

      cdsMyData.Insert;
110   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;
120   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。

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

4-5 待辦清單(To-Do List)

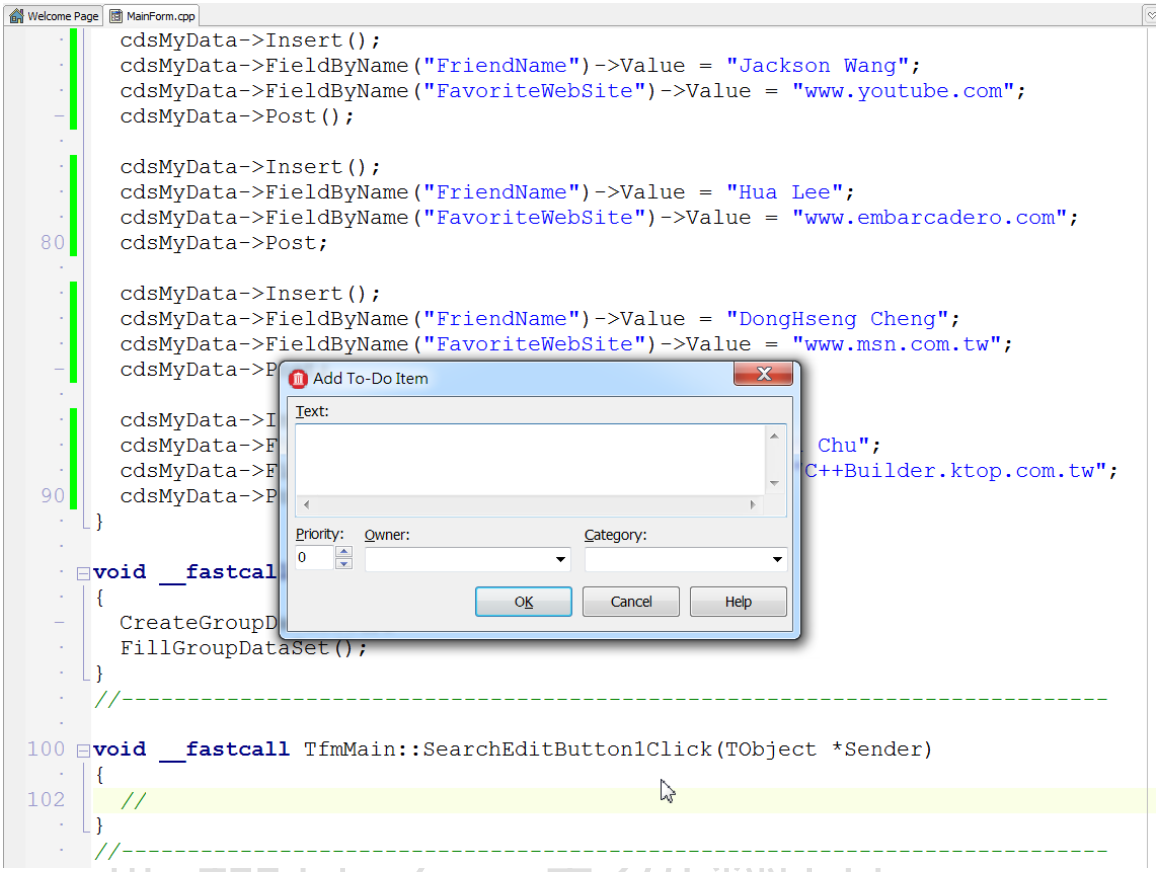
在撰寫程式碼時，您可以先在程式碼中列出待辦清單，接著一一的撰寫程式碼實現這些待辦清單，您也可以追蹤，管理程式碼中的待辦清單以便瞭解您是否完成了所有的待辦清單，以免遺漏應該完成的功能，這在撰寫大量的功能和程式碼時是非常實用的功能。

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。

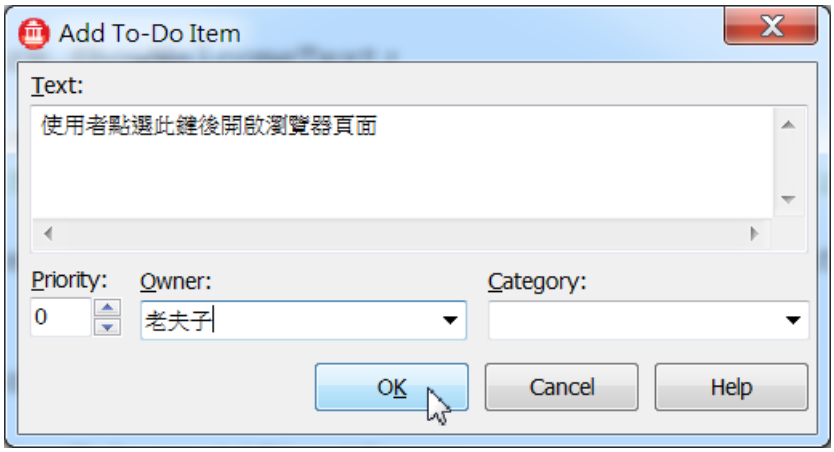
要在編輯器中加入待辦清單，您可以在程式碼中適當的地方同時按下 Shift+Ctrl+T，或是在編輯器中點選滑鼠右鍵，在快顯功能表中選擇『Add To-Do item...』選項。例如 TSearchEditButton 元件建立 OnClick 事件處理函式尚未實作，因此讓我們移動滑鼠到 SearchEditButton1Click 程序的第 1 行程式碼之上，同時按下 Shift+Ctrl+T 準備加入待辦清單，此時 IDE 就會顯示一個 Add To-Do Item 對話盒如下，請您輸入待辦清單的詳細資訊，對話盒中欄位的意義說明如下:

欄位	說明
Text	待辦清單說明文字
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 程序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:

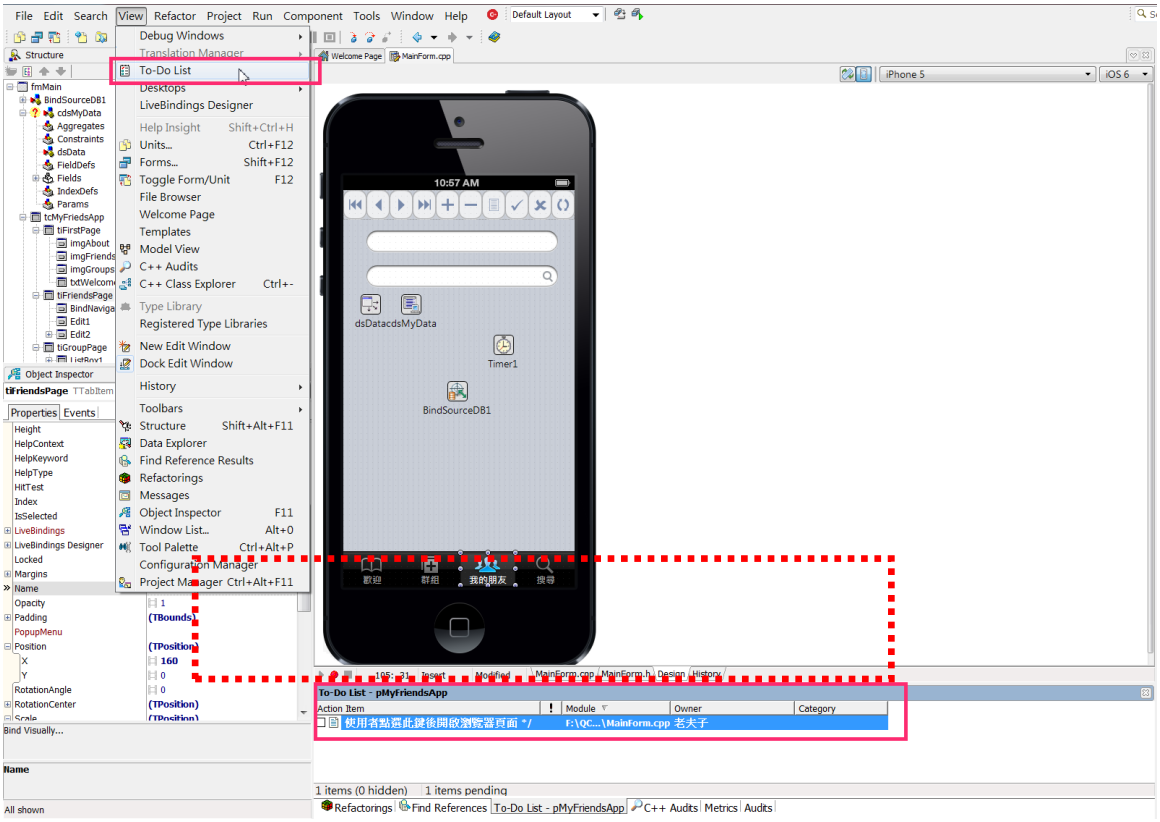


當您點選 Add To-Do Item 對話盒中的 OK 按鈕之後，您就可以在程式碼中看到 IDE 在您的程式碼中加入了如下的待辦清單注釋:

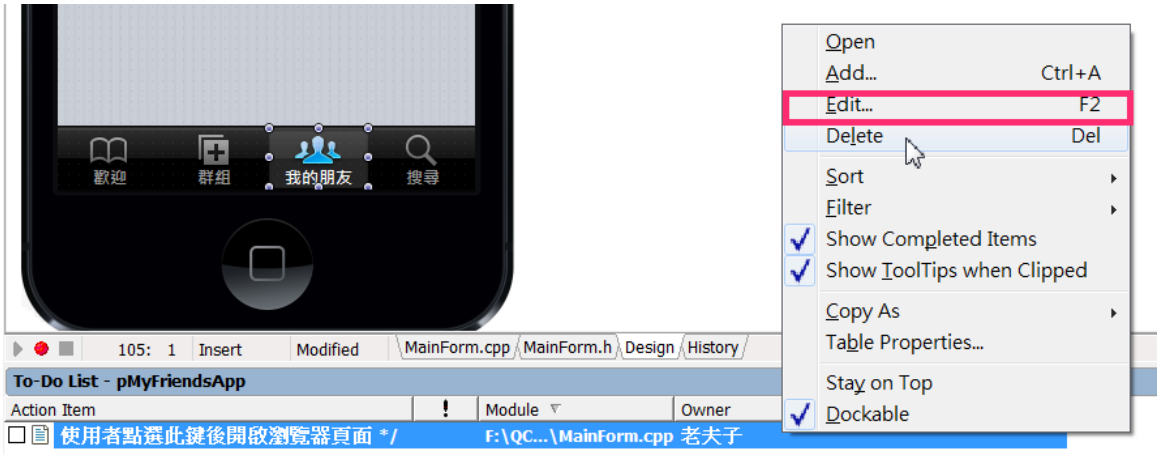
```
void __fastcall TfmMain::SearchEditButton1Click(TObject *Sender)
{
  ///* TODO -o老夫子 : 使用者點選此鍵後開啟瀏覽器頁面 */
```

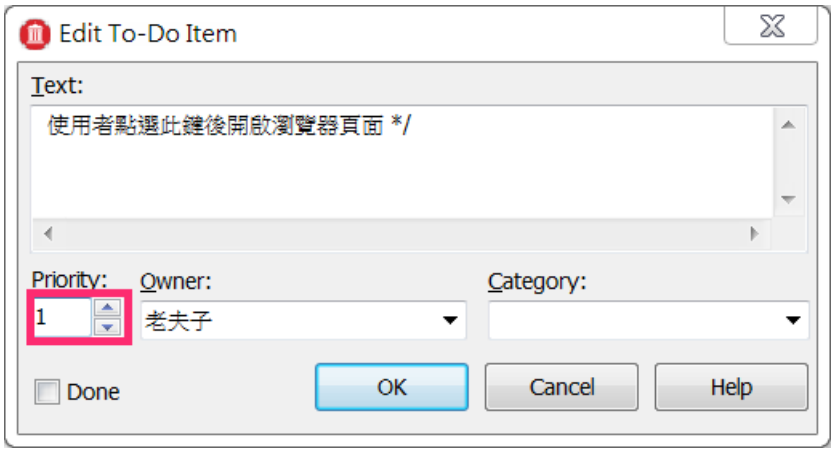
```
}
```

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，當您雙擊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:



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『**使用者點選此鍵後開啟瀏覽器頁面**』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。

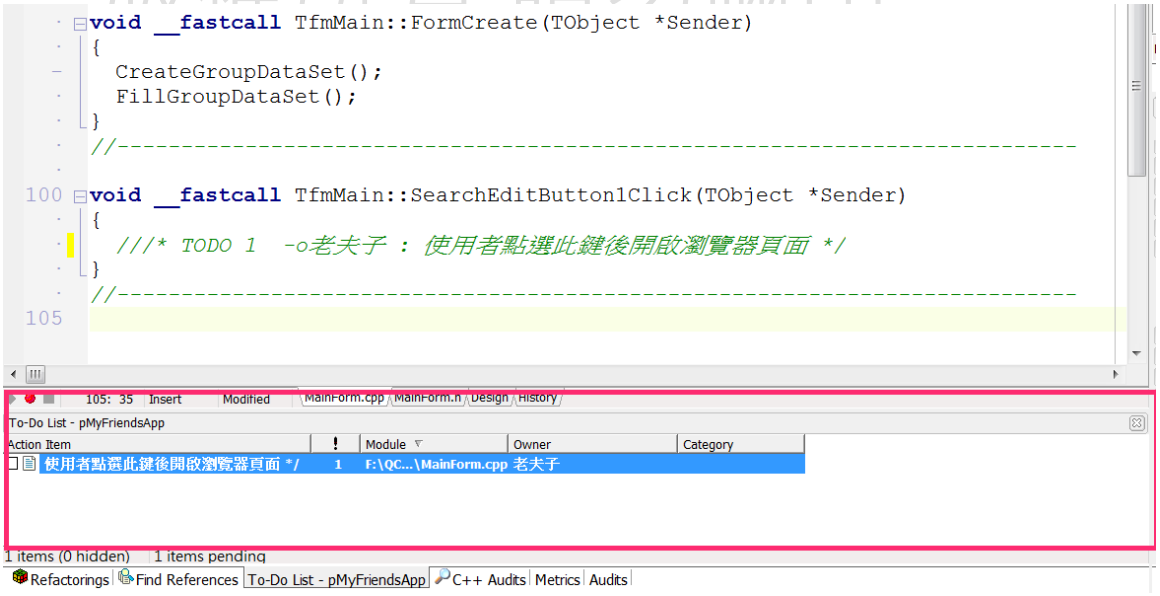




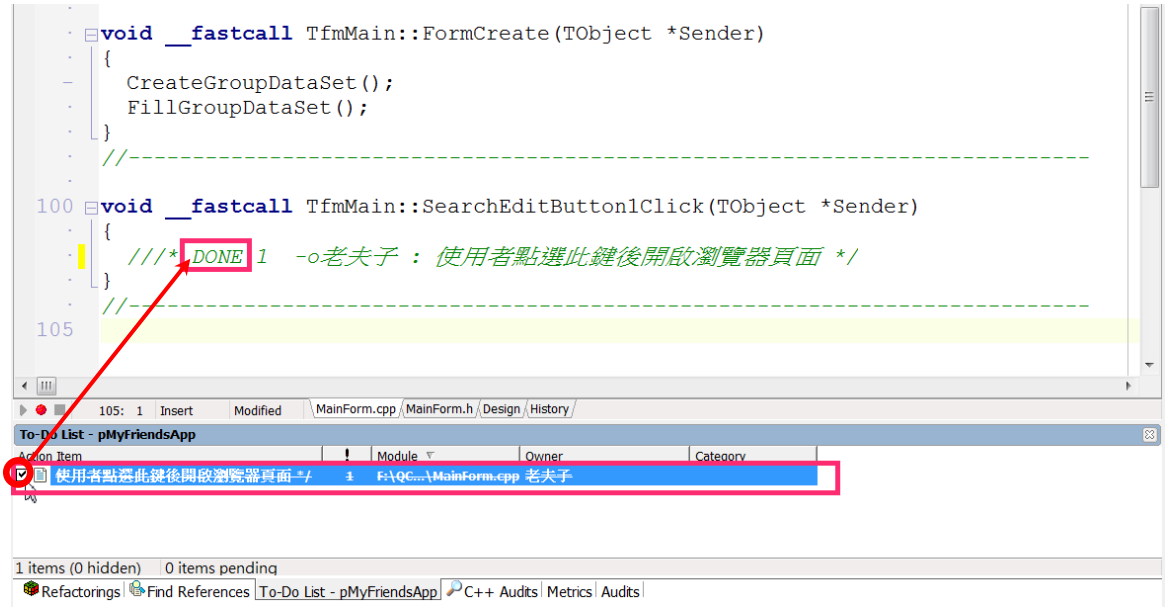
點選 OK 按鈕之後程式碼就會被修改如下:

```
void __fastcall TfmMain::SearchEditButton1Click(TObject *Sender)
{
  ///* TODO 1  -o 老夫子 : 使用者點選此鍵後開啟瀏覽器頁面 */
}
```

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清單:



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:



4-6 程式區塊註解

在 C++Builder for iOS 程式語言中，您可以使用『//』註解單行程式碼的意義，或是使用『/*...*/』來註解多行程式碼的意義。在 IDE 中如果您想註解多行的程式碼，您可以使用滑鼠選擇想註解的程式碼，然後同時按下『Ctrl+/』，那麼 IDE 就會自動幫您註解這些程式碼，如下所示:


```
void TfmMain::InsertData(const String sName, const String sWebSite)
{
//  cdsMyData->Insert();
//  cdsMyData->FieldByName("FriendName")->Value = sName;
//  cdsMyData->FieldByName("FavoriteWebSite")->Value = sWebSite;
//  cdsMyData->Post();
}
```

如果您想取消註解的程式碼，那麼同樣的您只要選擇已經註解程式碼，再同時按下『Ctrl+/』就可以取消選擇的程式碼的註解了。

4-7 程式碼瀏覽

當程式碼中的事件處理函式和方法愈來愈多時，您可能需要快速在不同的方法程式碼中瀏覽或是移動，您可以雙擊 IDE 左上方的專案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:

或是您可以使用下面的快捷鍵組合在不同的方法中快速移動:

鍵盤快捷鍵	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捷鍵。

4-8 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。

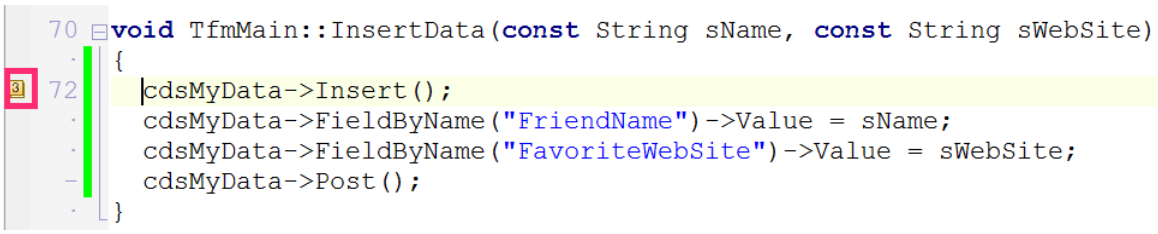
IDE 提供了最多 10 個書籤可讓您在程式碼中標註，一旦您在程式碼中設定了書籤，就可以藉由快捷鍵立刻在不同的書籤程式碼標註處移動。要在 IDE 中設定書籤，您可以使用 Ctrl+Shift+0，Ctrl+Shift+1 一直到 Ctrl+Shift+9 最多設定 10 個書籤。

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

您也可以使用 Ctrl+k+0 到 Ctrl+k+9 來設定書籤，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()方法的第 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籤:

接著再請您移動游標到 InsertData()方法的第 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籤:



現在您可以藉由同時按下 Ctrl+2 立刻把游標移動回 FillGroupDataSet()方法的第 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()方法的第 1 行程式碼處。

藉由使用 IDE 的書籤功能，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。

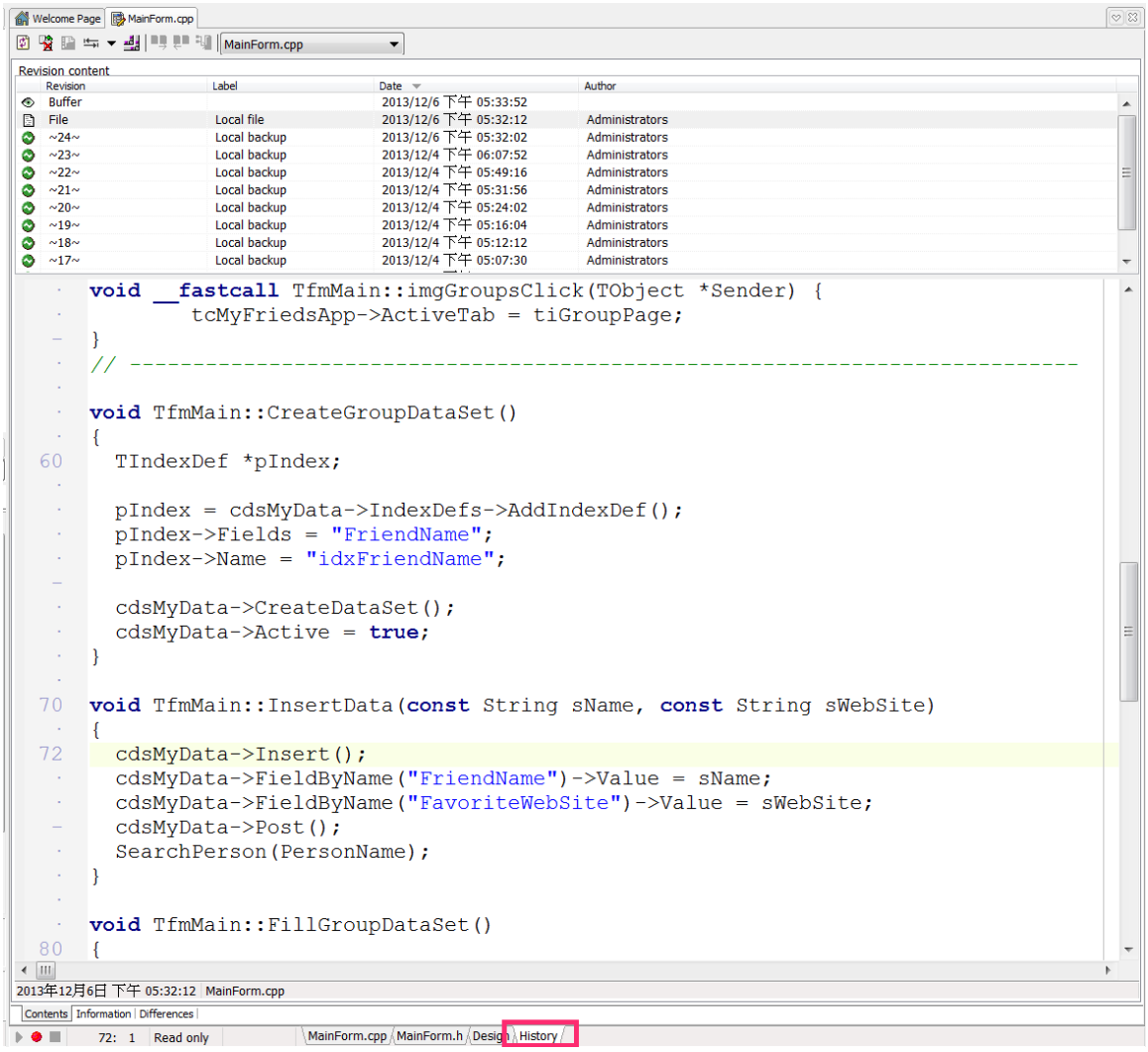
4-9 歷程管理員(History)

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，其中的檔案名稱比較特別，都是以『XXX.XXX.~數字~』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。

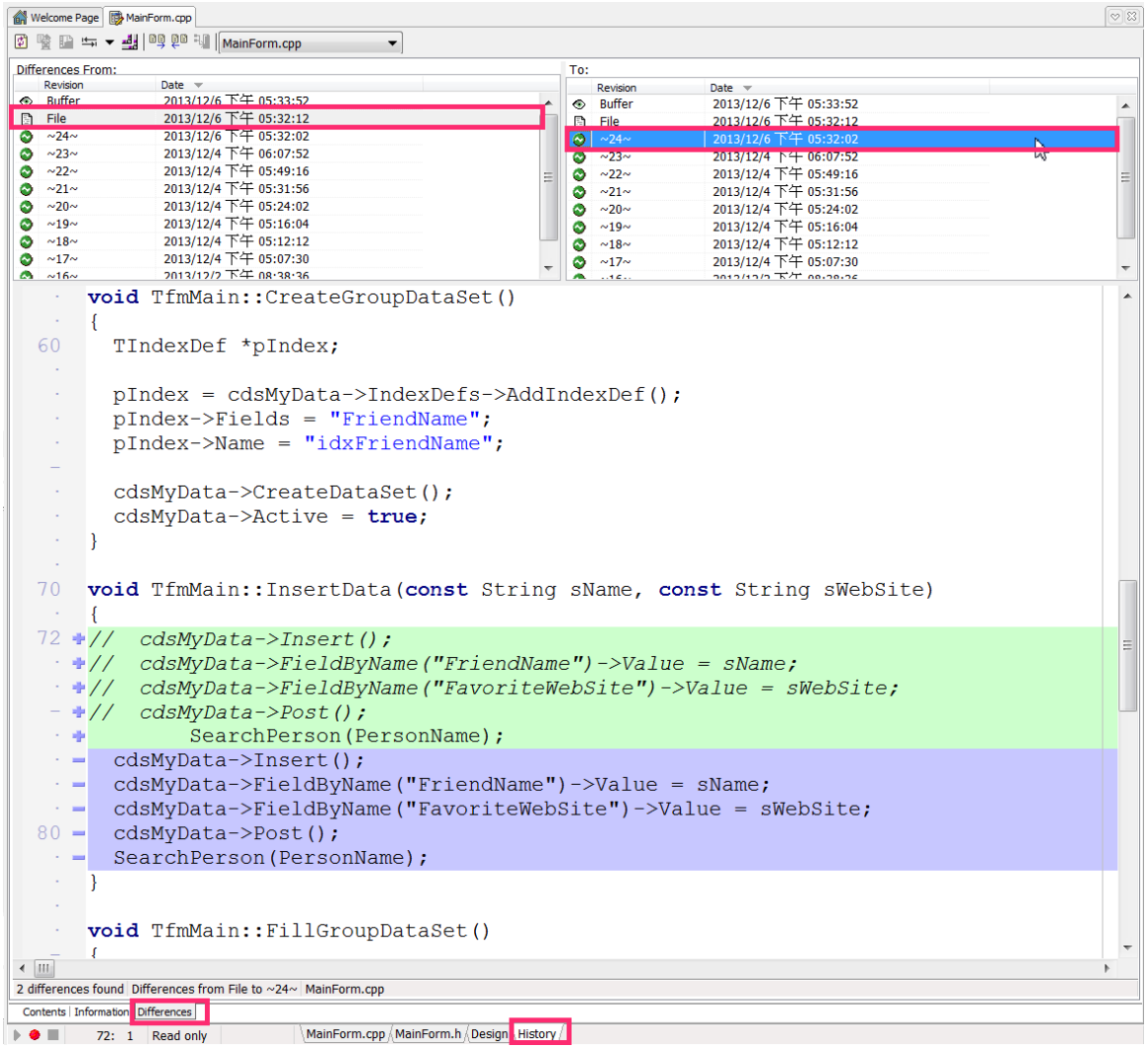
事實上當您在 IDE 中對專案中的檔案進行異動時並且儲存之後，IDE 便會在這個 History 子目錄中儲存一份版本，因此這個 History 子目錄中所有的檔案便是您對專案修改的歷程，也就是說 IDE 會在這個 History 子目錄提供一些類似版本控制軟體提供的功能。

> 如果您使用 C++Builder 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，請雙擊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:



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(~24~)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。

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

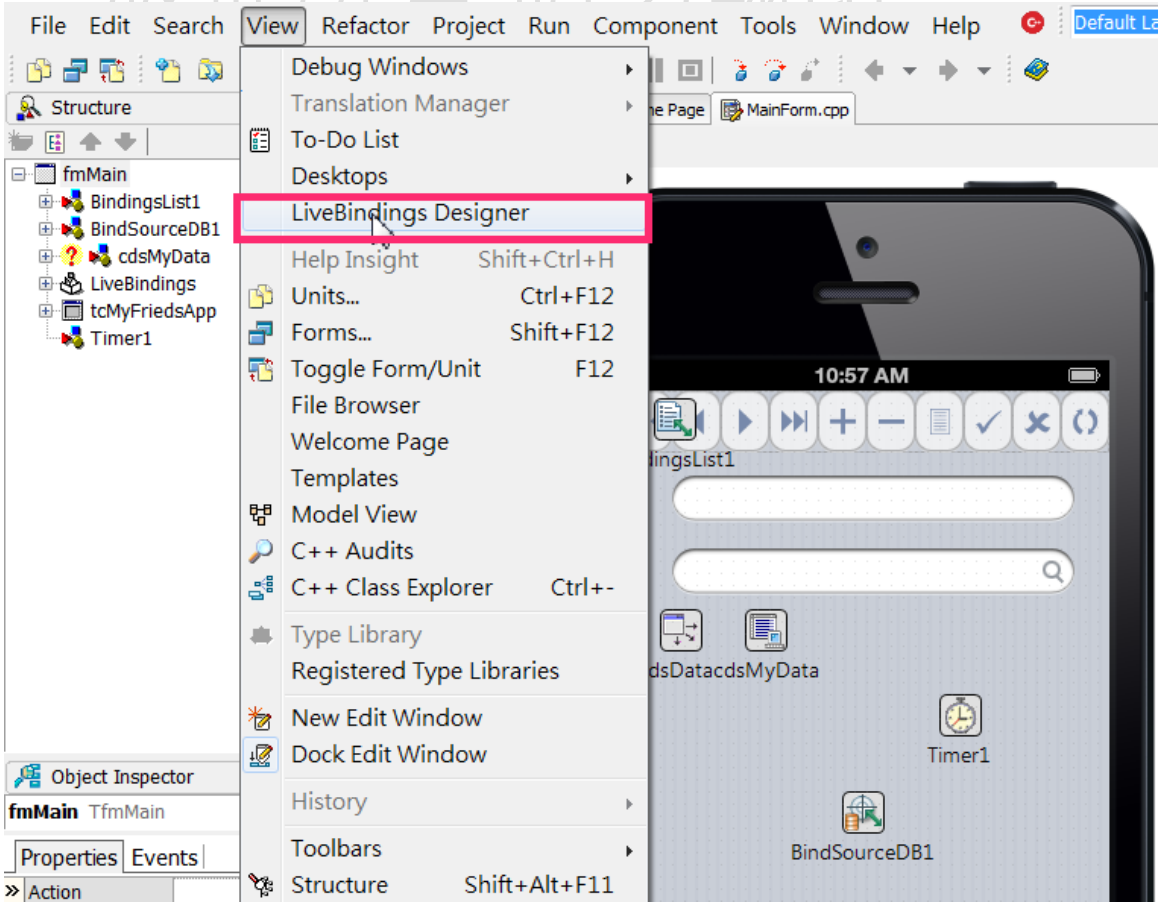
	目前存在於緩衝記憶體的版本，包含了尚未儲存的異動程式碼
	儲存在版本控制系統中的版本
	從控制系統中簽出(Check Out)的版本

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 功能繫結資料，最後再實作 SearchEditButton1Click 事件處理函式。

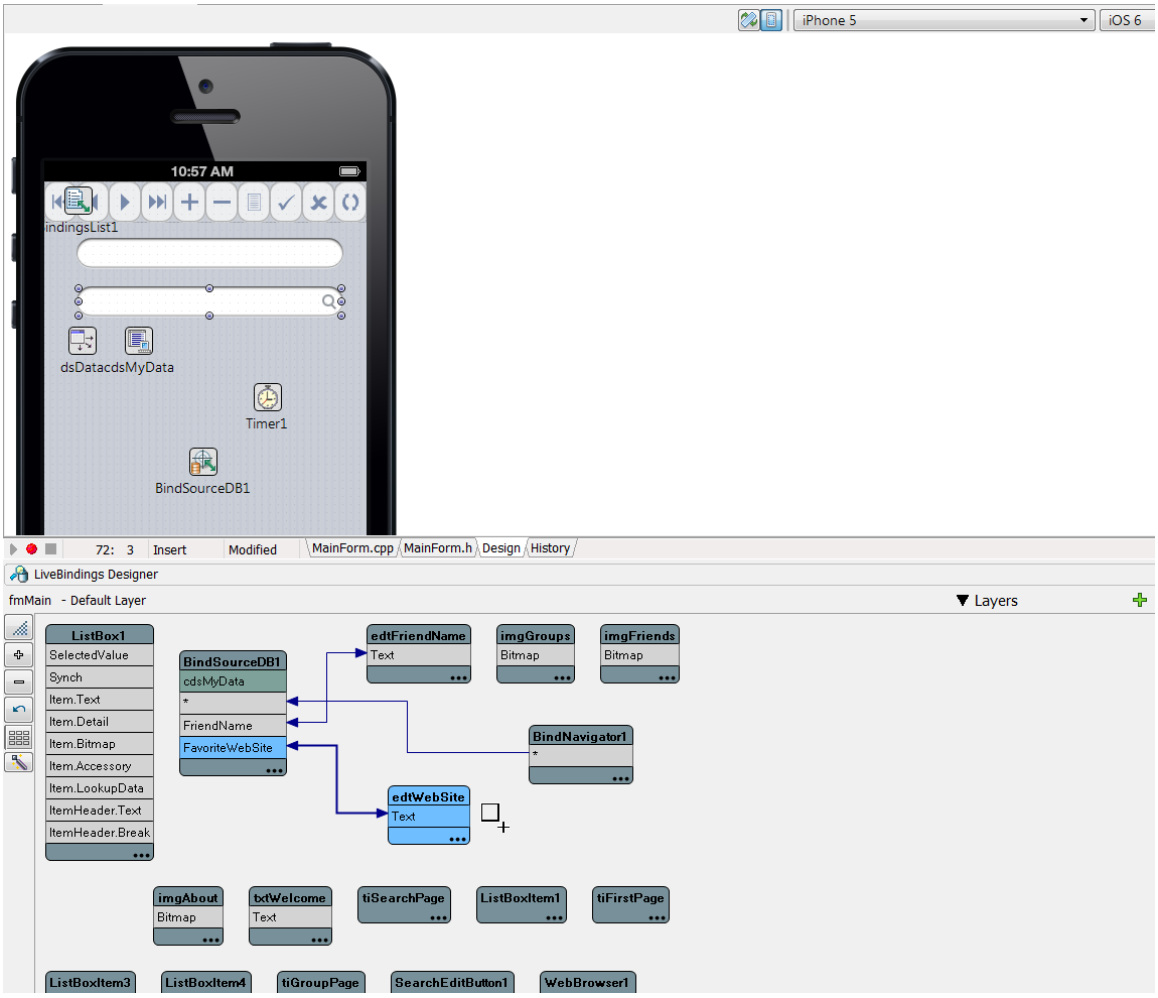
4-10 使用 LiveBinding 繫結資料和視覺化元件

C++Builder 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(繫結)在視覺化元件中。

在前面的 FillGroupDataSet() 方法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 | LiveBindings Designer 功能表以開啟視覺化即時資料繫結設計家，如下所示:

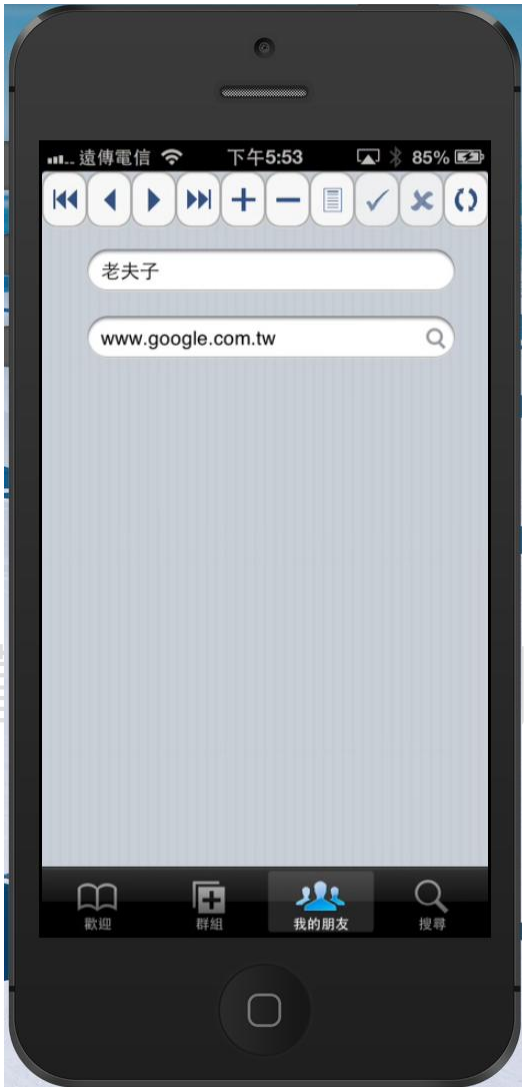


視覺化即時資料繫結設計家中您可以使用拖曳拉線的方式繫結資料和元件。例如現在我們想把 TClientDataSet 中的 FriendName 欄位顯示在 TTabControl 的第 3 個頁次中的第一個 TEdit 元件，把 FavoriteWebSite 欄位顯示在第二個 TEdit 元件中。因此請在視覺化即時資料繫結設計家中先點選 BindSourceDB1 中的 FriendName 欄位再持續按著滑鼠左鍵拖曳到 edtFriendName 的 Text 特性上再放開滑鼠左鍵，此時在這 2 者之間就會出現一個雙向箭頭的線條，這就代表現在我們已經繫結了 FriendName 欄位的數值和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



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瀏覽和移動了，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 C++Builder 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。

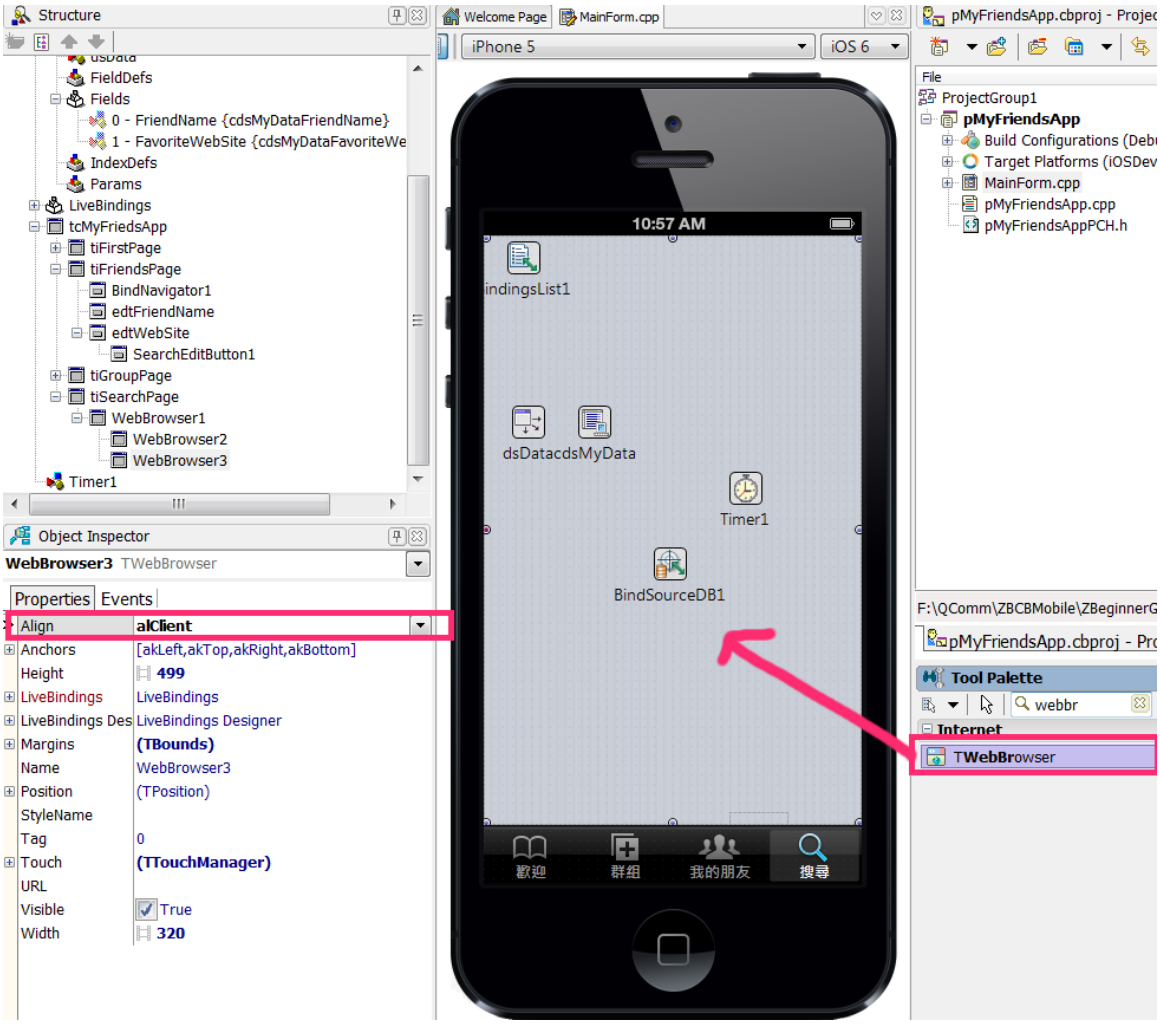


4-11 實作 SearchEditButton1Click 事件處理函式

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 個頁次並且使用瀏覽器帶領使用者到 FavoriteWebSite 欄位數值指定的網站。

實作這個功能非常的簡單，C++Builder 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。因此點選主表單中的『搜尋』頁次並且如下圖般把

TWebBrowser 元件拖曳到『搜尋』頁次中並且在物件檢視器中設定 TWebBrowser 元件的 Align 特性值為 alClient:



接著請在 SearchEditButton1 的事件處理函式中撰寫如下的程式碼:

```
void __fastcall TfmMain::SearchEditButton1Click(TObject *Sender)
{
  ///* DONE 1  -o 老夫子 : 使用者點選此鍵後開啟瀏覽器頁面 */
  WebBrowser1->URL = edtWebSite->Text;
  tcMyFriedsApp->ActiveTab = tiSearchPage;
  WebBrowser1->Navigate();
}
```

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 果然可以瀏覽到 FavoriteWebSite 欄位指定的網站，Cool。

4-12 開啟多執行程序編譯功能

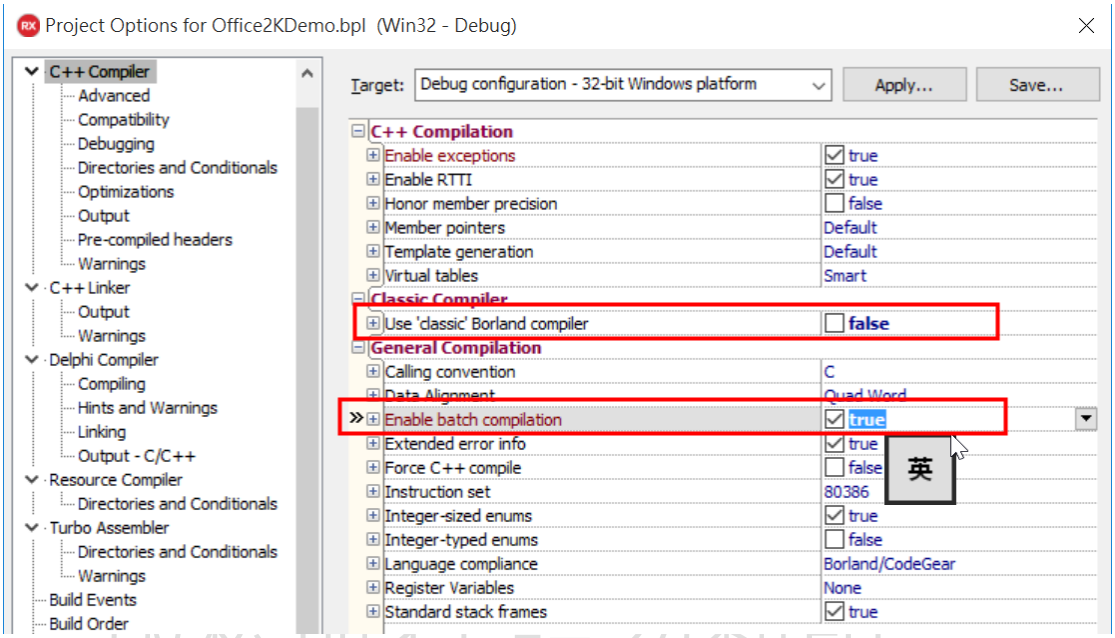
最重要的功能之一就是為 Win32 開發了基於 LLVM 的全新 Clang 編譯器，新的 Clang Win32 編譯器除了如同 Win64 編譯器支援 C++11 標準外，C++Builder 的 IDE 也為 Clang 編譯器開啟了多執行程序編譯功能，如此一來可讓 C/C++程式師在使用多核心的機器時能夠同時使用每一個閒置核心來編譯專案中每一份不同的 C++程式單元，這樣可以大幅加快專案的編譯速度。

要開啟 C/C++的多執行緒編譯功能，程式師必須：

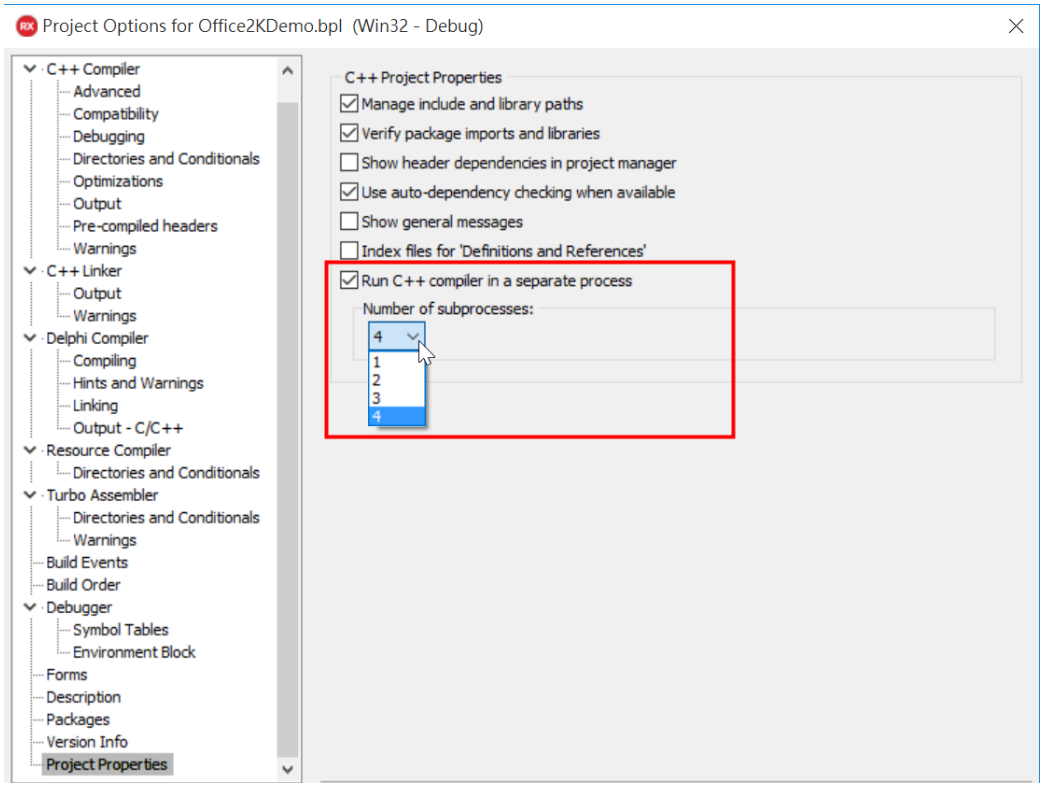
1. 使用 Clang 編譯器而不是使用舊的 Borland 編譯器
2. 開啟批次編譯功能
3. 開啟多執行程序編譯功能

第一步是使用 Clang 編譯器，請在專案管理員中使用滑鼠右擊專案，點選突顯示選單中的 Options 選項。在 C++ Compiler 選項中取消”Use ‘classsic’ Borland compiler” 選項。

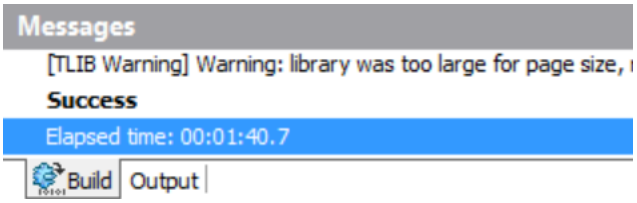
第二步是勾選其中的”Enable batch compilation” 選項：



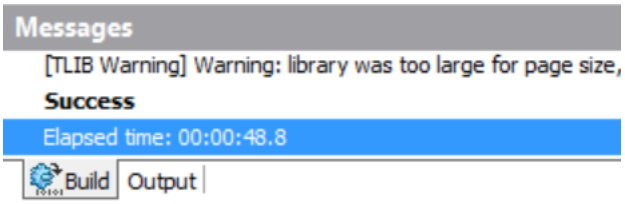
最後一步則是在對話盒左方最下面的 Project Properties 選項中勾選其中的”Run c++ compiler in a separate process” 選項，此時其下方的”Number of subprocesses”下拉盒就會啟動，您就可以根據您機器可以的 CPU 核心數來設定要啟動多少個程序一起編譯專案，例如筆者的虛擬機分配了 4 個核心，因此筆者選擇 4，如下所示：



一旦啟動了多執行程序編譯功能，當您編譯您的專案時就會查覺到專案編譯的速度快了許多，例下面是筆者一個包含 8 個 C++表單的專案，在正常情形下需要 1 分 40 秒左右在筆者的虛擬機中完全編譯：



但在開啟多執行程序編譯功能後，整個專案完全編譯的時間下降到只需要 48 秒左右：



4-13 Visual Assist 功能

C++Builder 在 12 的版本終於加入期待以久的 Visual Assist(VA)功能。VA 功能其實來自多年前 Idera 並購的 Whole Tomato， Whole Tomato 是 Visual C++非常受歡迎的外掛程式軟體，其功能是可大幅加強程式師撰寫程式碼的生產力，因此 C++Builder 的 VA 功能也就是強化程式師撰寫程式碼效率的功能。

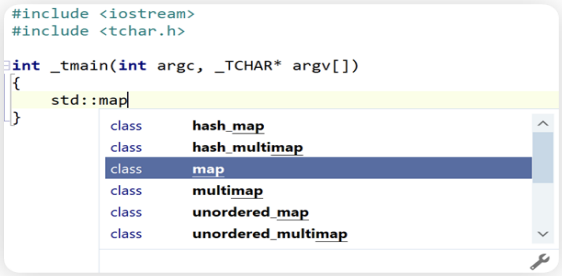
12 版的 VA 主要包含 3 個主要功能:

- Code Insight
- Refatoring
- Navigation

Code Insight

長久以來 C++Builder 的 Code Insight 功能一直無法跟上 Delphi Code Insight 的功能，其中最主要的原因就是因為編譯器的不同。C++Builder 和 Delphi 的 Code Insight 都是基於編譯器提供必要的顯示資訊，但由於 C++是 3-Pass 編譯器，而 Delphi 則是 1-Pass 編譯器，因此 C++Builder 的 Code Insight 在顯示速度，資訊更新都差很多。從以前的 Borland 傳統 C/C++編譯器一直到現在的 Clang 編譯器都有一樣的問題，即使 C++Builder 的 LSP 功能的確為 Code Insight 帶來了改善，但是 C++Builder 的 Code Insight 在使用經驗上仍然無法提供像 Delphi 的 Code Insight 一樣的速度和正確性。

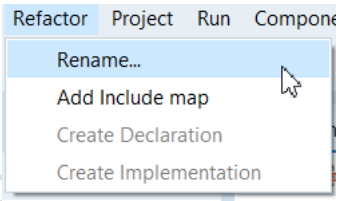
但是 12 版的 VA 的 Code Insight 功能帶來了改變，VA 的 Code Insight 並不是基於編譯器，VA 使用了自己的模糊邏輯運算法則來提供 Code Insight 的資訊，因此 12 版的 Code Insight 速度已經能夠和 Delphi 一樣快了，而且由於模糊邏輯運算法則，因此 VA 甚至能根據程式師撰寫的程式碼來預測 Code Insight 的資訊，並且能夠根據程式師撰寫的程式碼來建議要加入的表頭檔(.h/.hpp)。例如在下面的圖形中顯示出 VA 的 Code Insight 在尚未加入必須的表頭檔之前就能夠提供 Code Insight 顯示的資訊：



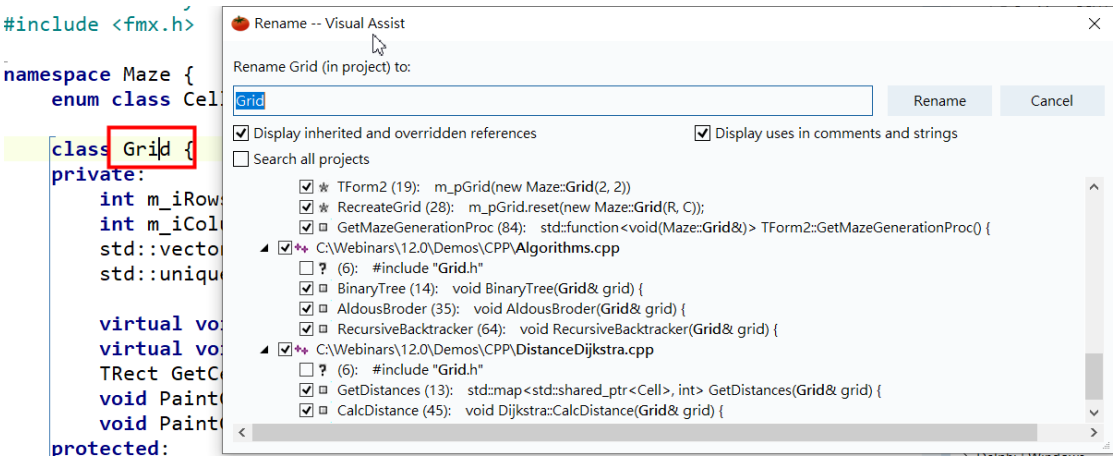
這個能力是以前的 Code Insight 無法提供的。

Refactoring

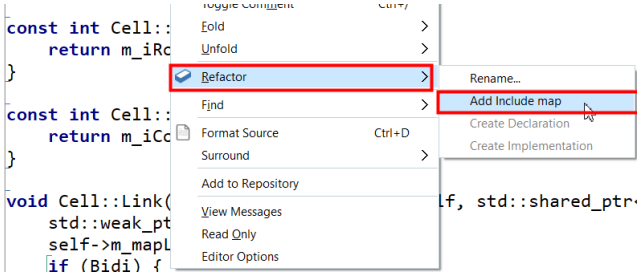
VA 的 Refactoring 提供了 3 個功能：可以為程式碼中的程式安全的重新命名，自動為程式加入表標頭檔以及自動產生函式的宣告或是實作內容。這 3 項功能都是以前 C++Builder 無法正確提供的



要安全的重新命名任何變數/函式，程式師可以把游標放在變數，函式中然後點選 Refactor|Rename，或是在編譯器中點選滑鼠右鍵再點選 Refactor|Rename，接著 VA 會顯示 Rename 對話盒列出所有定義和使用此變數/函式的程式碼給程式師檢查，接著程式師只需要輸入變數/函式的新名稱，VA 就會安全的把所有相關的程式師改動完成。



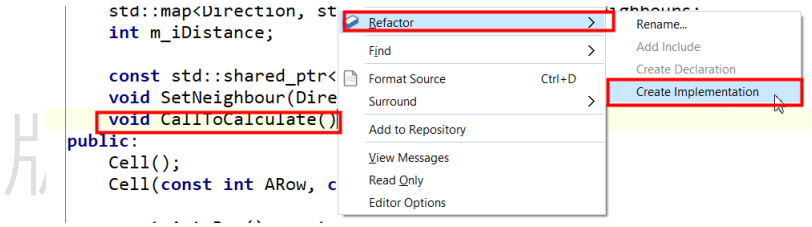
Refactoring 也可以自動幫程式師加入相關函式/API 的表頭檔，例如在前面 Code Insight 的範例中使用了 std::map 變數，那麼程式師可以點選滑鼠右鍵再點選 Refactor|Add Include map:



VA 就會自動在程式碼中加入相關的表頭檔

```
#include <map>
```

最後當程式師在宣告部份定義了函式之後就可以點選滑鼠右鍵再點選 Refactor|Create Implementation：



VA 就會在 Cpp 檔的實作中自動產生函式實作程式碼：

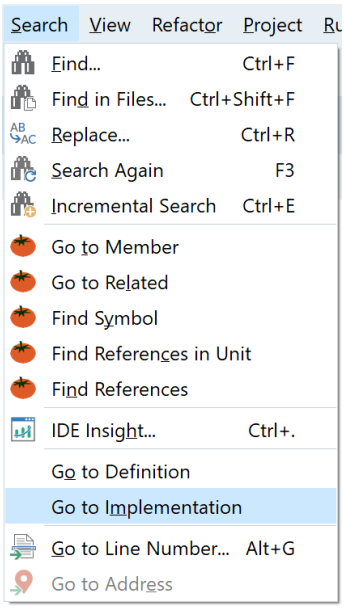
```
void Cell::CallToCalculate()
{


}
```

當然程式師也可以在 Cpp 檔的實作中先撰寫函式實作程式碼再要求 VA 自動在表頭檔中產生函式宣告。

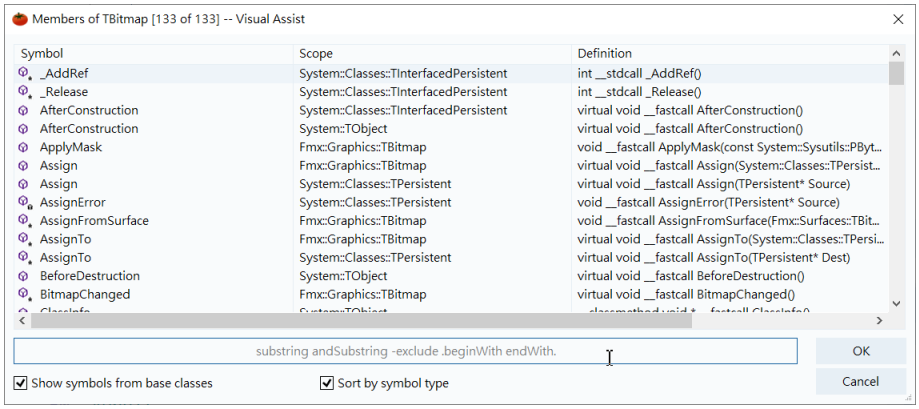
Navigation

VA 的 Navigation 幾乎可以帶領程式師尋找和觀看變數/函式/類別所有的相關資訊，這包含了宣告/實作/繼承資訊。

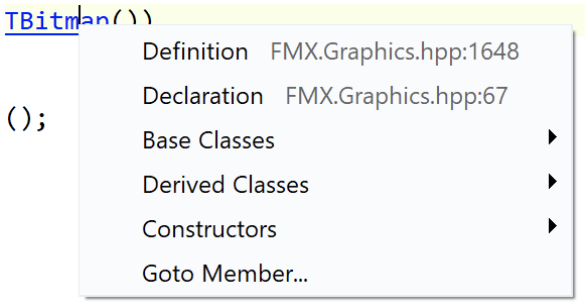
在 12 的 Search 選單中加入了 VA Navigation 所有的功能：



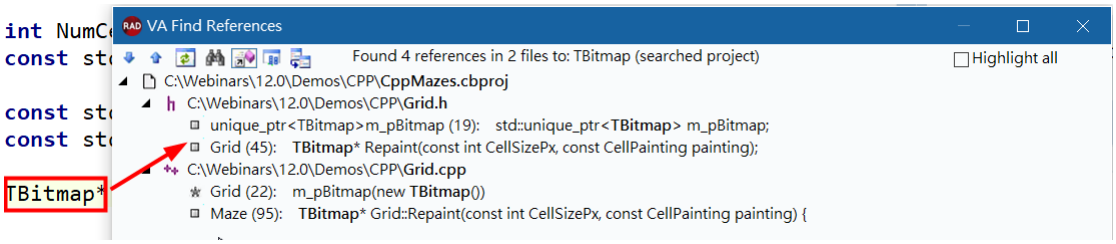
Go to Member 可以帶領程式師到類別/變數定義和實作程式碼，讓程式師可以快速在相關程式碼中來回。



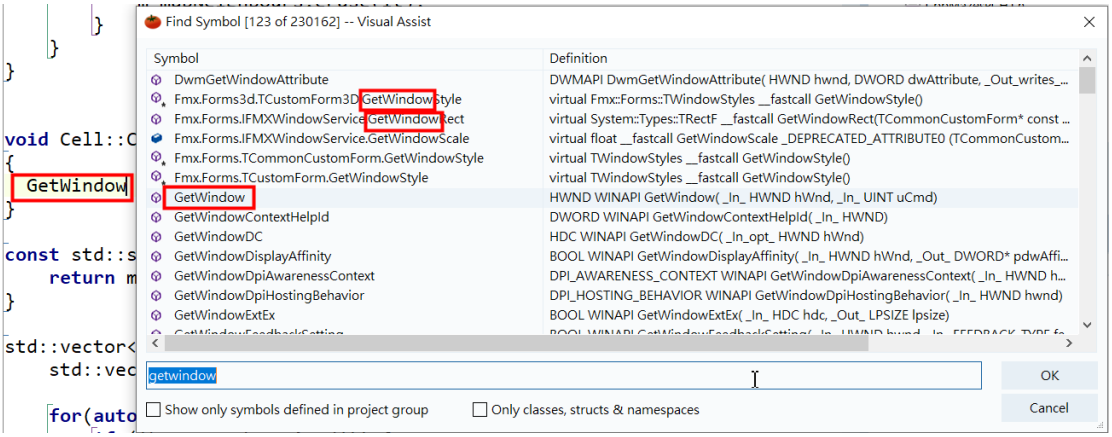
Go to Related 則可以讓程式師查閱類別/變數的相關資訊，例如點選類別並點選 Go to Related 後就可以查看類別的定義，成員，父代類別以及衍生類別：



Find References in Unit 和 Find References 可以在本程式單元或是在專案中搜尋所有使用特定類別/變數的相關程式碼，例如在下面顯示使用 Find References 功能在專案中搜尋所有使用或是參考 TBitmap 類別的相關程式碼：



Find Symbol 則可以在專案或是**系統**中搜尋特定字串/標記的資訊，例如下面是在程式碼中呼叫 GetWindow 這個 API，但是程式師不確定 GetWindow 的定義以及定義的檔案，此時程式師可以使用 Find Symbol 功能來搜尋，VA 就會顯示所有和 GetWindow 有關的程式碼：



程式師只需要點選想要的資訊，VA 就會自動開啟 GetWindow 定義的 winuser.h 檔案並且帶領程式師到達 GetWindow 的定義程式碼：

```
WINUSERAPI
HWND
WINAPI
GetWindow(
    _In_ HWND hWnd,
    _In_ UINT uCmd);
```

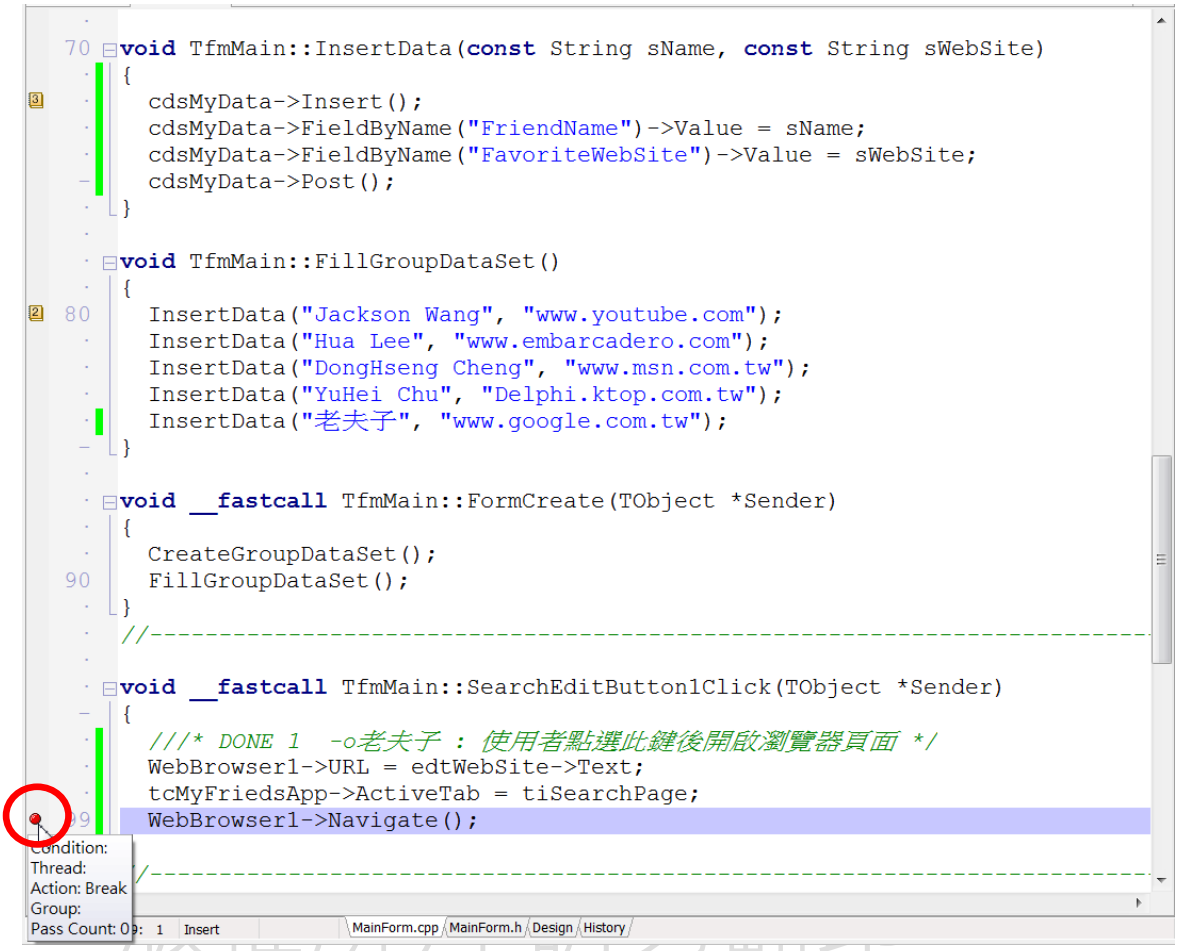
12 版的 VA 功能的確為 C++Builder 程式師帶來了更高效率的程式碼生產力，特別是當您的項目愈來愈龐大時就能更體會到 VA 的好處，未來 C++Builder 會持續加入更多的 VA 功能。

5 除錯您的 iOS App

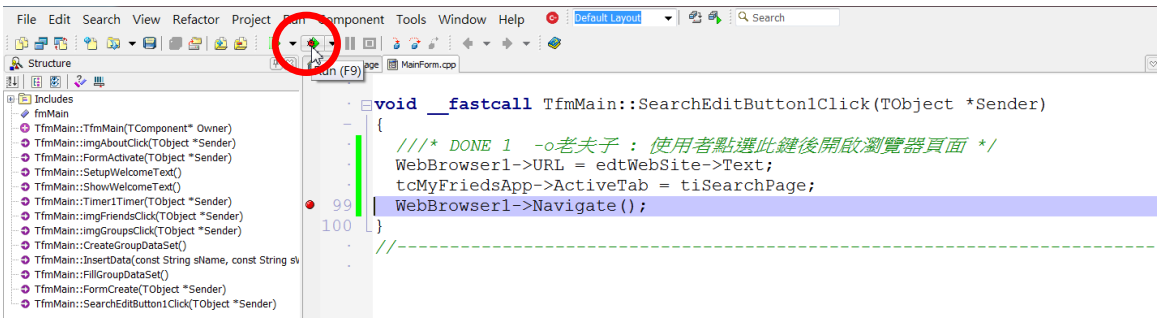
當您使用 C++Builder for iOS 開發 iOS App 時一定會需要除錯您的應用程式，C++Builder for iOS 提供了非常方便又強大的除錯功能能幫助您，例如本書的範例應用程式在實作了上一小節的程式碼後可能發生了一些錯誤，因此讓我們學習如何除錯應用程式以便修正範例應用程式中可能的錯誤。

C++Builder 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。

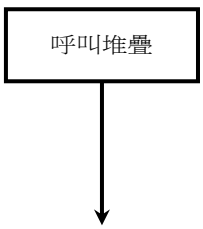
現在讓我們在前面剛實作的 SearchEditButton1Click 程序中設定一個中斷點，請切換到程式碼頁次，使用滑鼠在 SearchEditButton1Click 程序的最後一行程式碼處的最左邊單擊滑鼠左鍵，此時在 WebBrowser1.Navigate 最左邊就會出現一個紅色的圓點『  』，這就代表在此設定了一個『中斷點』，稍後當範例 iOS App 在 iOS 設備中執行到此地時就會中斷執行並且把執行權從應用程式切換回 C++Builder for iOS 的 IDE，如下所示:

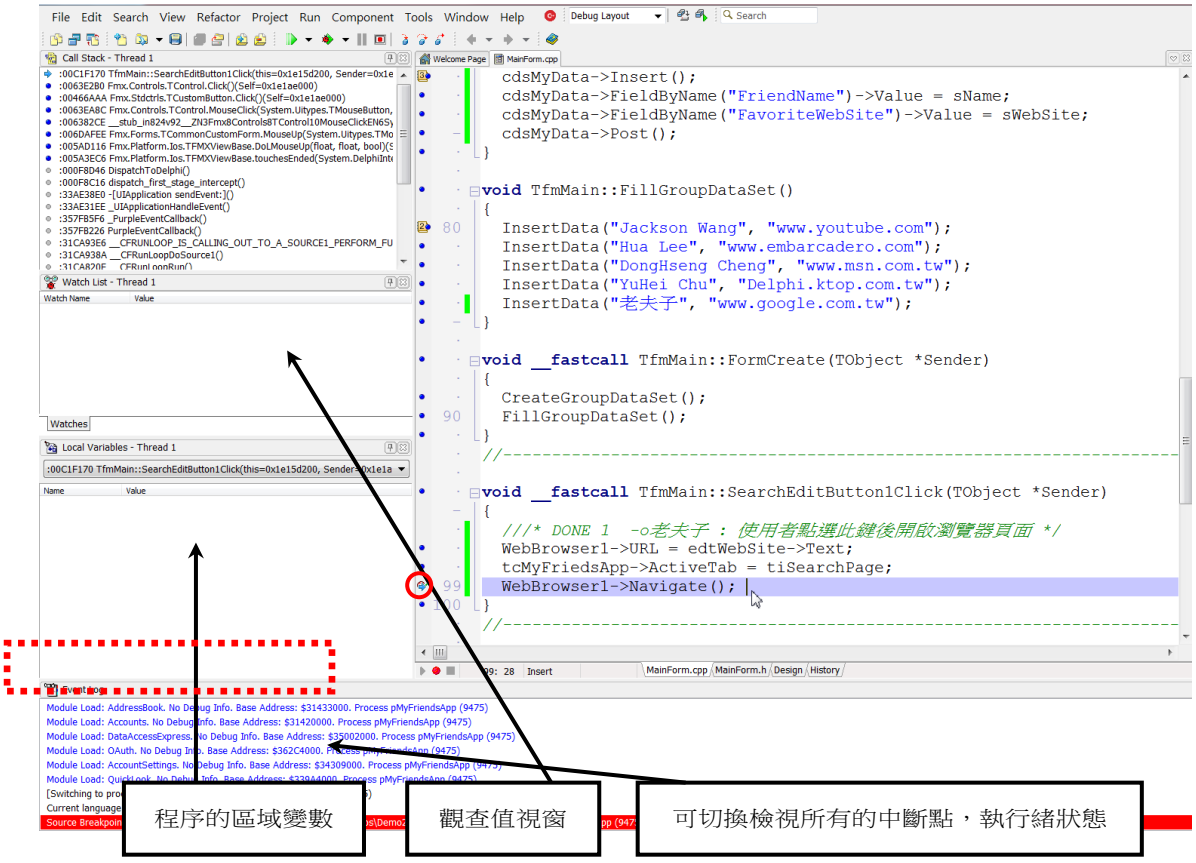


設定好此『中斷點』之後，您就可以點選 IDE 上方的『Run』按鈕，或是按下『F9』，IDE 就會啟動除錯器開始執行您的應用程式，如下所示:



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 C++Builder 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。



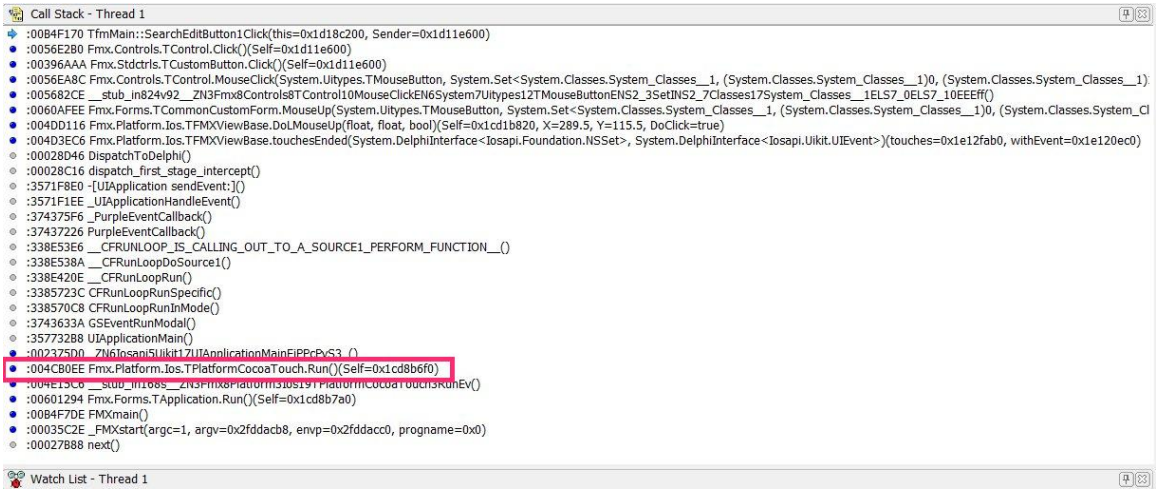


而且請您注意 IDE 右上方，此時 IDE 也被設定成在除錯的桌面組態設定:

除錯的組態設定會自動顯示呼叫堆疊視窗，程序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:

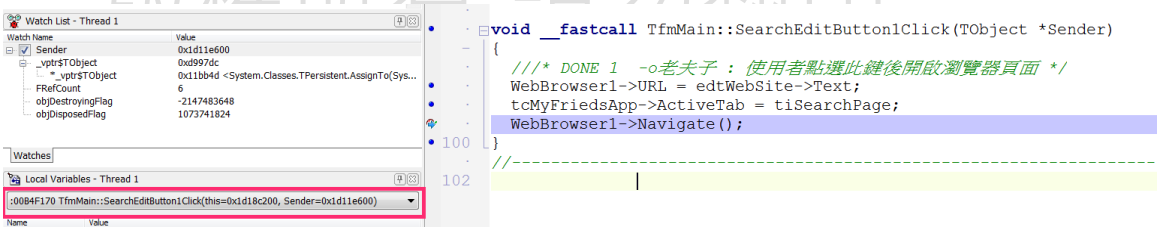
視窗	說明
堆疊視窗	應用程式的呼叫堆疊次序，您可以在這個視窗中看到中斷點被呼叫的執行次序
程序的區域變數視窗	此視窗自動顯示目前程序中所有的區域變數的數值
觀查值視窗	您可以在此視窗中加入檢視任何的全域變數，資料結構或是物件的數值

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:



『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 程序，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 程序。

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 程序，如下所示:



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

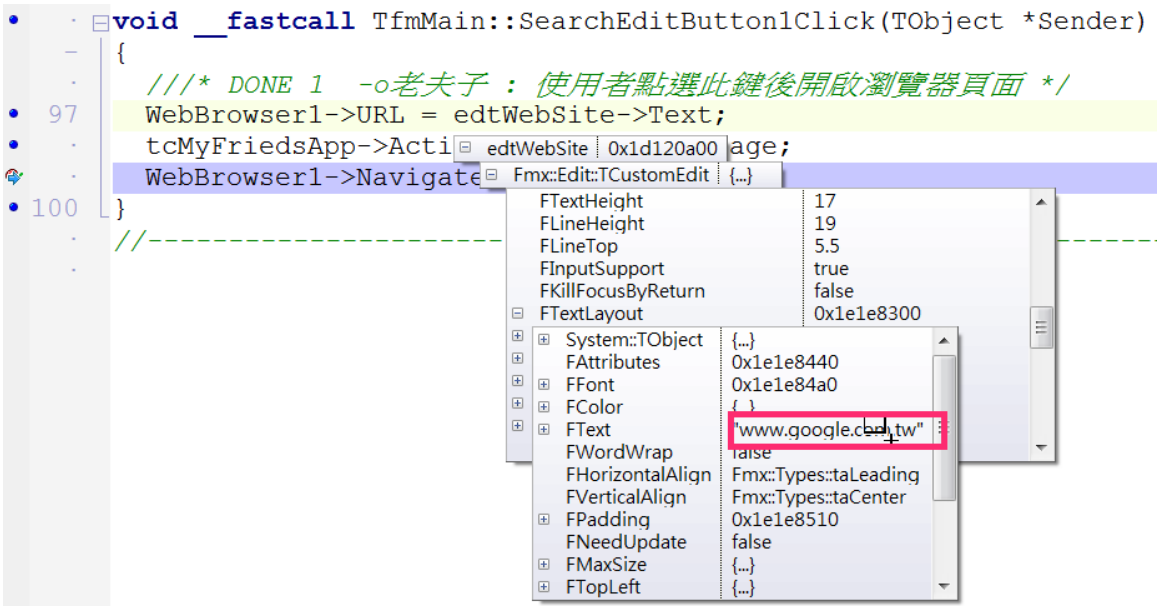
當應用程式執行權暫停在中斷點時，您也可以使用滑鼠來檢視程式碼中的變數，資料結構或是物件的數值。例如現在 SearchEditButton1Click 程序中使用了:

```
WebBrowser1->URL = edtWebSite->Text;
```

上面的 edtWebSite.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 edtWebSite->Text 之後，您就可以看到 IDE 在游標下方顯示了目前 edtWebSite->Text 中的數值:

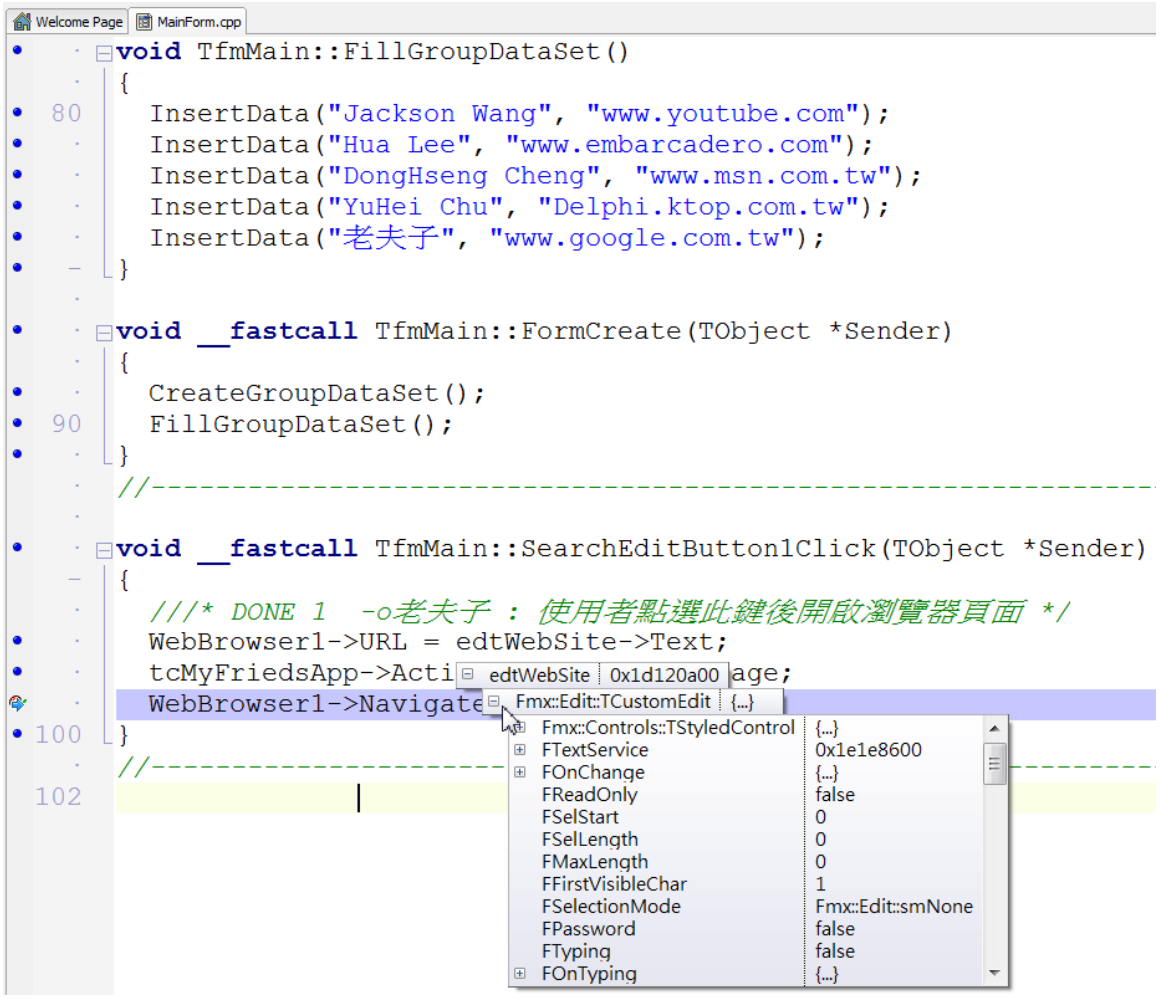


第 2 種方法是把這個程式碼拖曳到『觀查值視窗』中，例如下圖就是使用滑鼠選擇了 edtWebSite->Text 之後，把它拖曳到『觀查值視窗』中。

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 edtWebSite 之上，除錯器就立刻顯示 edtWebSite 之中所有的資料:

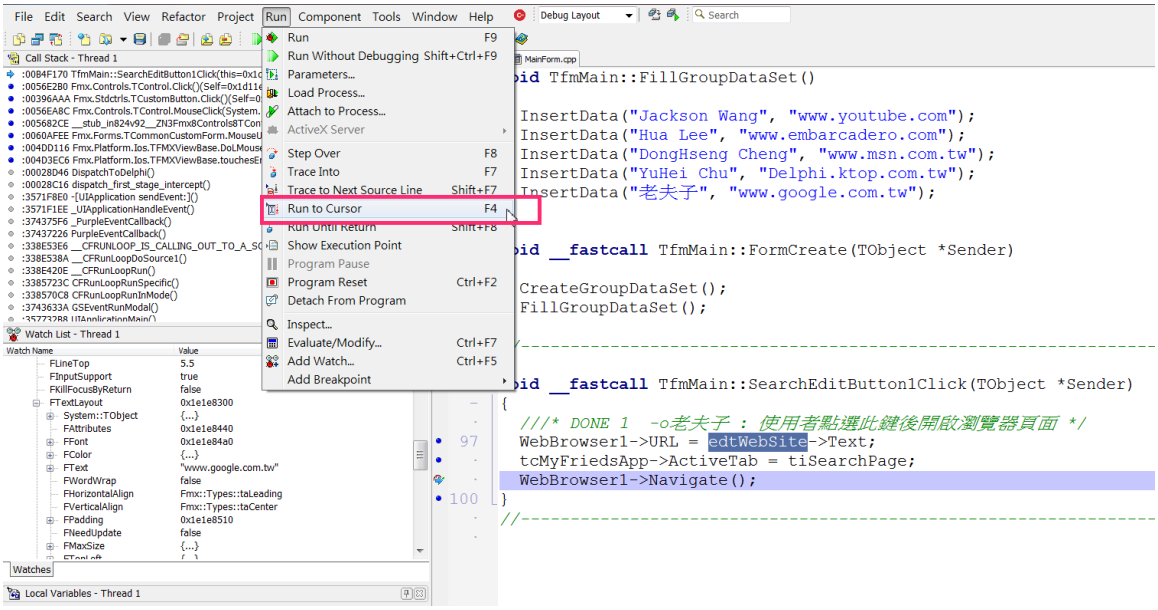
請注意上圖中 edtWebSite 左方有一個『+』符號，這代表您可以使用滑鼠展開其中的內容。例如在下圖使用滑鼠單擊 edtWebSite 左方的『+』符號，就可以看到除錯器展開 edtWebSite 中的內容，除錯器詳細的列出了 edtWebSite 中每一個元素的數值，這個功能對於觀察複雜的資料結構或是物件的內容是非常有用的。



現在請您再次單擊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:

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 開發和分發 iOS App 到 iOS 設備中

在 C++Builder For 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 C++Builder For iOS 中建立 iOS 設備的遠端組態
- 在 C++Builder For iOS 開發和除錯您的 iOS App
- 取得 Apple 開發/分發認證
- 使用 C++Builder For iOS 的部署管理員分發您的 iOS App

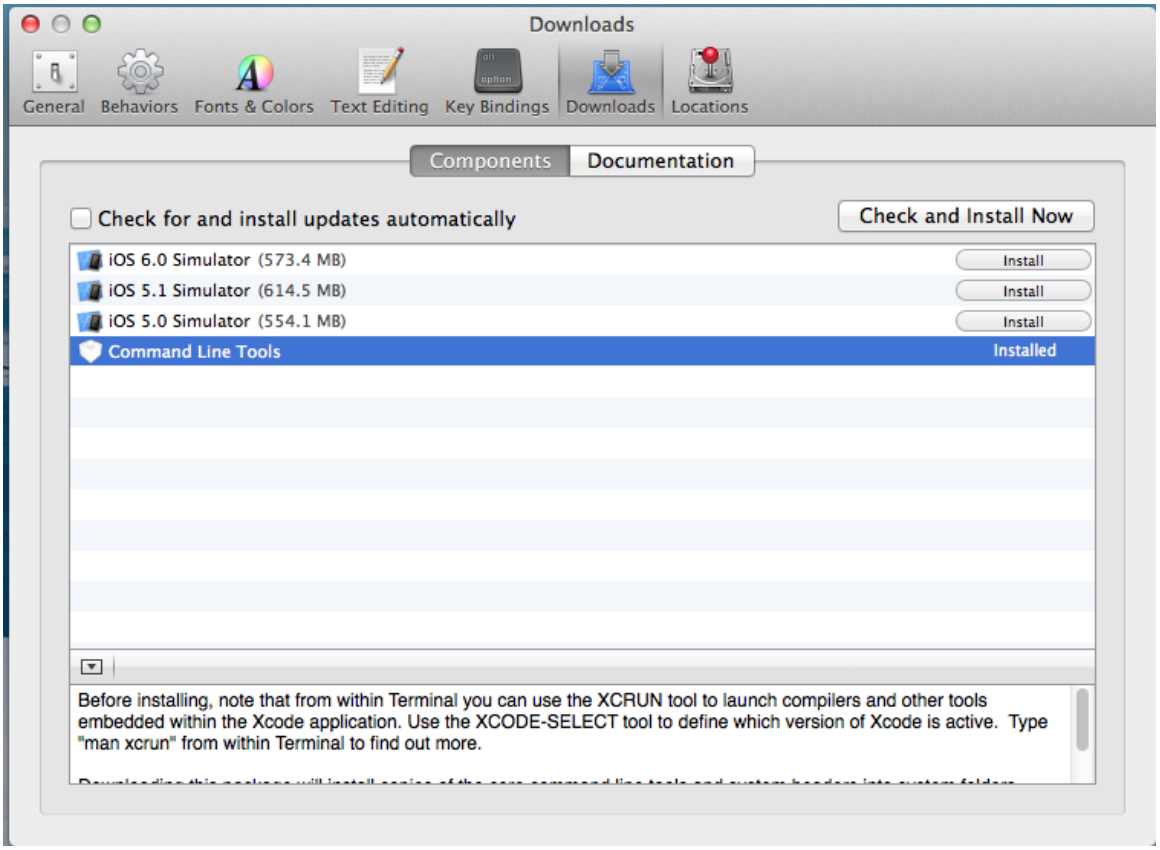
在下面的小節將使用一個實際的範例來說明上述的流程。

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 C++Builder For 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:



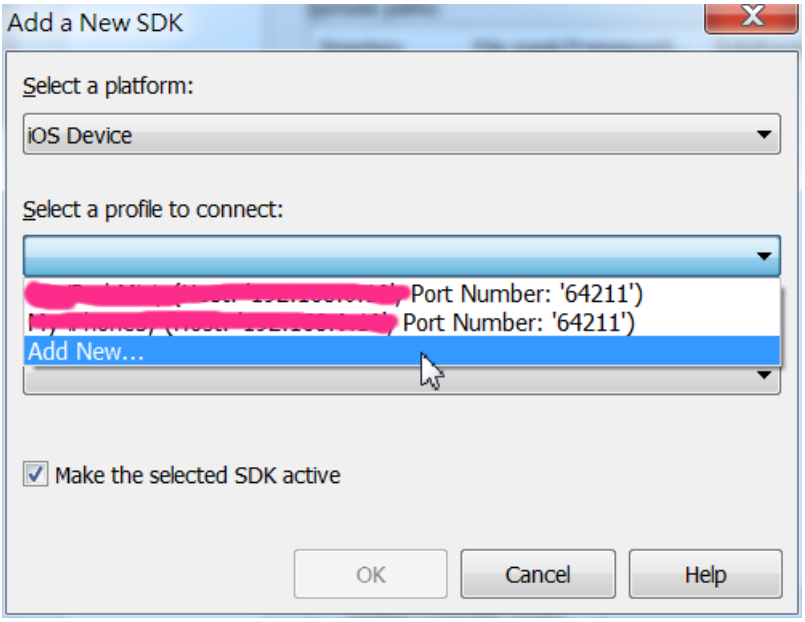
現在我們就可以進行下一步了。

建立 iOS 設備的遠端組態

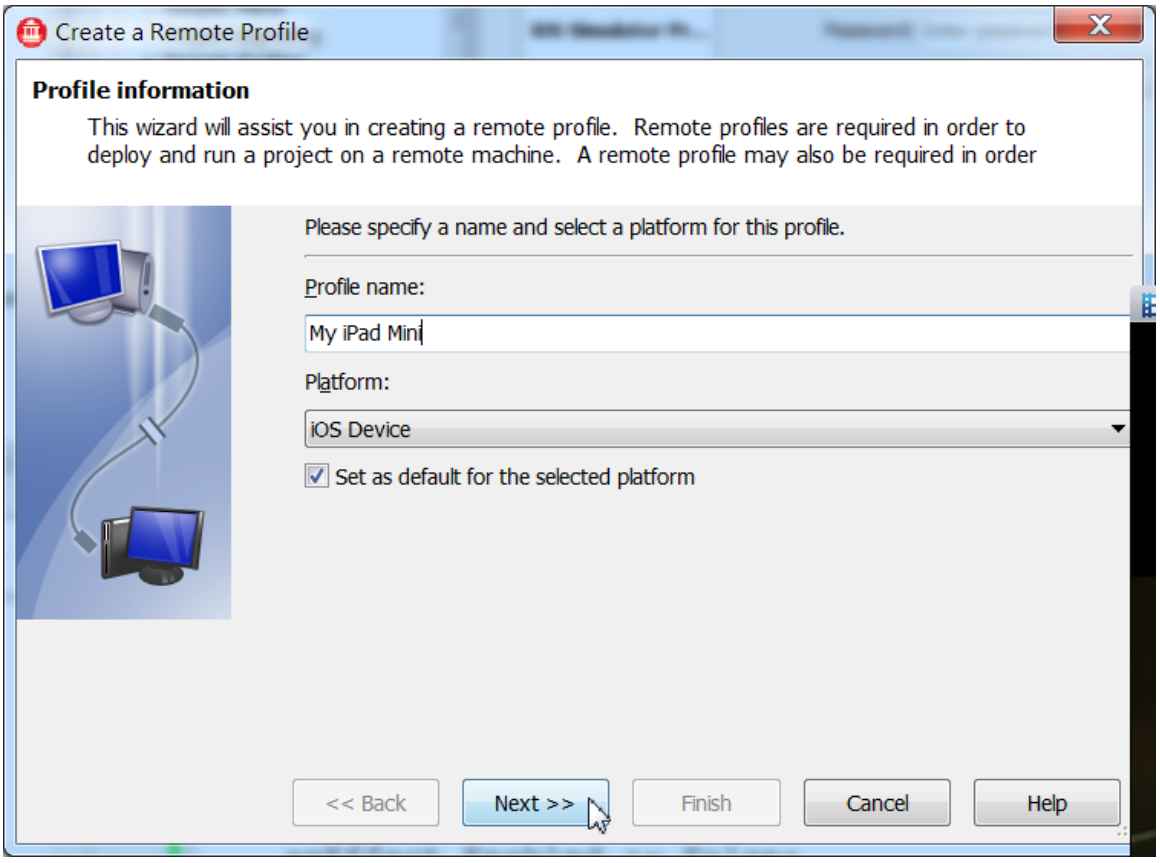
在前面我們說明了如何在 C++Builder For iOS 整合發展環境中建立 iOS 模擬器的組態以便我們開發和測試 iOS App。如果我們需要在 C++Builder For iOS 整合發展環境中實際分發 iOS App，我們也需要建立 iOS 設備的遠端組態，才能藉由 C++Builder for iOS 和 XCode 的命令列工具把 App 自動分發到 iOS 設備中。

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

請在 C++Builder 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:

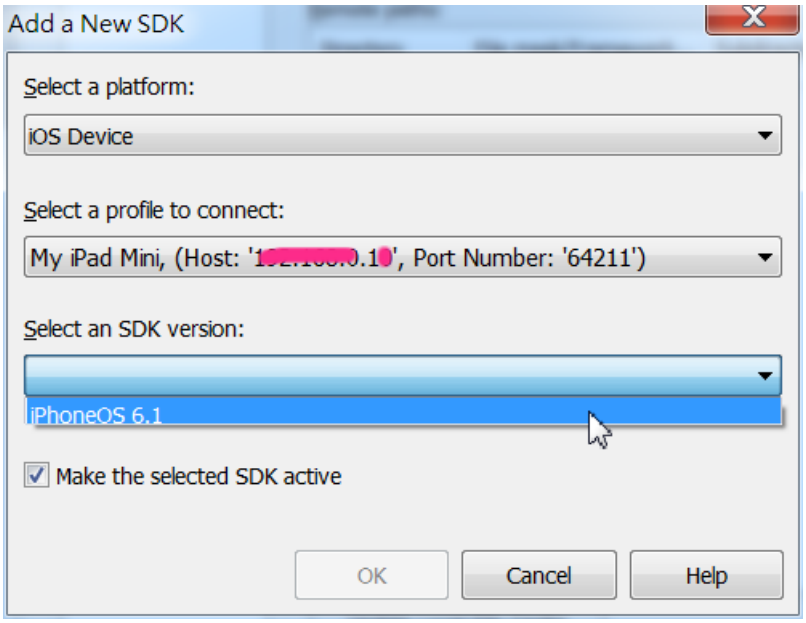


然後在 Create a Remote Profile 對話盒中為您的 iOS 設備取一個名稱並且在 Platform 欄位中選擇建立『iOS Device』平台組態。例如筆者使用的 iOS Device 是 iPad Mini，因此在下面的對話盒中取名為 My iPad Mini 設備名稱:



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

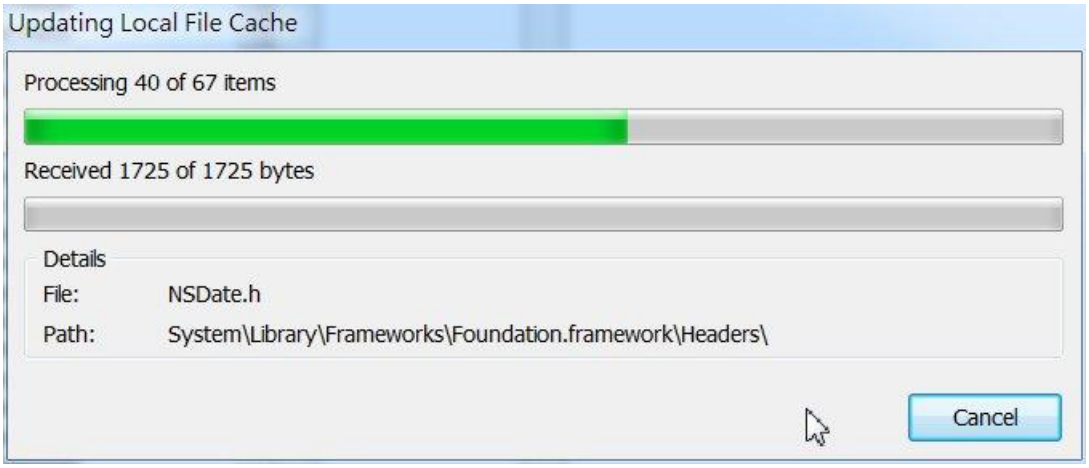
New SDK』對話盒就會立刻和 Mac 通訊並未找出此 iOS 設備使用的 SDK 版本，例如下圖就顯示『Add a New SDK』對話盒找到筆者使用的 iPad Mini 是使用 iOS 6.1 的版本，請選擇顯示的版本:



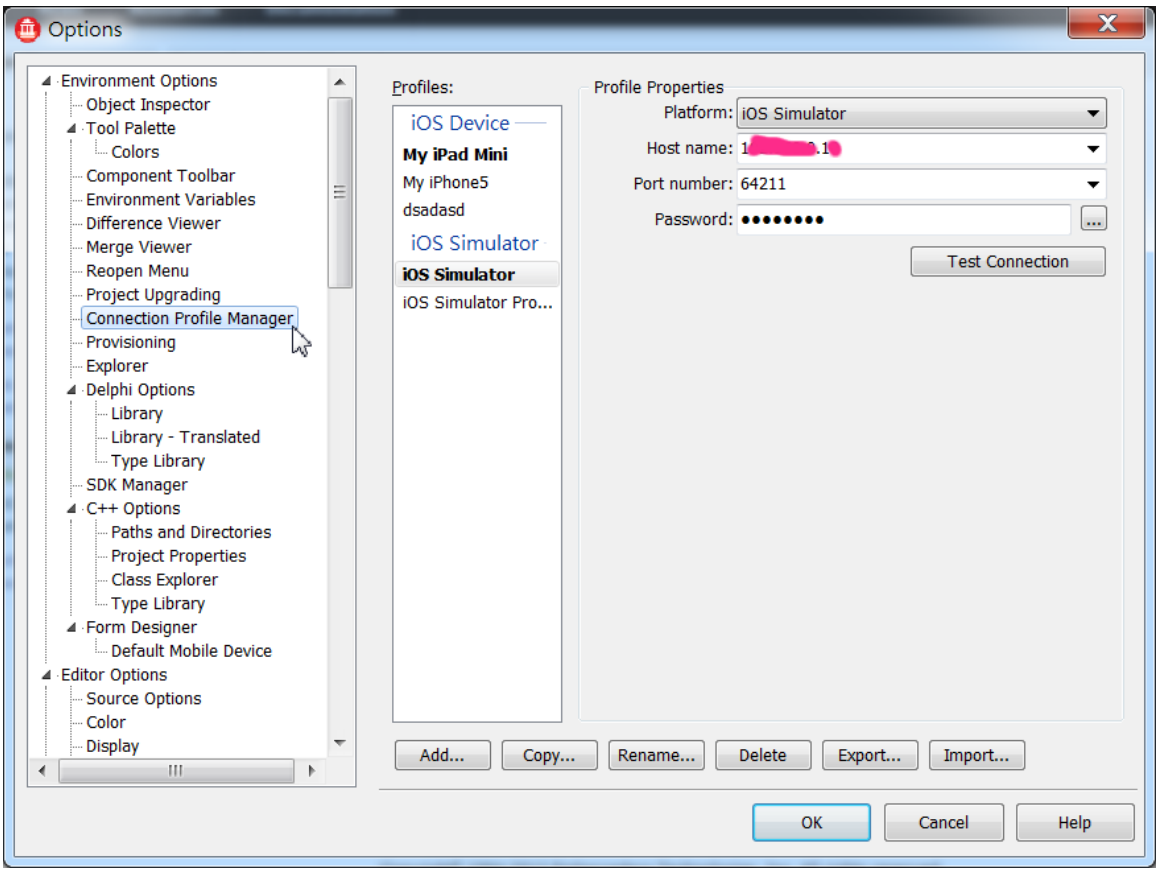
再點選 OK 按鈕之後

(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)

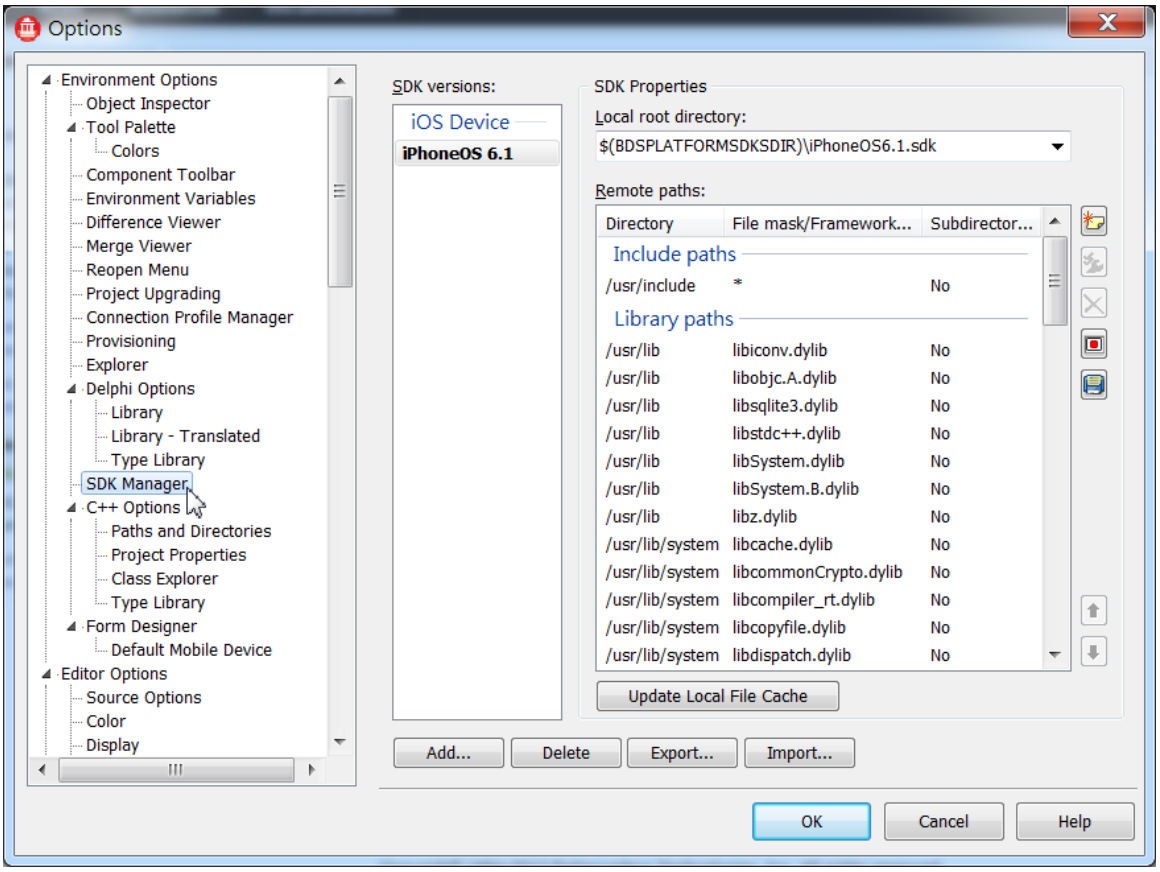
此時 C++Builder for iOS 整合發展環境就會自動根據您的設定拷貝和更新相關的檔案以幫助您分發 iOS App，如下所示:



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:



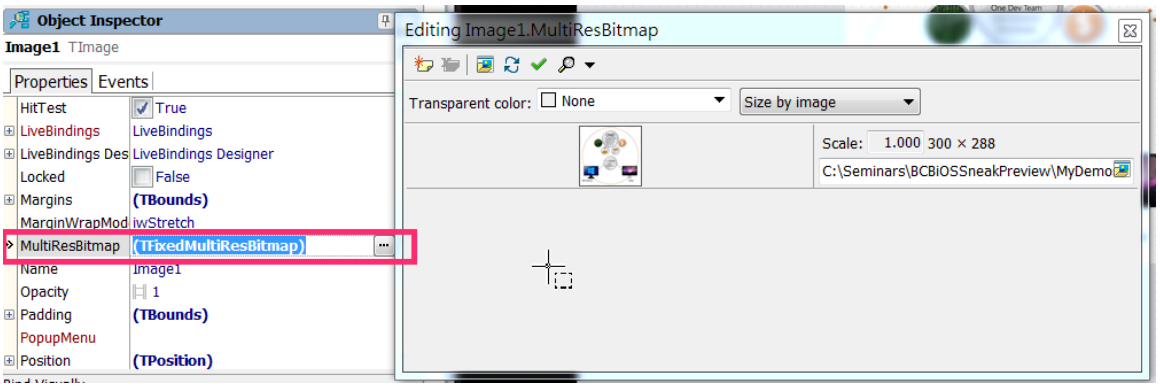
而 SDK Manager 中也會顯示您的 iOS 設備使用的 SDK 版本資訊和相關的檔案:

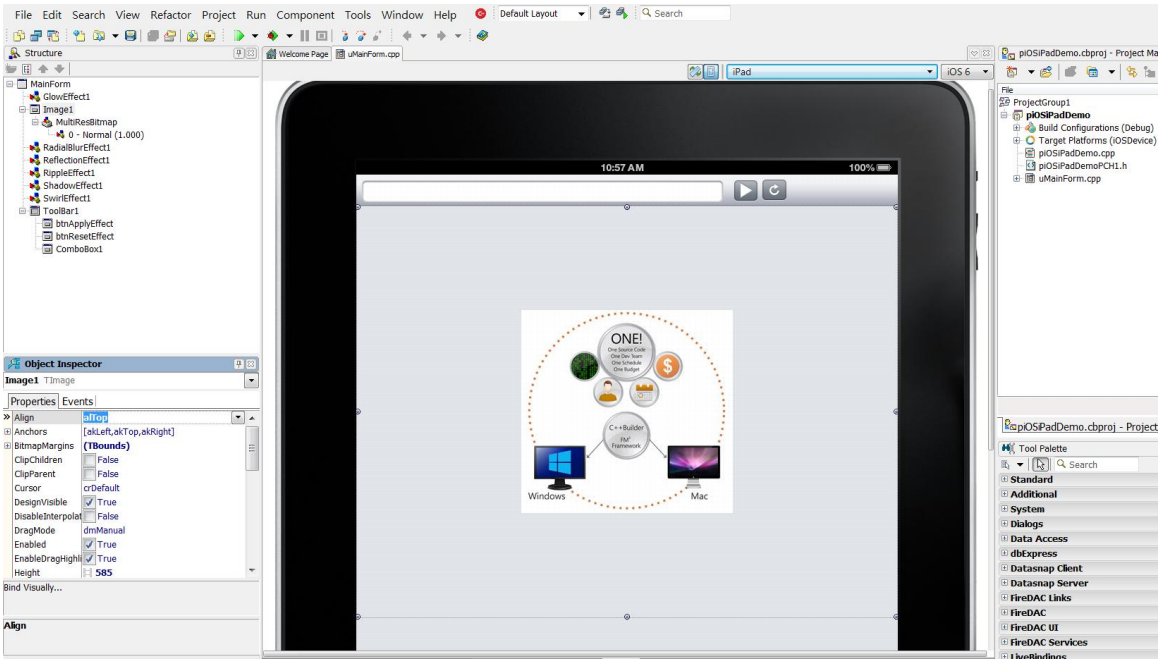


開發 iPad Mini 範例 App

現在請在 C++Builder 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:

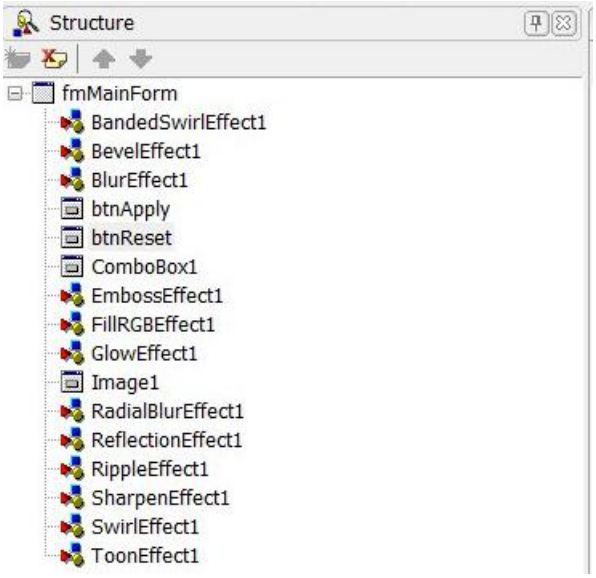
接著在主表單中放入 TImage，TEdit，TComboBox 和 2 個按鈕元件，再從工具盤中隨便選擇數個 Effect 元件，例如 TRippleEffect TShadowEffect 等。最後再使用 TImage 元件的 MultiResBitmap 特性的特性值編輯器載入一個影像，如下所示:





這個 iPad 範例 App 的功能是讓使用者可以從 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。

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:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
   //取得 FireMonkey 的 Effect
   GetAllAvailabelEffects(ComboBox1->Items);
}
```

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:

```
void TMainForm::GetAllAvailabelEffects(TStrings *pSL)
{
  for (int iCount = 0; iCount < this->ComponentCount; iCount++)
  {
    TComponent *pComponent = this->Components[iCount];
    if (pComponent->InheritsFrom(__classid(TEffect)))
    {
      TEffect *pEffect = (TEffect *) (this->Components[iCount]);
      pSL->AddObject(pEffect->ClassName(), (TObject *) pEffect);
    }
  }
}
```

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 005 行就從 TComboBox 中取得前面儲存在 TComboBox 中

的效果元件參考，008 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 009 行設定效果元件的 Parent 特性值為 TImage 元件，010 行再設定效果元件的 Enabled 特性值為 true 即可。最後 011 行把目前作用中的效果元件儲存在 pOldEffect 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    void __fastcall TMainForm::btnApplyEffectClick(TObject *Sender)
002    {
003      if (ComboBox1->ItemIndex != -1)
004      {
005        TEffect *pEffect = (TEffect*)
(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
006        if (pEffect != NULL)
007        {
008          ResetEffect(pOldEffect);
009          pEffect->Parent = Image1;
010          pEffect->Enabled = true;
011          pOldEffect = pEffect;
012        }
013      }
014    }
```

第二個按鈕的 OnClick 事件處理函式是在移除效果元件的作用:

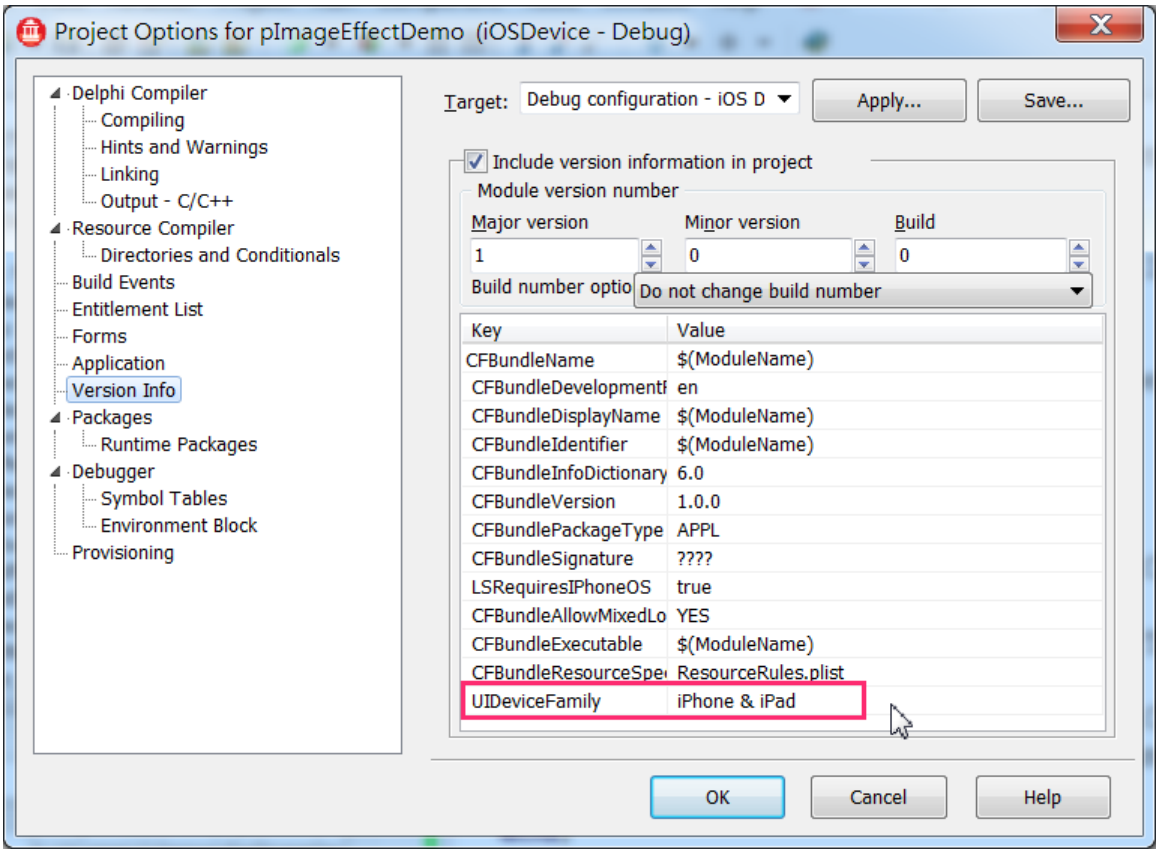
```
void __fastcall TMainForm::btnResetEffectClick(TObject *Sender)
{
  ResetEffect(pOldEffect);
}
```

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。

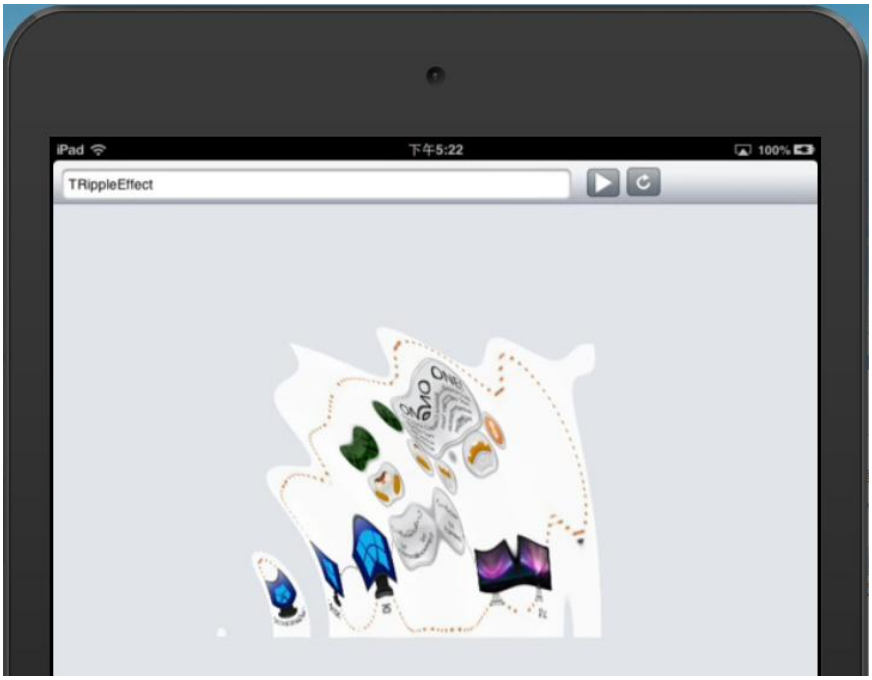
```
void TMainForm::ResetEffect(TEffect *pEffect)
{
  if (pEffect != NULL)
  {
    pEffect->Enabled = false;
    pEffect->Parent = NULL;
    pOldEffect = NULL;
```

```
  }
}
```

不我們可以準備執行這個範例 iPad App 了，但在這之前請在專案管理員中點選滑鼠右鍵，選擇 Options|Version Info 選項，確定其中的『UIDevice Family』的選項值為『iPhone & iPad』，如下所示:



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 C++Builder for iOS 開發的 App 到 iOS 設備之前，您需要具備 iOS 開發人員計劃資格。您可以使用 2 種方式取得 iOS 開發人員計劃資格:

- 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劃
- 或是加入您公司的 iOS 企業開發計劃，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 企業開發計劃

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劃並且據以取得分發認證以部署筆者使用 C++Builder 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 企業計劃管理者發 EMail 要求加入 iOS 企業計劃
- iOS 企業計劃管理者會把您加入 iOS 企業計劃，Apple 會自動寄一封 EMail 告訴您已經加入了 Apple 的 iOS 企業開發計劃:

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

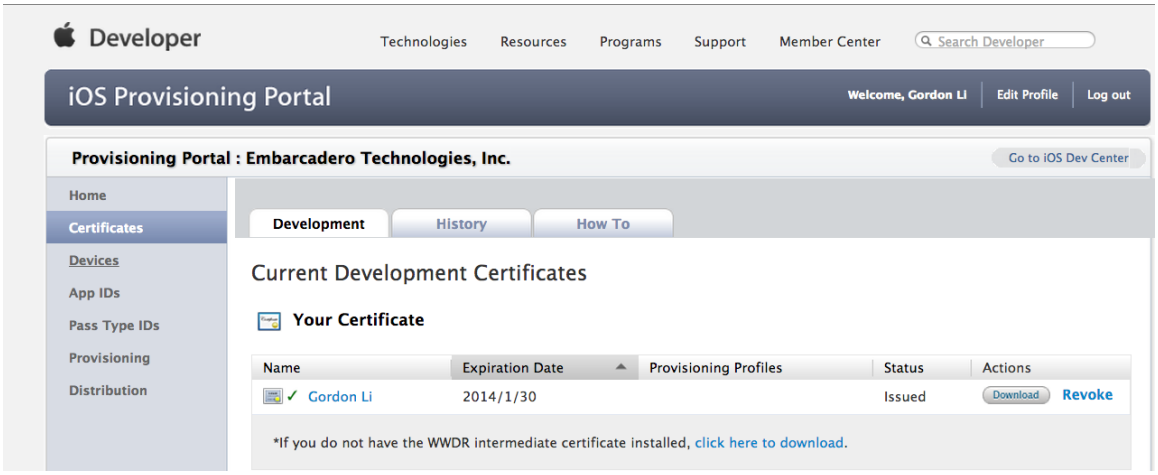
- 請使用瀏覽器前往:

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

並且使用您的 Apple ID 登錄:

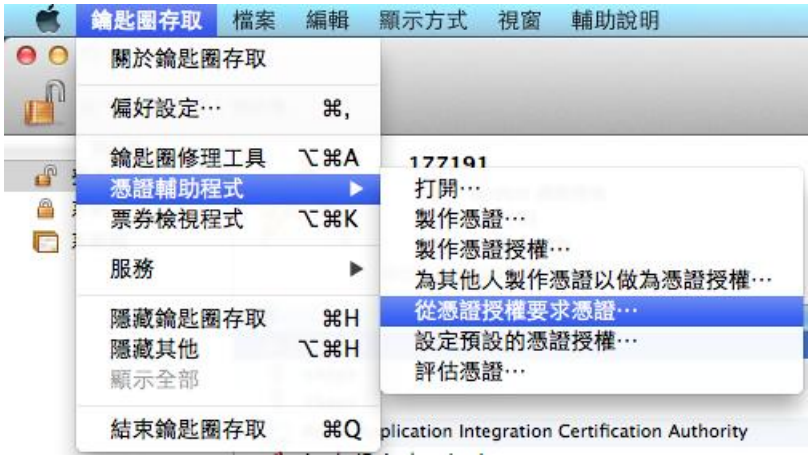
在成功登錄之後請點選瀏覽器右上方的『iOS Provisioning Portal』準備向 Embarcadero 的 iOS 企業計劃管理者申請筆者 iPad Mini 的認證。

首先點選瀏覽器左方的 Certificates 項目，然後點選下載 AppleWWWDRCA.cer 檔案:

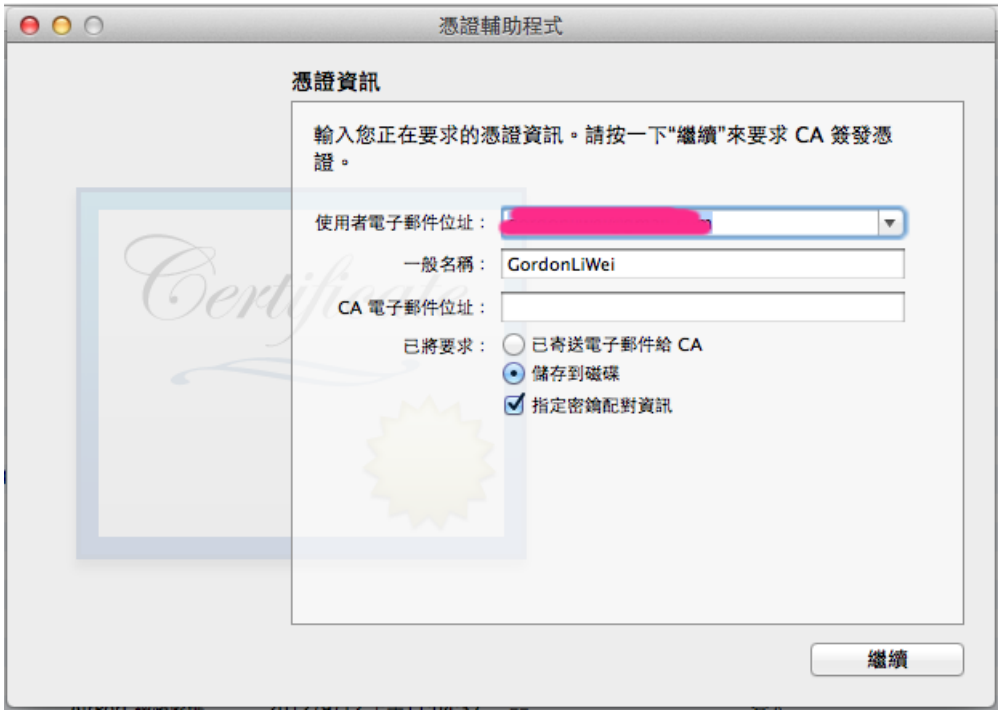


下載之後雙擊 AppleWWWDRCA.cer 檔案就會自動啟動鑰匙圈存取程式，在其中讀者可以看到 Apple WorldWide 開發資訊已經註冊:

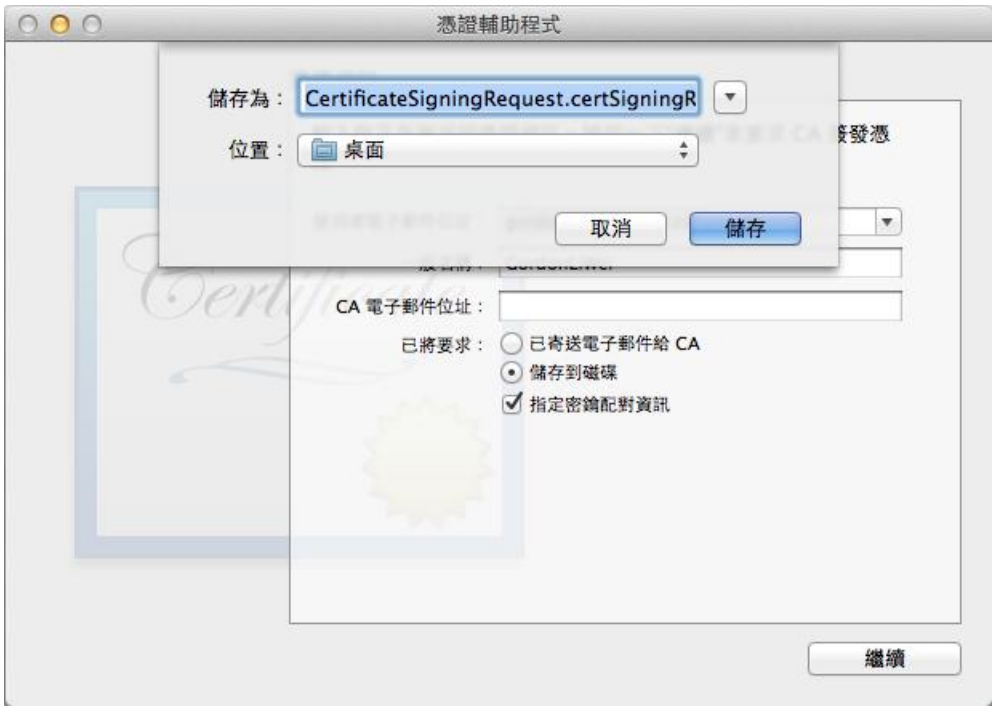
接著請點選鑰匙圈存取程式功能表，選擇從憑證授權機構(對筆者來說就是 Embarcadero)要求憑證，如下所示:



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:

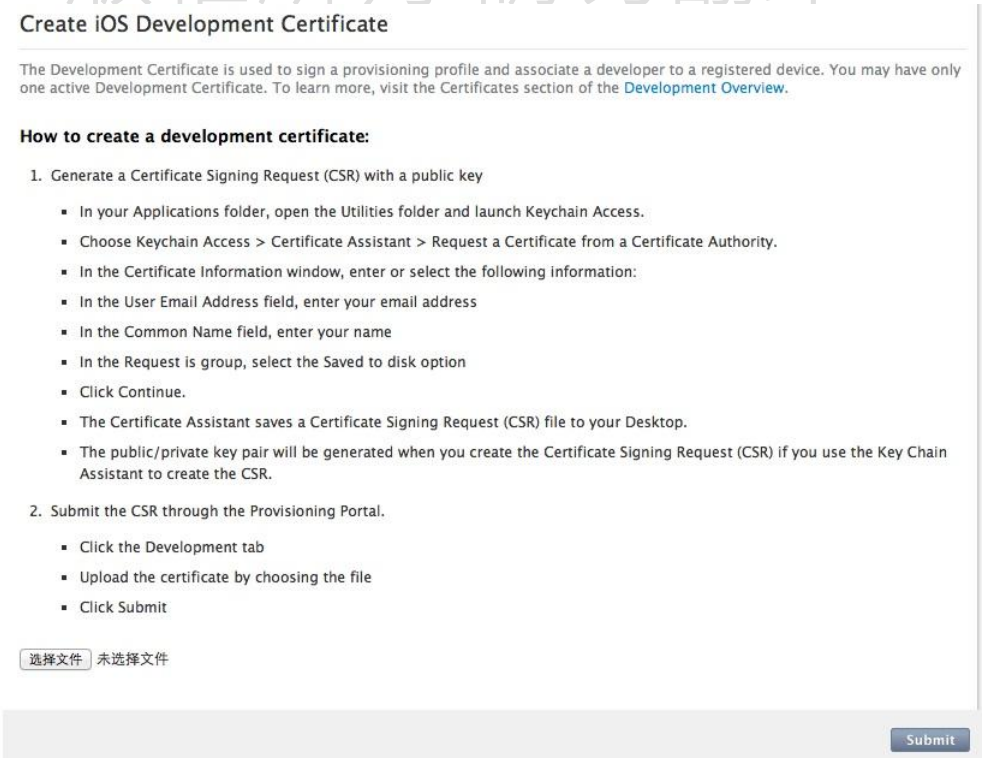


請選擇 RSA 演算法，點選繼續

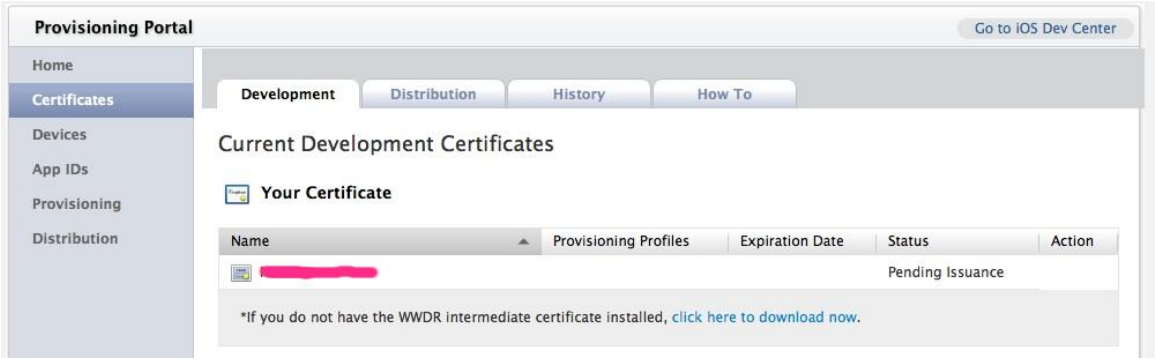
點選繼續之後就會在 Mac 的桌面看到產生的 CertificateSigningRequest.certSigningRequest:

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

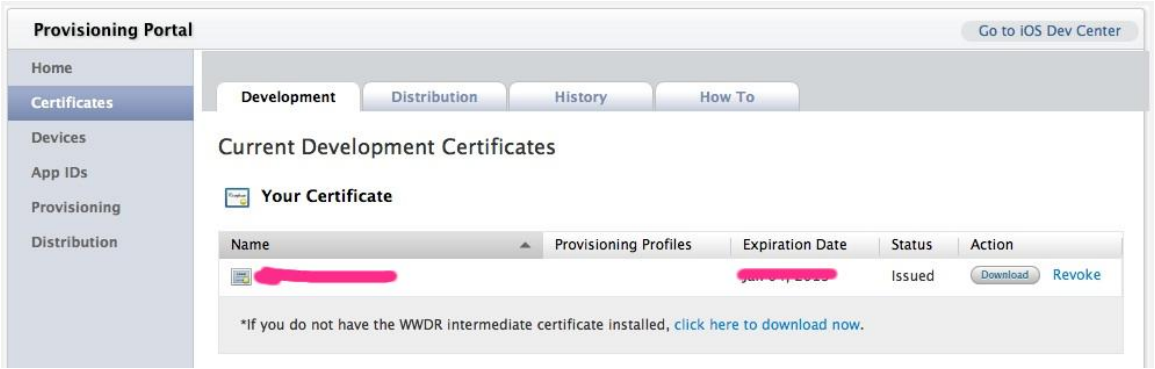
在下面的畫面中選擇前面儲存的 CertificateSigningRequest.certSigningRequest 檔案:



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:

最後雙擊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: [redacted]

Hi: [redacted]

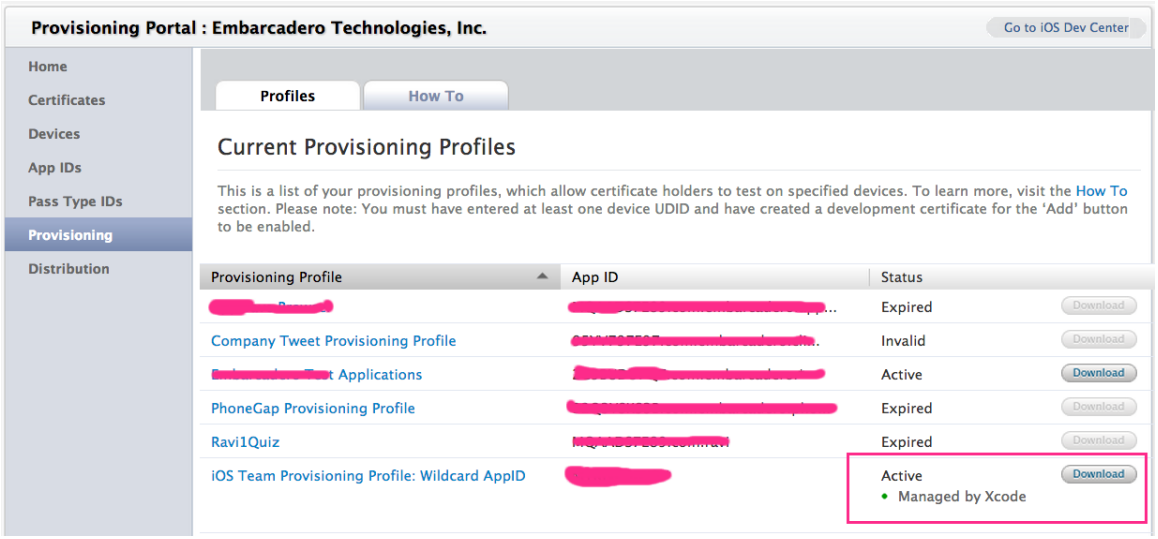
I have downloaded my ios_development.cer, so, could you register it at iOS portal?

Device Name: Gordon iPad Mini
UUID: a[redacted]1

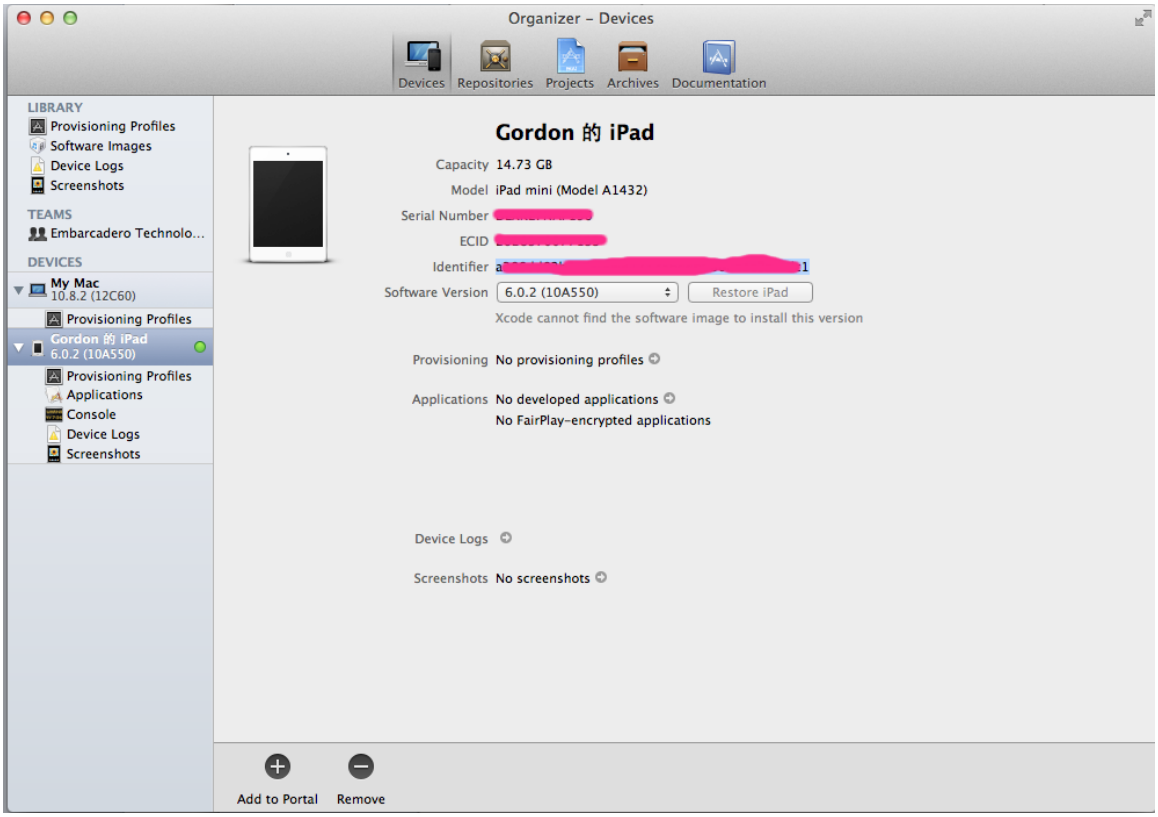
Thanks.

Cheers
Gordon

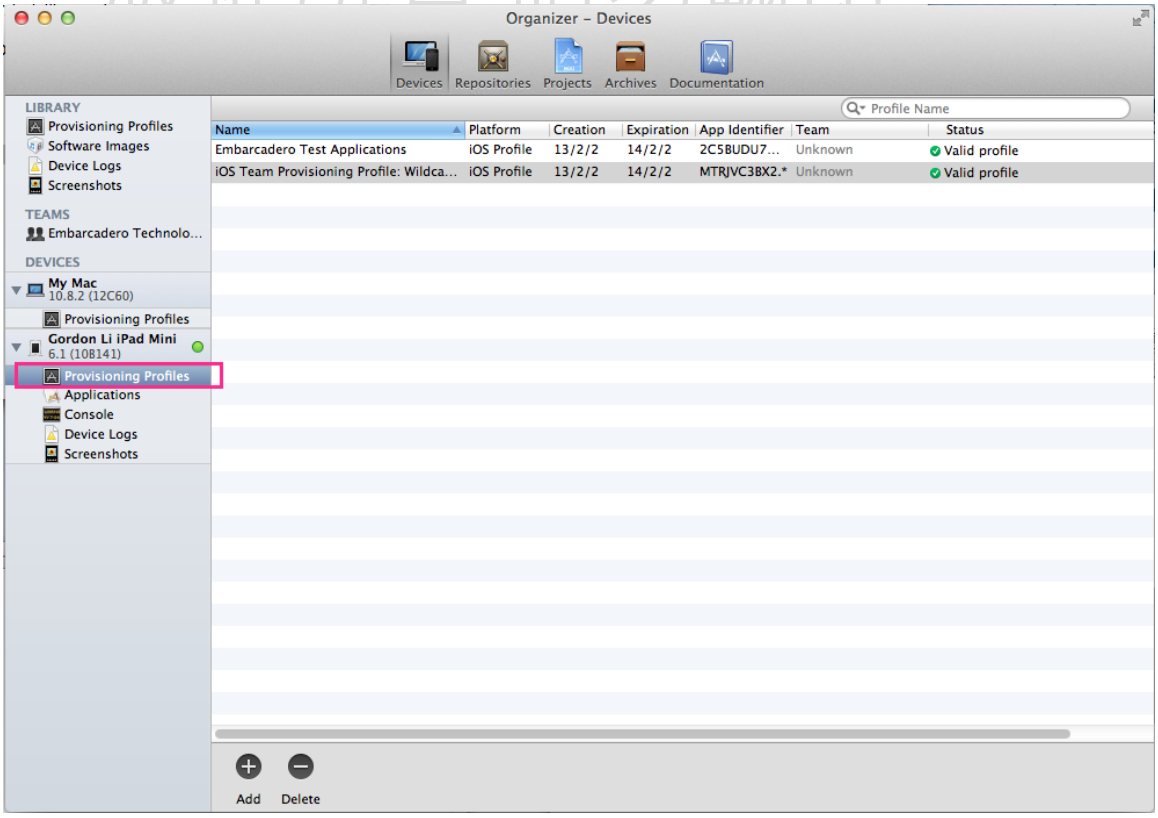
最後到 Provisioning 頁次下載可使用的認證檔案，再雙擊它以便把認證資訊匯入到 XCode 中:



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:



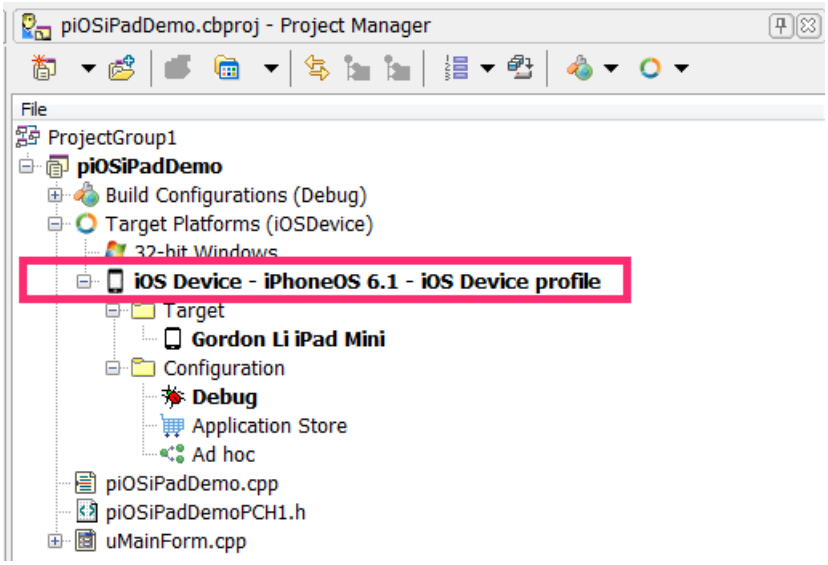
筆者使用 iPad Mini 執行設定程式，於一般 | 描述檔項目就可以看到授權認證資訊被分發到筆者的 iPad Mini 中:

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 C++Builder for iOS 部署和分發 iOS App 了。

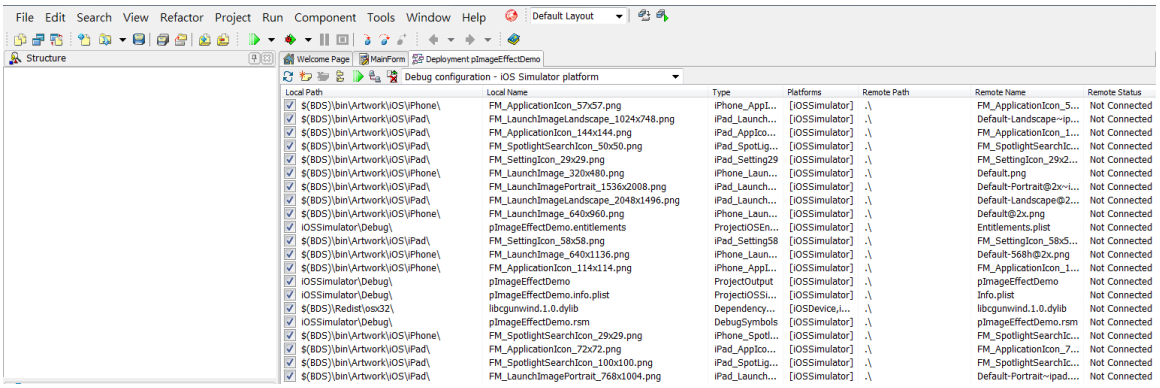
使用 C++Builder for iOS 部署和分發 iOS App 非常的簡單，下一節就會說明。

使用部署管理員分發您的 iOS App

對於簡單的iOS App專案，讀者只需要在專案管理員的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，因此我們只需要在專案經理中選擇 iOS 遠端組態，再執行此範例 iPad App 之後，C++Builder for iOS 就會自動編譯和分發了

> 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

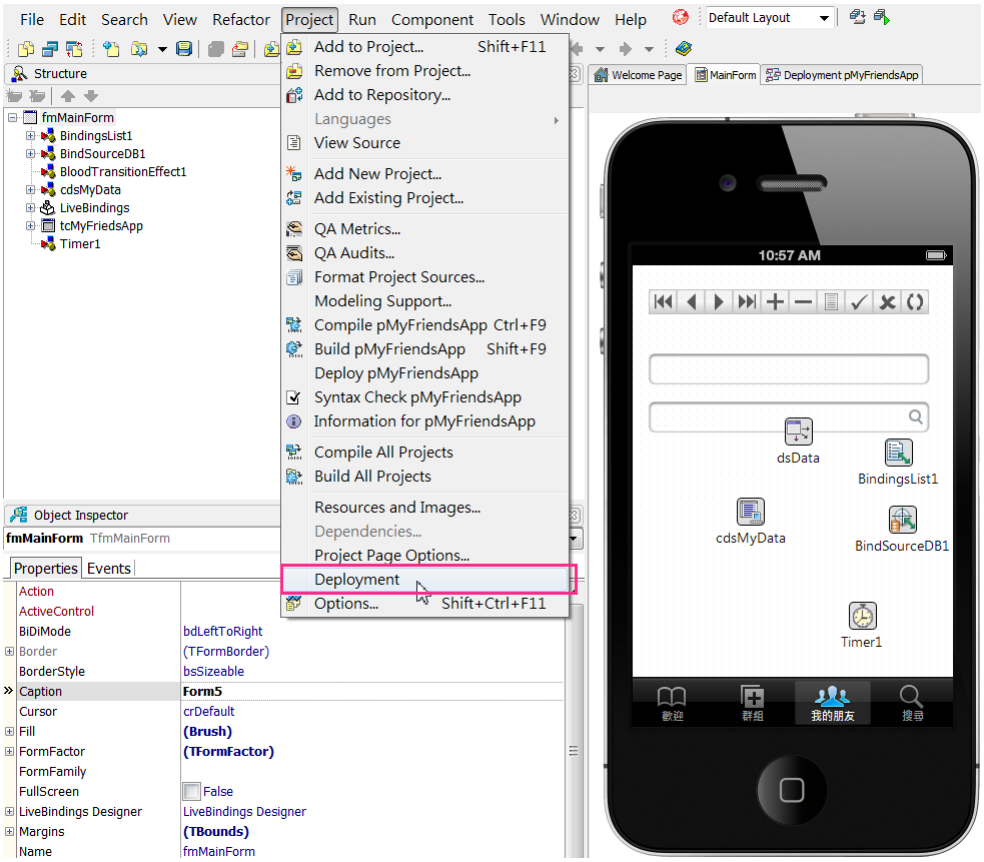
好了，現在讀者應該瞭解了使用 C++Builder for iOS 分發 iOS App 是非常簡單的工作了。

7 分發複雜的 iOS App

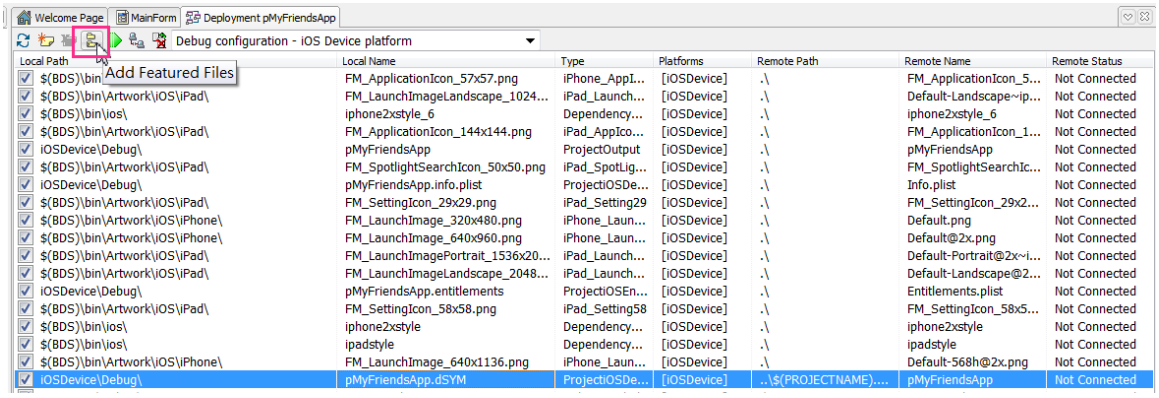
我們即將進入本書最後的小節，討論如何分發除了 App 本身之前還需要分發其他額外的檔案。

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:

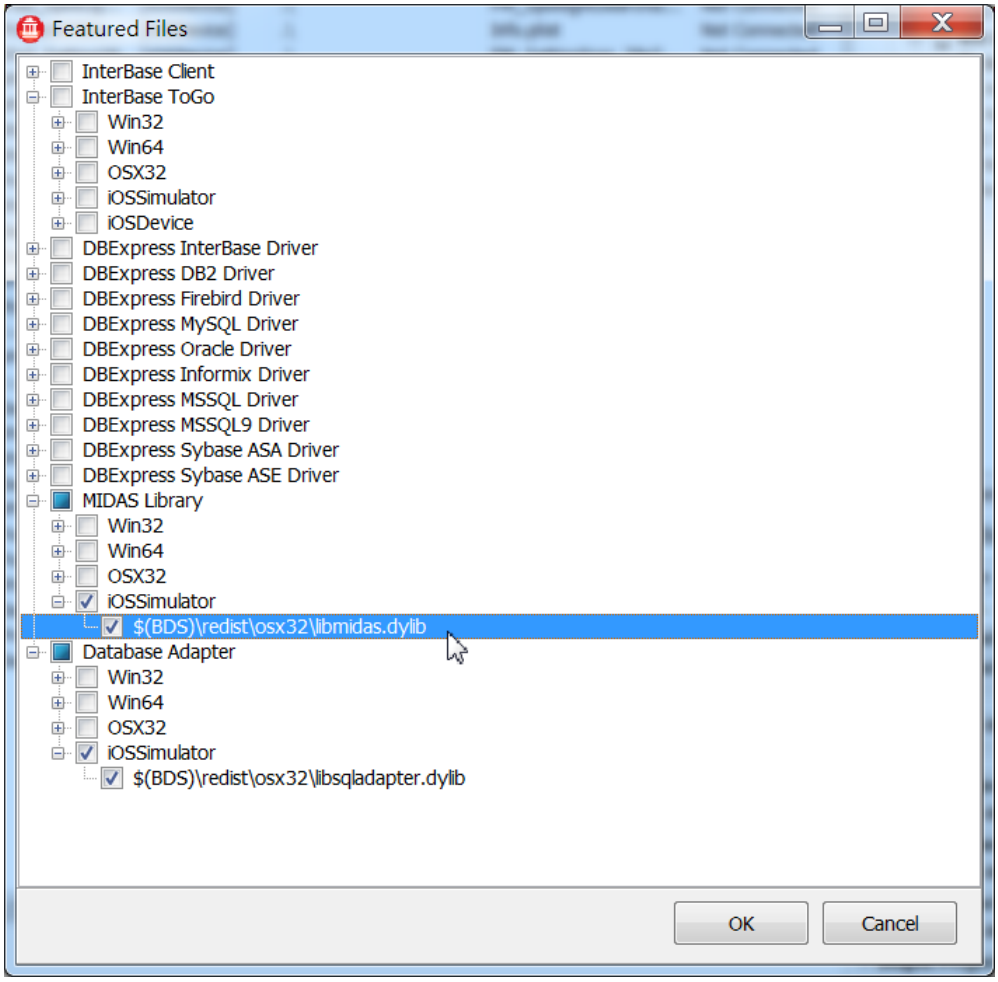


分發管理員接著會顯示如下的視窗，顯示目前這個範例 App 在分發時所有相關的檔案。請點選分發管理員視窗左上方的『Add Featured Files』圖像 ，如下所示:

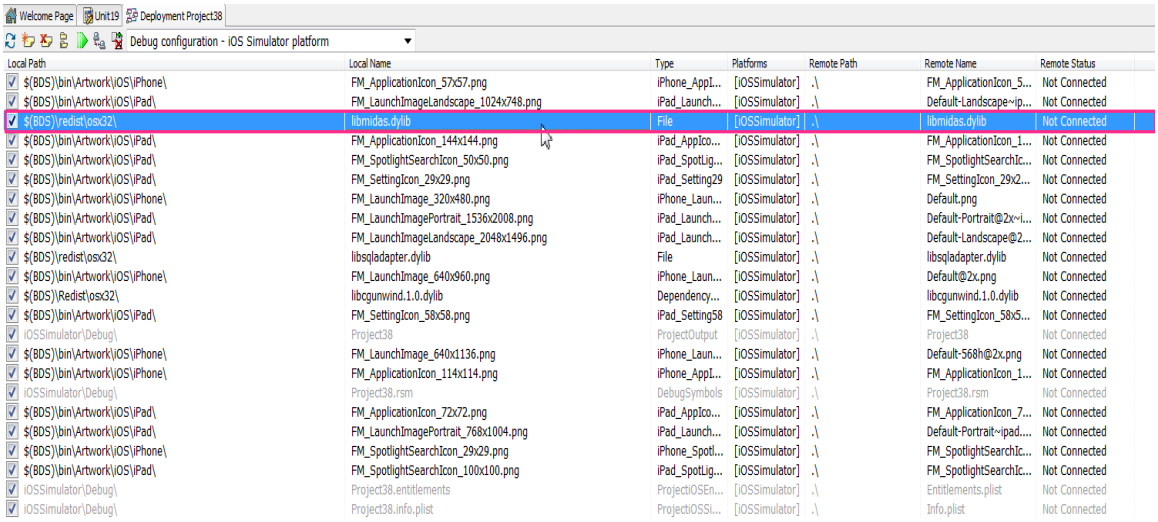


點選『Add Featured Files』圖像之後整合發展環境會顯示 Featured Files 對話盒，您可以從其中點選您需要分發的功能檔案。例如現在我們需要分發 DataSnap 功能的相關檔案，因此請點選 Featured Files 對話盒中的 Midas Library 選項，從其下再勾選 iOSSimulator 項目，在 iOSSimulator 項目下您

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列表中，如下所示:

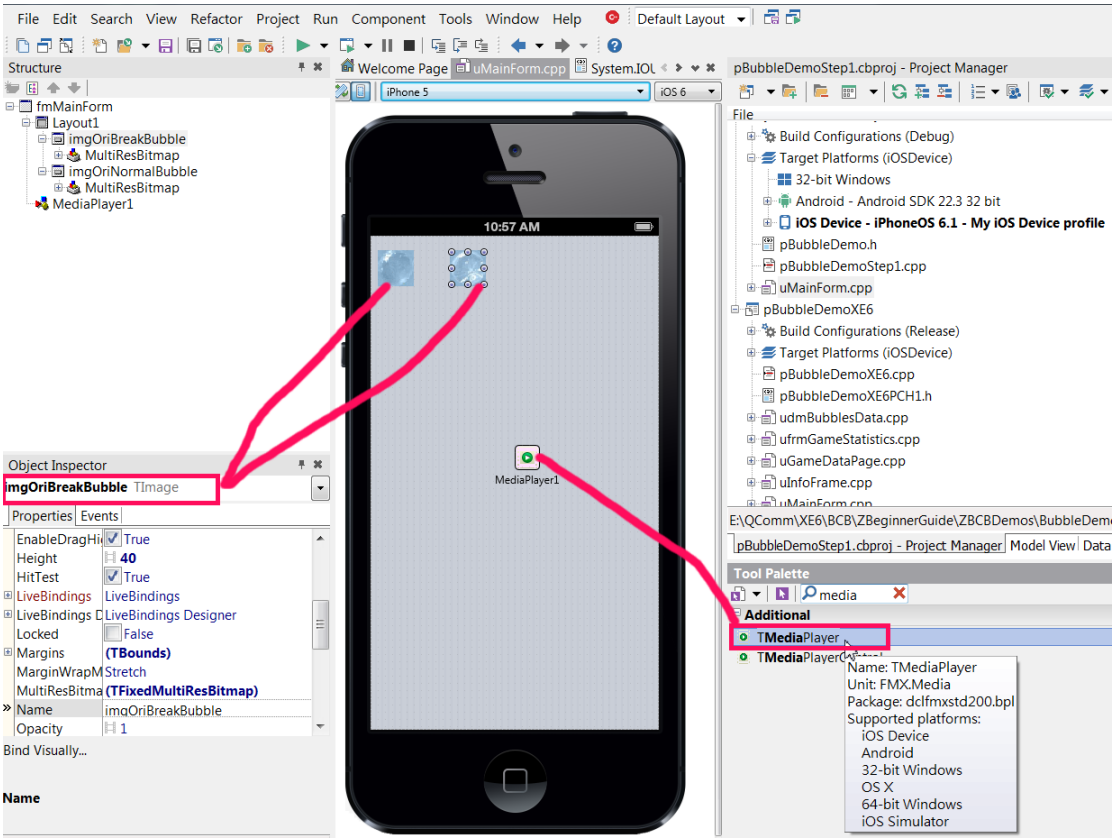


如此就完成了分發額外功能檔案的工作，現在請在專案管理員中點選 iOS 設備的組態，編譯並且執行此範例 App，那麼這個範例 App 應該就可以分發到您的 iOS 設備中執行了。例如下面的畫面就是筆者把這個範例 App 分發到筆者的 iPad Mini 中執行的結果，這個範例 App 成功的在 iPad Mini 中執行，處理資料並且瀏覽資料指定的網站了。

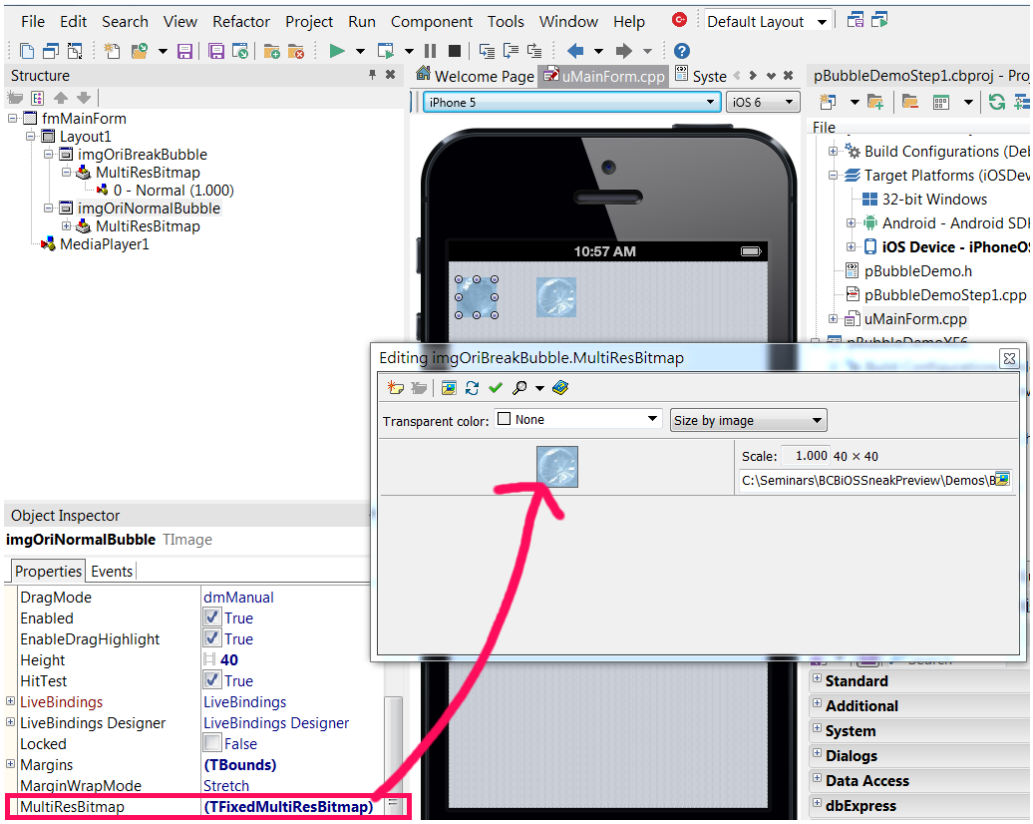
8 用 C++Builder 寫個遊戲吧

C/C++的長處之一就是快速的執行速度，為了展示 BCB 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 BCB 開發 iOS App 的技巧。

首先在 BCB 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:



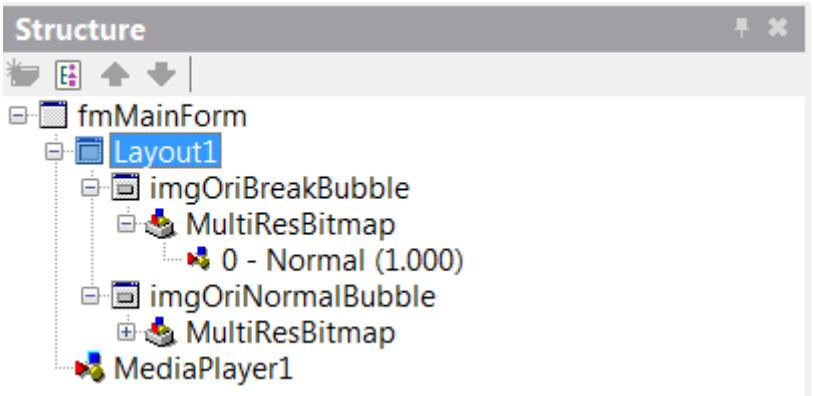
並且在物件檢視器中點選 TImage 元件的 MultiResBitmap 特性載入 2 個泡泡圖像，如下所示：



並且特定 2 個 TImage 元件的 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:



8-1 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() 方法設定捏破泡泡聲音檔的位置，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：

```
void __fastcall TMainForm::FormActivate(TObject *Sender)
{
  SetupBubbleSound();
  SetupBubbles();
  imgOriNormalBubble->Visible = false;
  imgOriBreakBubble->Visible = false;
}
```

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，012 行設定拷貝正常泡泡圖像物件的 Parent 是 Layout1，012/013 行設定這個拷貝正常泡泡圖像物件的正確位置，015 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，016 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   void TMainForm::SetupBubbles()
002   {
003     if (!bSetup)
004     {
005       TImage *pCloneImage;
006
007       for (int iRow = 0; iRow < IROWS; iRow++)
008       {
009         for (int iCol = 0; iCol < ICOLS; iCol++)
010         {
011           pCloneImage = (TImage *) imgOriNormalBubble->Clone(this);
012           pCloneImage->Parent = Layout1;
013           pCloneImage->Position->X = iCol * 40;
014           pCloneImage->Position->Y = iRow * 40;
015           pCloneImage->OnClick = OnBubbleClick;
```

```
016           pBubbles[iRow][iCol] = pCloneImage;
017           BubblePoppedStatus[iRow][iCol] = false;
018         }
019       }
020       bSetup = true;
021     }
022   }
```

而 BubblePoppedStatus 和 pBubbles 則是宣告在表頭檔中：

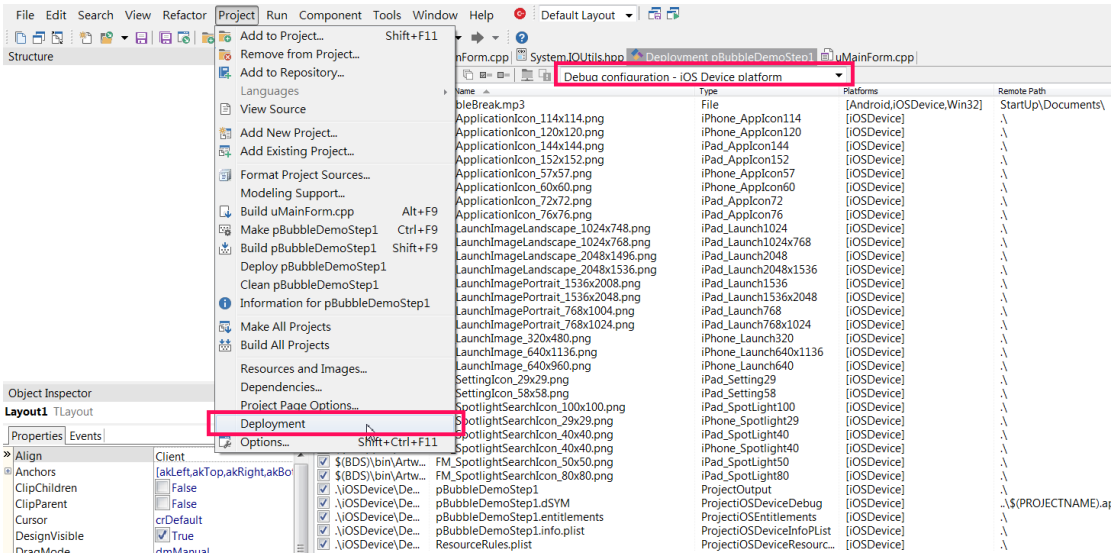
```
    bool BubblePoppedStatus[IROWS][ICOLS];
    TImage* pBubbles[IROWS][ICOLS];
```

SetupBubbleSound()方法的工作則是設定捏破泡泡時要播放的聲音檔位置，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 元件載入聲音檔正確的路徑。由於部署 iOS App 時只能把 App 需要使用的其他檔案部署在 App 的沙盒中而不能隨意部署 App 要使用的額外檔案，因此對於像本範例使用的聲音檔，我們應該把它部署在 App 的 Documents 目錄中。

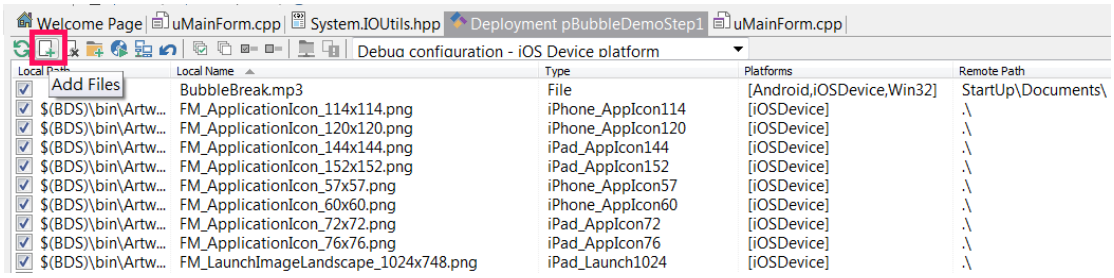
因此在 SetupBubbleSound()方法的 003 行我們可呼叫 GetHomePath()方法取得 iOS App 本身的根目錄，接著再於它的 Documents 目錄下載入聲音檔"BubbleBreak.mp3"，004 行判斷如果聲音檔"BubbleBreak.mp3"存在的話就設定 TMediaPlayer 元件的 FileName 特性值為此聲音檔，否則就顯示一警告訊息：


```
001   void TMainForm::SetupBubbleSound()
002   {
003     String sFileName =System::Ioutils::TPath::GetDocumentsPath() +
PathDelim + "BubbleBreak.mp3";
004     if (FileExists(sFileName))
005     MediaPlayer1->FileName = sFileName;
006     else
007     ShowMessage(sFileName  + "Not Exist");
008   }
```

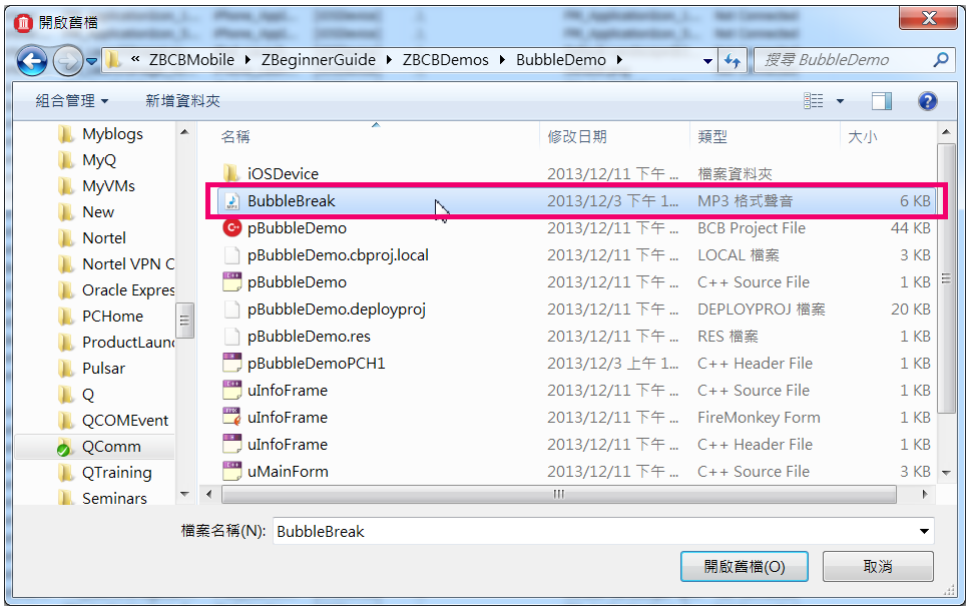
好了，到這裡此範例 App 就可以先執行看看了，但在執行之前我們需要連同聲音檔"BubbleBreak.mp3"一起部署到 iPhone 5 手機中。請在 IDE 中點選 Project|Deployment 選單，IDE 會啟動部署精靈，如下所示：



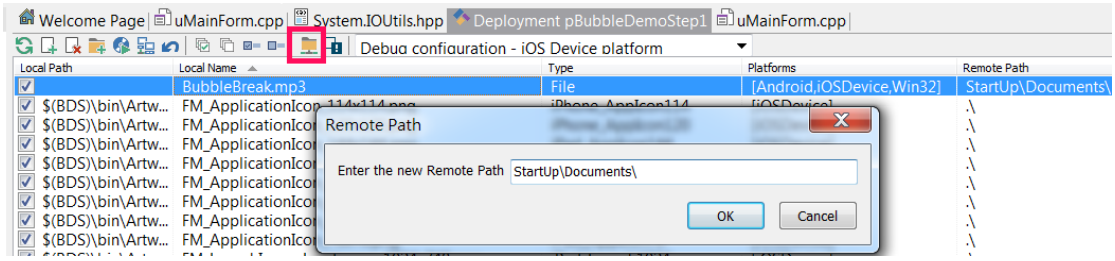
在部署精靈中點選左方上的『Add Files』按鈕  以便加入聲音檔"BubbleBreak.mp3"：



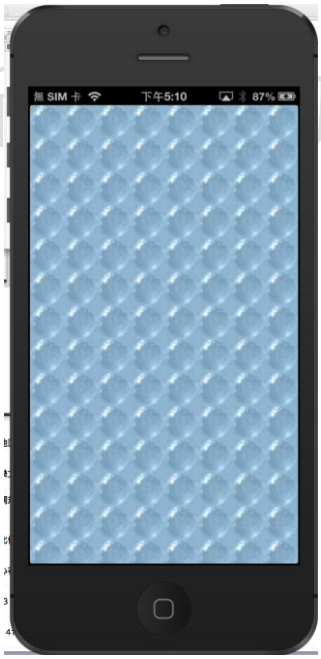
部署精靈此時會顯示開啟舊檔對話盒讓您載入聲音檔：



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 把聲音檔的遠端部署位置設定為『StartUp\Documents\』，如下所示：



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：



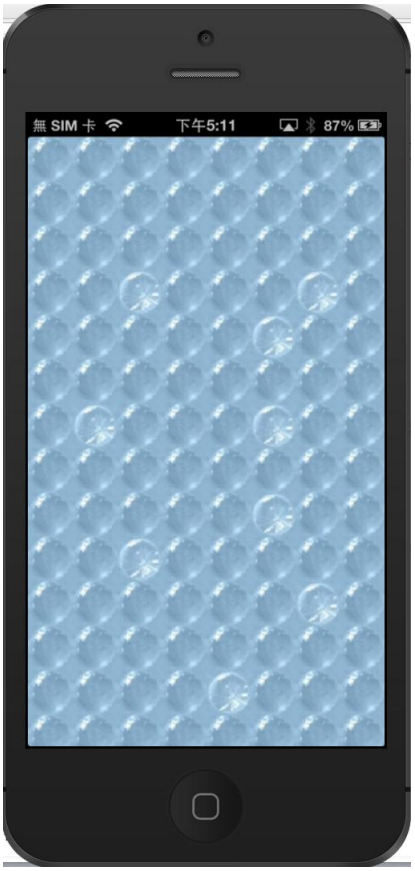
8-2 點選捏破泡泡

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，007 行把聲音檔的播放位置重置回開始的位置，008 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 元件播放聲音檔"BubbleBreak.mp3"：

```
001   void __fastcall TMainForm::OnBubbleClick(TObject *Sender)
002   {
003     int iRow = ((TImage *) Sender)->Position->Y / BUBBLESIZE;
004     int iCol= ((TImage *) Sender)->Position->X / BUBBLESIZE;
005
006     MediaPlayer1->Stop();
007     MediaPlayer1->CurrentTime = 0;
008     if (!BubblePoppedStatus[iRow][iCol])
009     {
010       BubblePoppedStatus[iRow][iCol] = true;
011       ((TImage *)
Sender)->MultiResBitmap->Assign(imgOriBreakBubble->MultiResBitmap);
012       MediaPlayer1->Play();
013     }
014   }
```

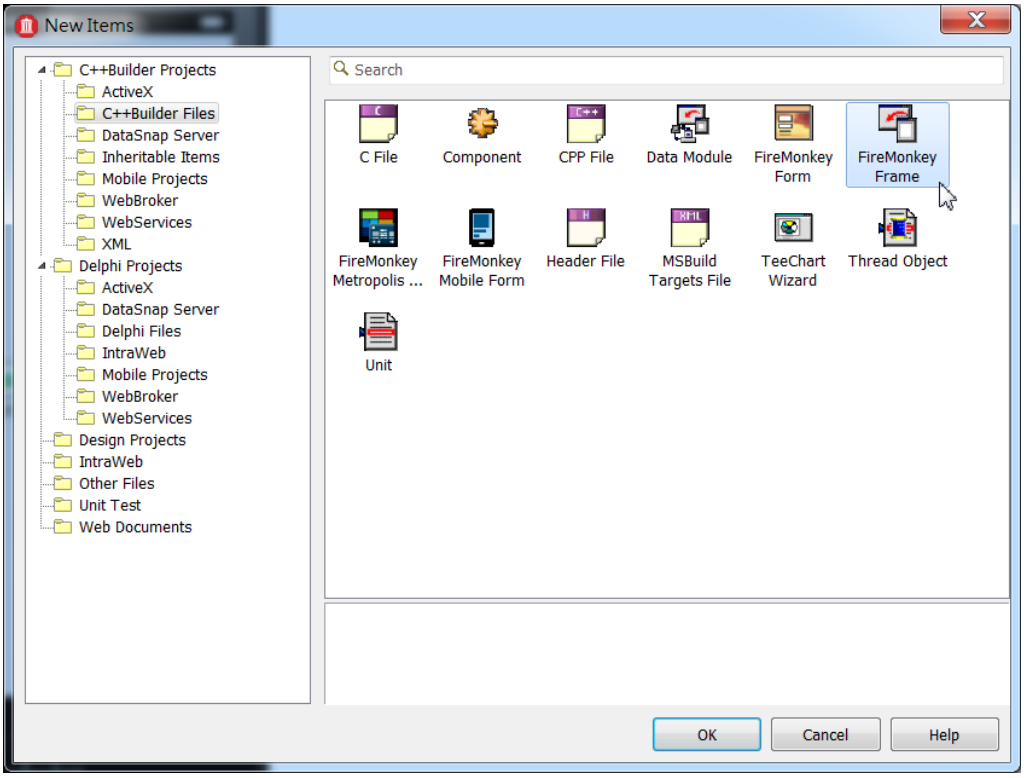

再次編譯和執行此範例 App 並且隨意點選泡泡就可以看到如下的畫面被點選的泡泡就會顯示被捏破並且會聽到泡泡被捏破的聲音了。



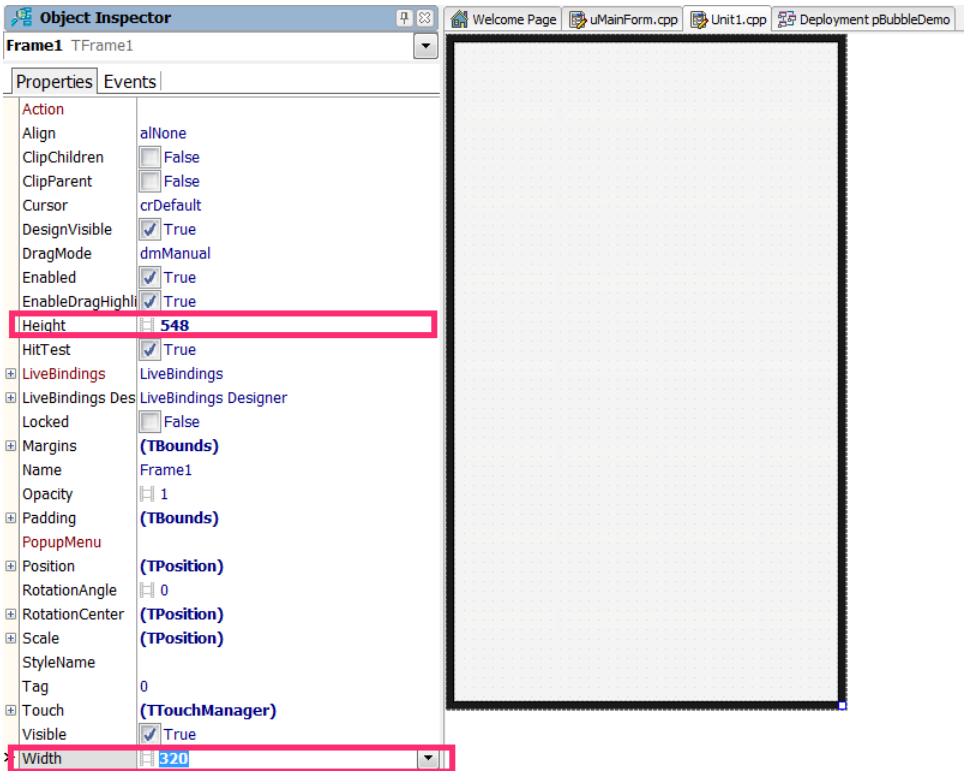
8-3 加入手勢功能

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。

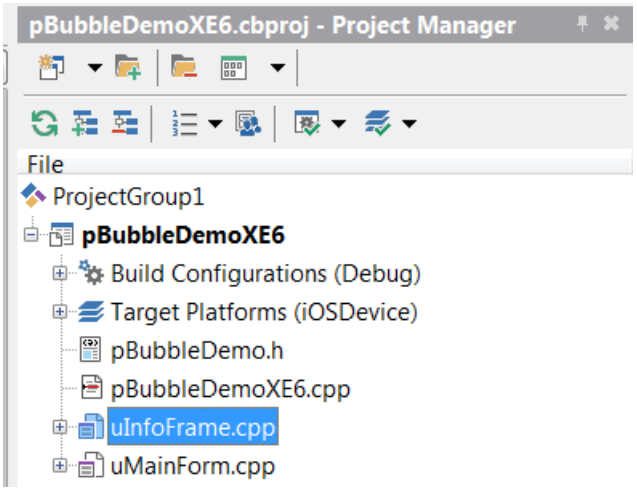
現在請在 IDE 中點選 New Items 快捷鍵於 New Items 對話盒中 C++Builder Files 項目中選擇建立 FireMonkey Frame 物件，如下圖所示：



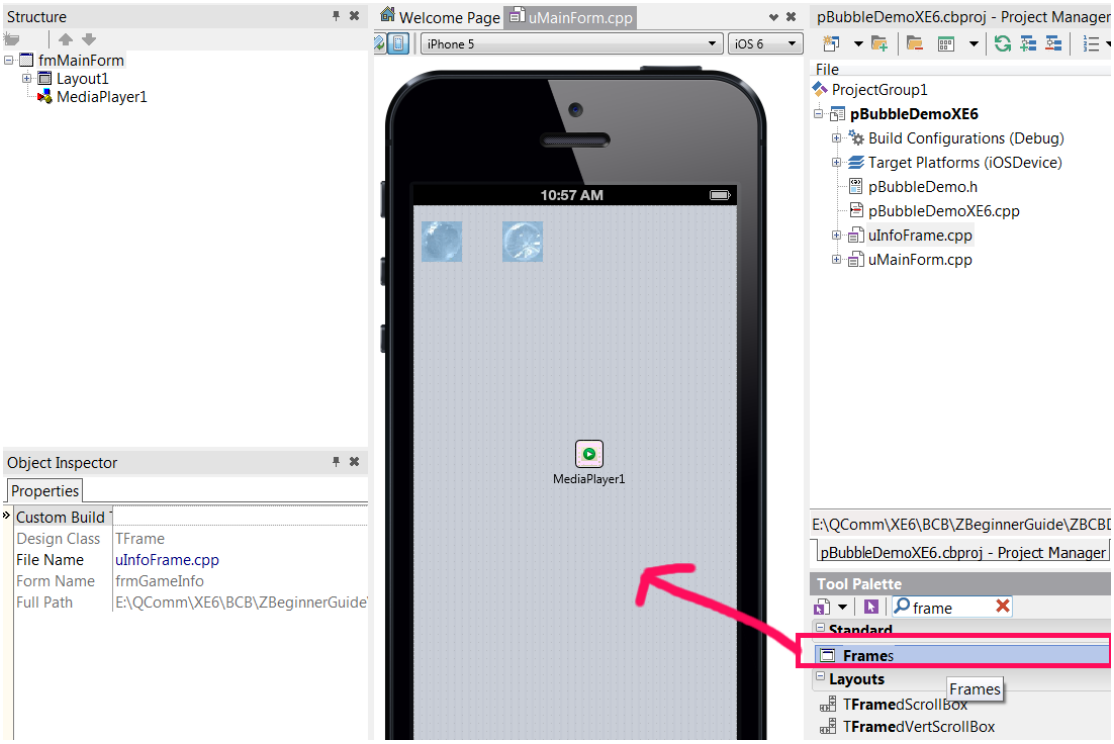
點選建立 FireMonkey Frame 物件之後 IDE 便會建立一個空白的 Frame 表單，請在物件檢視器中設定它的 Height 和 Width 特性值和主表單中 Layout 元件一樣的 Height 和 Width 特性值，如下所示：



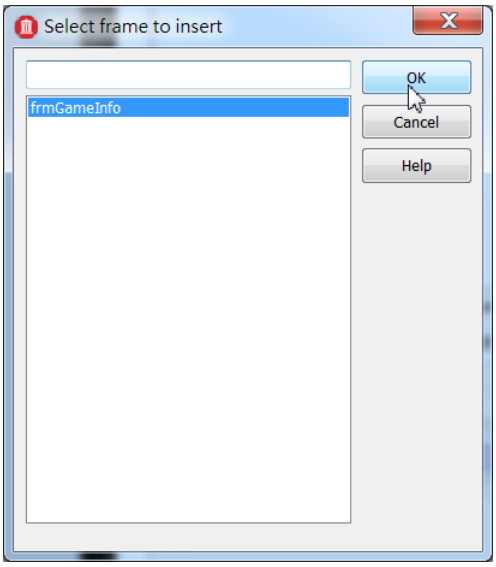
最後設定它的 Name 特性值為 frmGameInfo 並且以 uInfoFrame 儲存這個 FireMonkey Frame 物件，此時專案管理員的內容應如下所示：



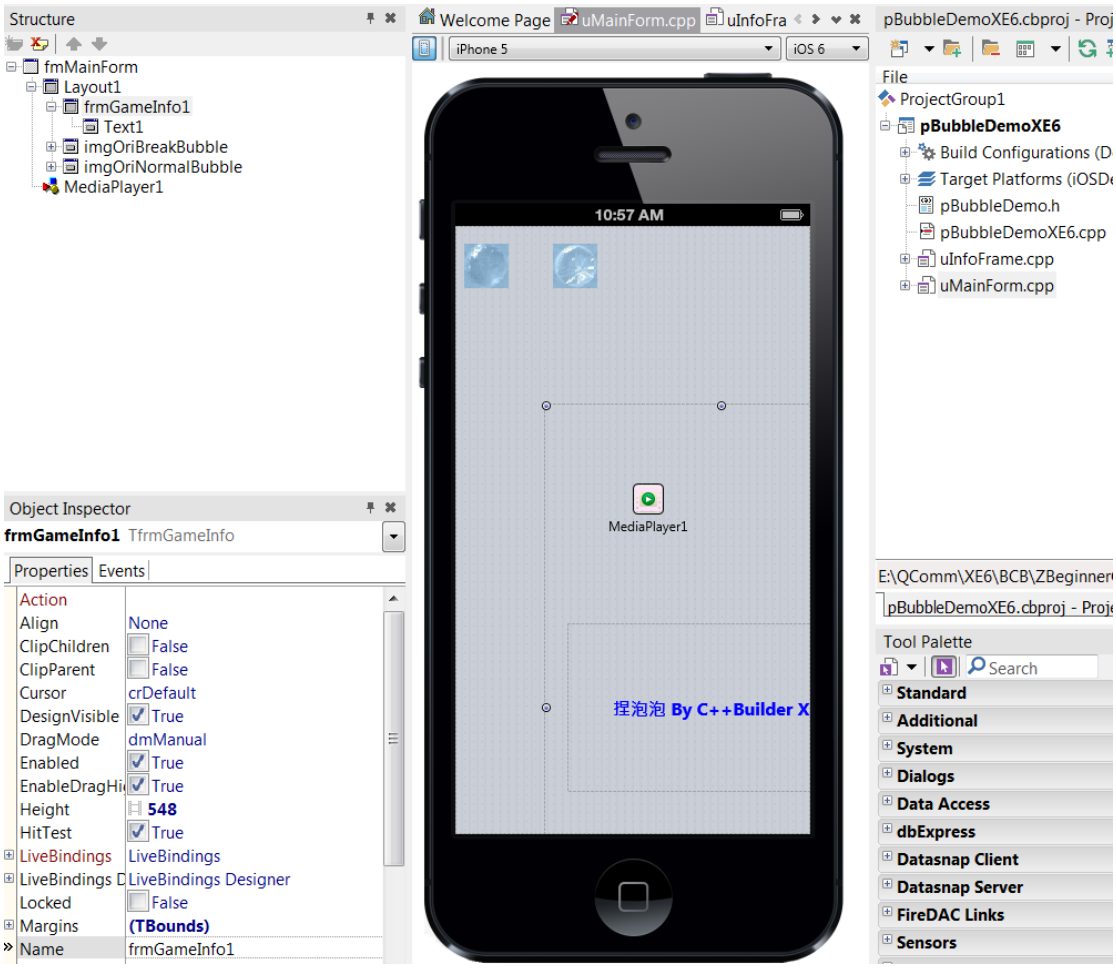
接著回到專案中的主表單，在工具盤中找到 Frame 元件並且把它拖曳到主表單中如下所示：



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：

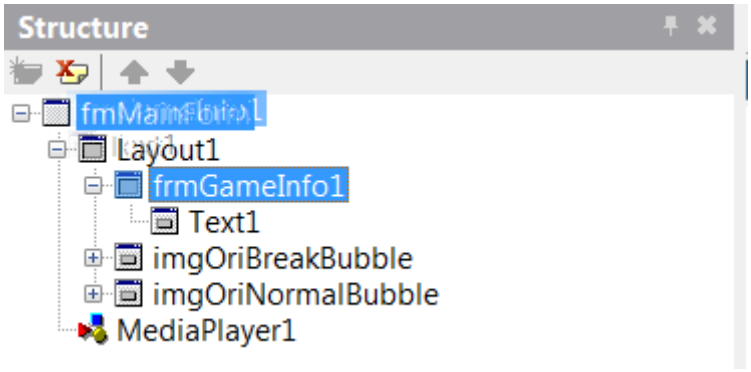


在上面的對話盒中選擇了 frmGameInfo 之後我們就可以在主表單中看到 frmGameInfo 也顯示在主表單中了，如下所示：

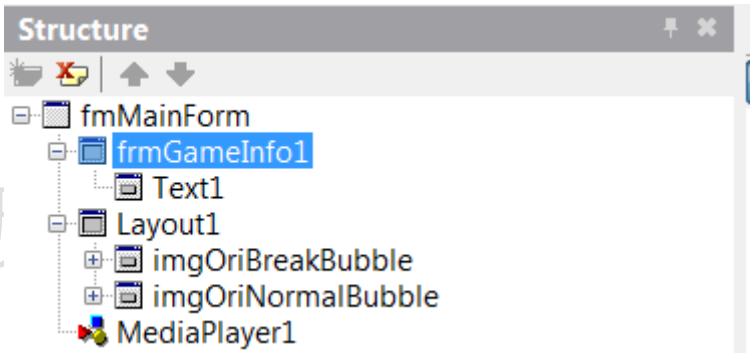


現在請在 IDE 左上方的架構視窗中查看 frmGameInfo 的父代元件是什麼，如果 frmGameInfo 是在 Layout1 元件之下就代表 Layout1 是它的父代元

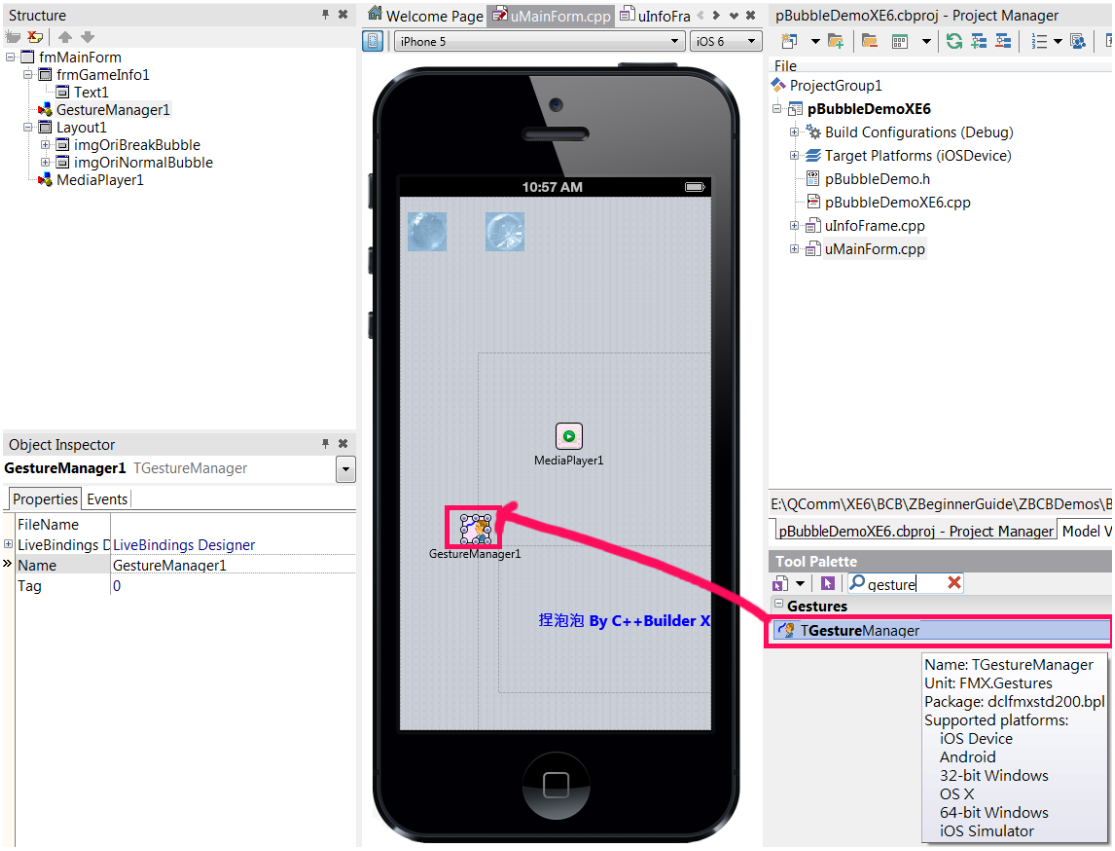
件，那麼就請在架構視窗中點選 frmGameInfo，在保持按著滑鼠左鍵的狀況下拖曳 frmGameInfo 到 MainForm 之下再釋放滑鼠左鍵讓 MainForm 成為 frmGameInfo 的父代元件，如下所示：



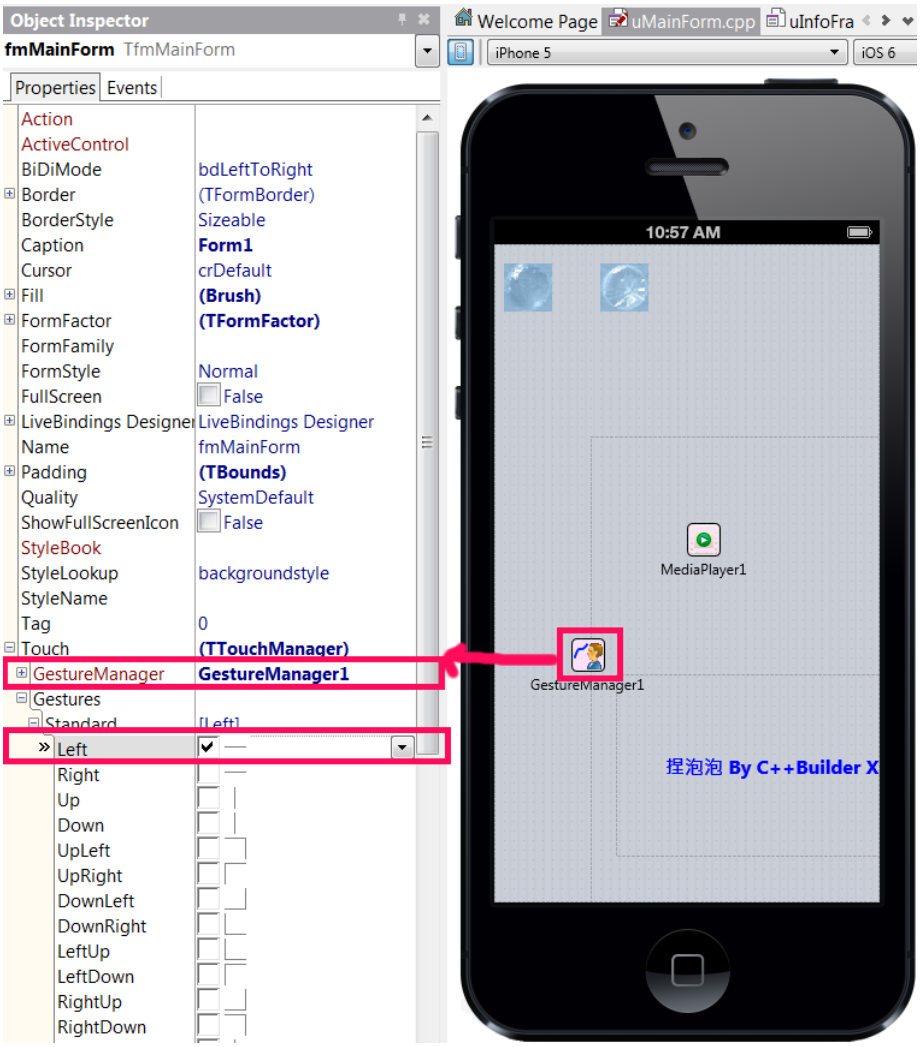
拖曳完成之後架構視窗應該如下所示：



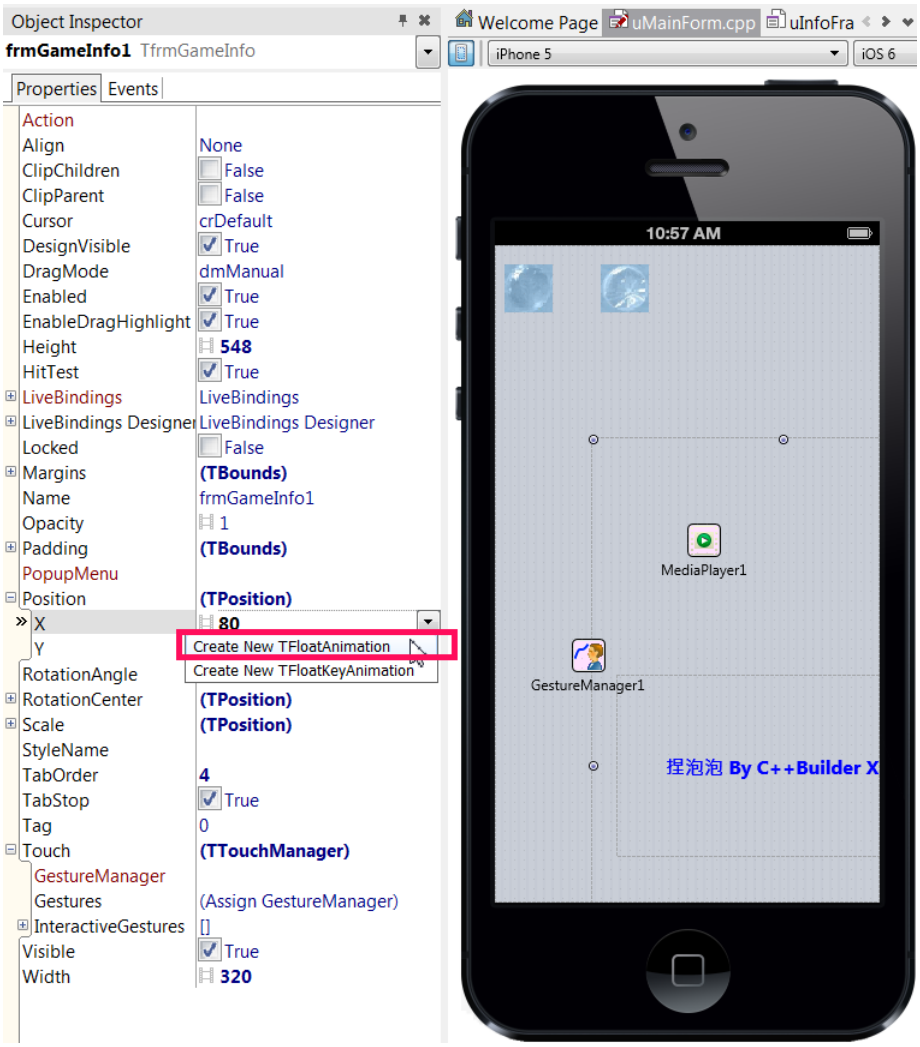
好了，接下來再於主表單中拖入 TGestureManager 元件準備加入處理手勢的功能，如下所示：



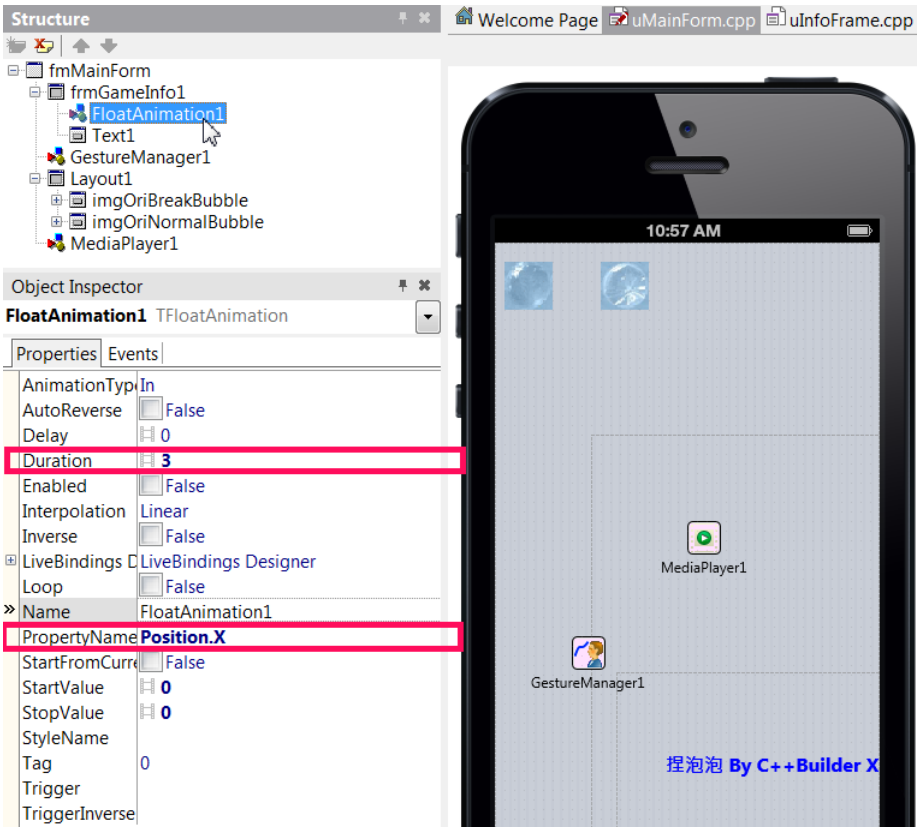
在物件檢視器中設定 MainForm 的 GestureManager 特性值為剛才拖入的 TGestureManager 元件，展開 MainForm 的 Gestures 特性在它的 Standard 子特性中勾選 Left 子特性代表主表單 MainForm 將處理向左滑動的手勢，如下所示：



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 左上方的架構視窗中點選新加入的 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：



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    void __fastcall TMainForm::FormGesture(TObject *Sender, const
TGestureEventInfo &EventInfo,
002            bool &Handled)
003    {
004      if (EventInfo.GestureID.get() == sgiLeft )
005      {
006          faniFrameX->Enabled = false;
007          faniFrameX->StartValue = Layout1->Width;
008          faniFrameX->StopValue = 0;
009          faniFrameX->Enabled = true;
010          Handled = true;
011      }
012    }
```

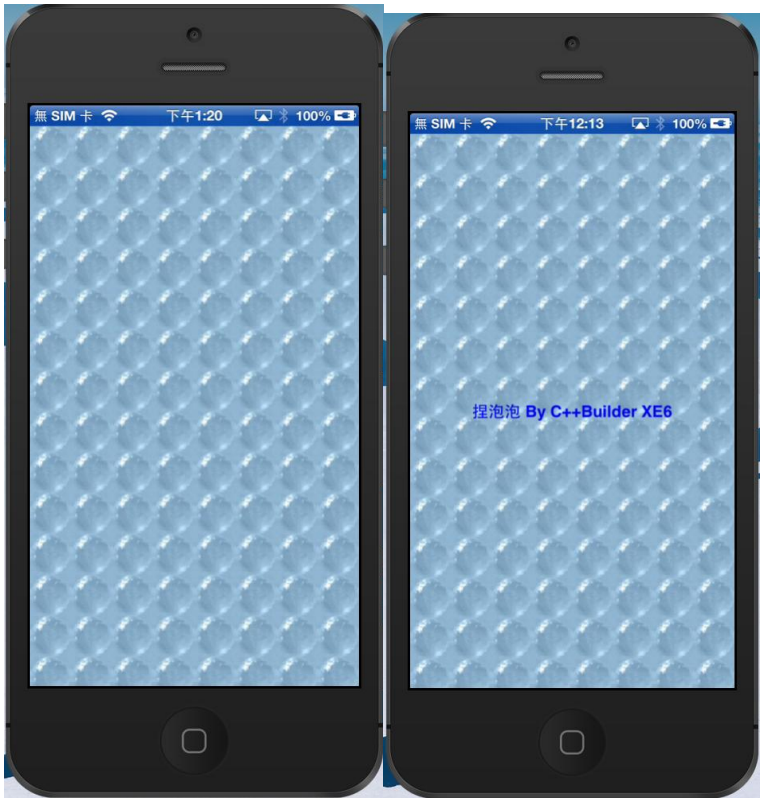
當玩家在手機螢幕做出向左滑動的手勢後 TGestureManager 就會觸發 MainForm 的 OnGesture 事件處理函式。OnGesture 事件處理函式的第 2 個參數 TGestureEventInfo 包含此手勢的相關資訊，在 TGestureEventInfo 中的 GestureID 變數代表使用者做出的手勢種類，因此我們只要判斷 GestureID 就可以知道玩家是不是做出了左向滑動的手勢。

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：

```
void TfmMainForm::SetupGameInfoFrame()
{
  frmGameInfo1->Position->X = Layout1->Width;
  frmGameInfo1->Position->Y = 0;
}
```

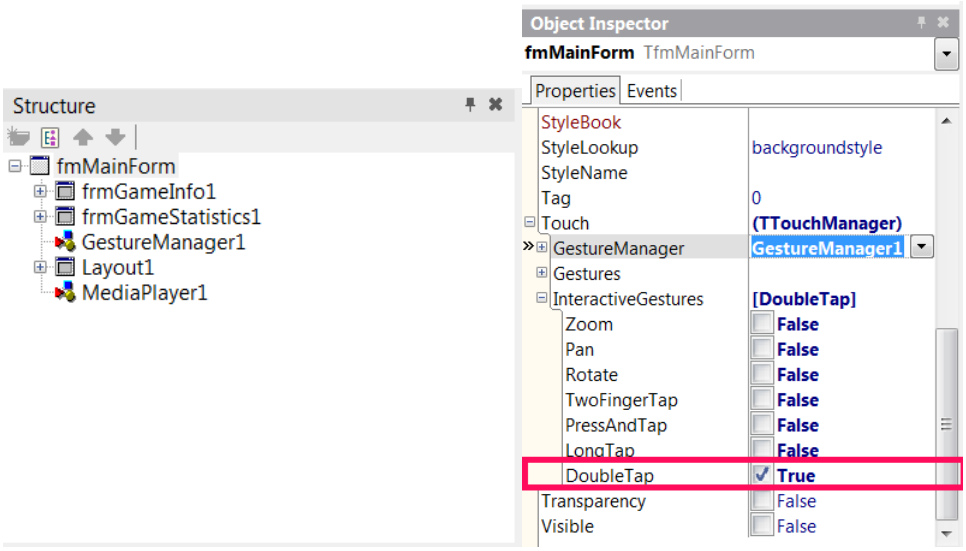
現在編譯和執行此範例 App 並且隨意點選泡泡，再使用手指向手機螢幕的左方滑動，就可以看到 frmGameInfo 物件從右向左慢慢的滑動出現了，如下所示：



8-4 重新開始遊戲

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.get() == igiDoubleTap )
      {
        ResetGame();
      }
```

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：

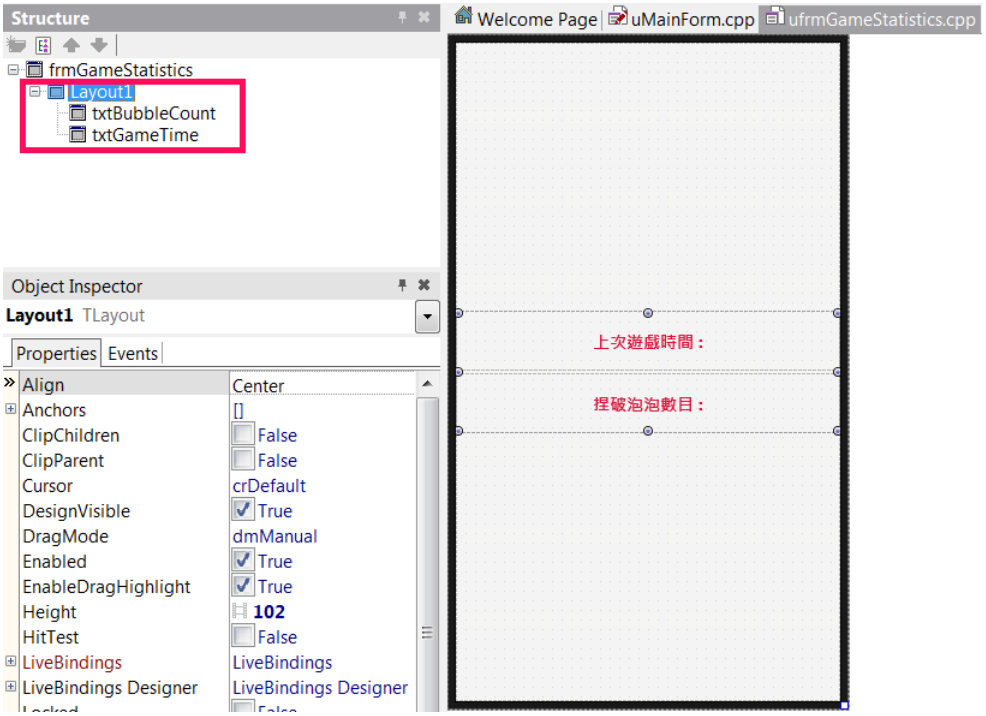
```
void TfmMainForm::ResetGame()
{
  SetupGameInfoFrame();
  for (int iRow = 0; iRow < IROWS; iRow++)
  {
    for (int iCol = 0; iCol < ICOLS; iCol++)
    {
      if (BubblePoppedStatus[iRow][iCol])
      {

    pBubbles[iRow][iCol]->MultiResBitmap->Assign(imgOriNormalBubble->
MultiResBitmap);
        BubblePoppedStatus[iRow][iCol] = false;
      }
    }
  }
}
```

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。

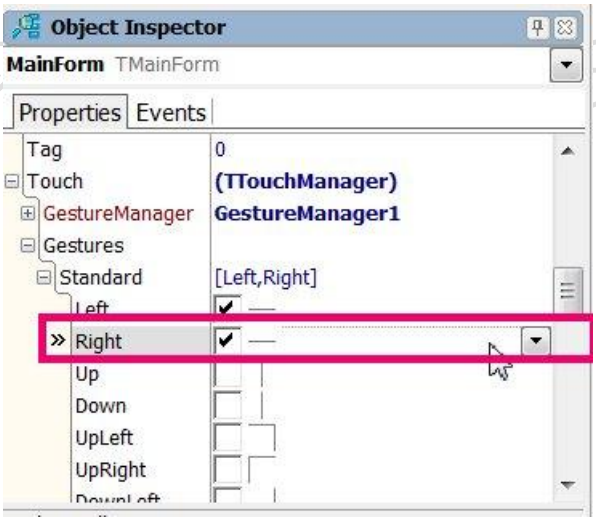
8-5 處理遊戲資訊

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。

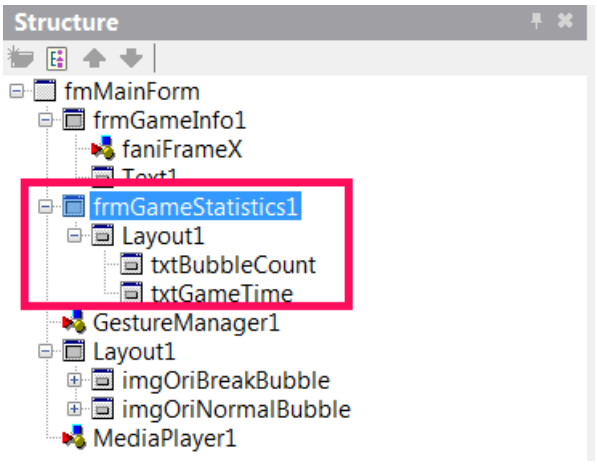
再次讓我們使用一個 Frame 物件來顯示遊戲的資訊，請使用前面介紹的方式再建立專案中第 2 個 FireMonkey Frame 物件，設定它的 Height 和 Width 特性和主表單中的 Layout1 一樣，再於其中放入一個 TLayout 元件再於其中放入 2 個 Text 物件並且設定 TLayout 元件的 Align 特性為 alCenter，設定 Name 特性為 frmGameStatistics，如下所示：



回到主表單讓主表單支援由左向右滑動的手勢：

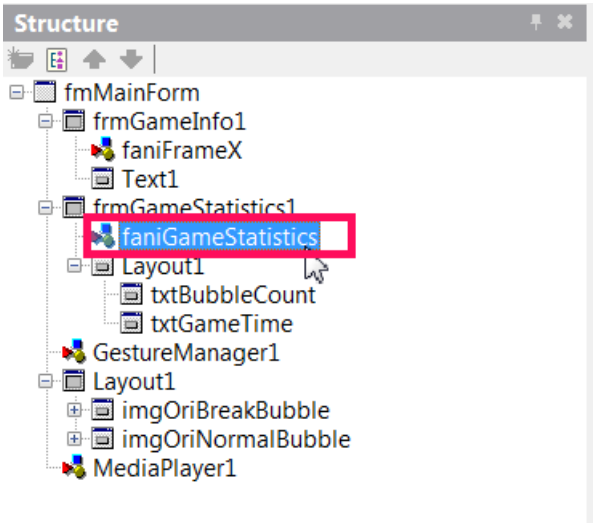


在 IDE 左上方的架構視窗中拖曳 frmGameStatistics 的父代元件為 MainForm，如下所示：



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：

```
void TMainForm::SetupGameInfoFrame()
{
  frmGameInfo1->Position->X = Layout1->Width;
  frmGameInfo1->Position->Y = 0;
  frmGameInfo1->Parent = this;

  frmGameStatistics1->Position->X = -Layout1->Width;
  frmGameStatistics1->Position->Y = 0;
  frmGameStatistics1->Parent = this;
}
```

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 沙盒的 Documents 目錄下。

因此在下面的 010 行先取得遊戲資訊組態檔案的正確路徑之後 011 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 String sBubbleGameInfo = "BubbleGameInfo.ini";
002
003    String TMainForm::GetGameInfoFile()
004    {
005      return System::Ioutils::TPath::GetDocumentsPath() + PathDelim +
sBubbleGameInfo;
```

```
006   }
007
008   void TMainForm::WriteGameInfo()
009   {
010     String sGameInfoFile = GetGameInfoFile();
011     TIniFile *pConfigFile = new TIniFile(sGameInfoFile);
012     try
013     {
014       pConfigFile->WriteString("遊戲", "時間",
DateTimeToStr(Now()));
015       pConfigFile->WriteString("遊戲", "捏破泡泡數目",
IntToStr(GetBrokenBubles())) ;
016     }
017     __finally
018     {
019       delete pConfigFile;
020     }
021   }
022
023   int TMainForm::GetBrokenBubles()
024   {
025     int iResult = 0;
026
027     for (int iRow = 0; iRow < IROWS; iRow++)
028     {
029     for (int iCol = 0; iCol < ICOLS; iCol++)
030       if (BubblePoppedStatus[iRow][iCol])
031         iResult++;
032     }
033
034     return iResult;
035   }
```

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.get() == sgiRight)
        {
```

```
    frmGameStatistics1->LoadGameStatistics(GetGameInfoFile());
            faniGameStatistics->Enabled = false;
            faniGameStatistics->StartValue = -Layout1->Width;
            faniGameStatistics->StopValue = 0;
            faniGameStatistics->Enabled = true;
            Handled = true;
        }
```

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   void TfrmGameStatistics::LoadGameStatistics(const String
sGameInfoFile)
002   {
003     if (FileExists(sGameInfoFile))
004     {
005         TIniFile *pConfigFile = new TIniFile(sGameInfoFile);
006         String sDT;
007         String sBubbles;
008         try
009         {
010           sDT = pConfigFile->ReadString("遊戲", "時間", "");
011           sBubbles = pConfigFile->ReadString("遊戲", "捏破泡泡數目",
"") ;
012         }
013         __finally
014         {
015           delete pConfigFile;
016         }
017
018         txtGameTime->Text = "上次遊戲時間 : " + sDT;
019         txtBubbleCount->Text = "捏破泡泡數目 : " + sBubbles;
020     }
021     else
022     {
```

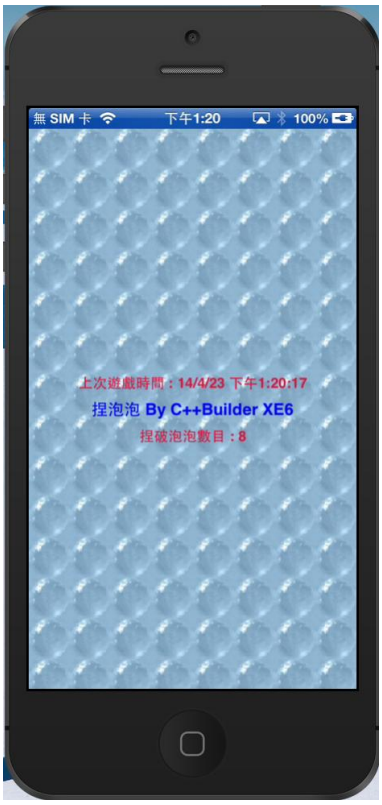
```
023          txtGameTime->Text = "上次遊戲時間 : 無";
024          txtBubbleCount->Text = "捏破泡泡數目 : 0";
025
026      }
027    }
```

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 元件中。

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：

```
void TfmMainForm::ResetGame()
{
  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：



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。

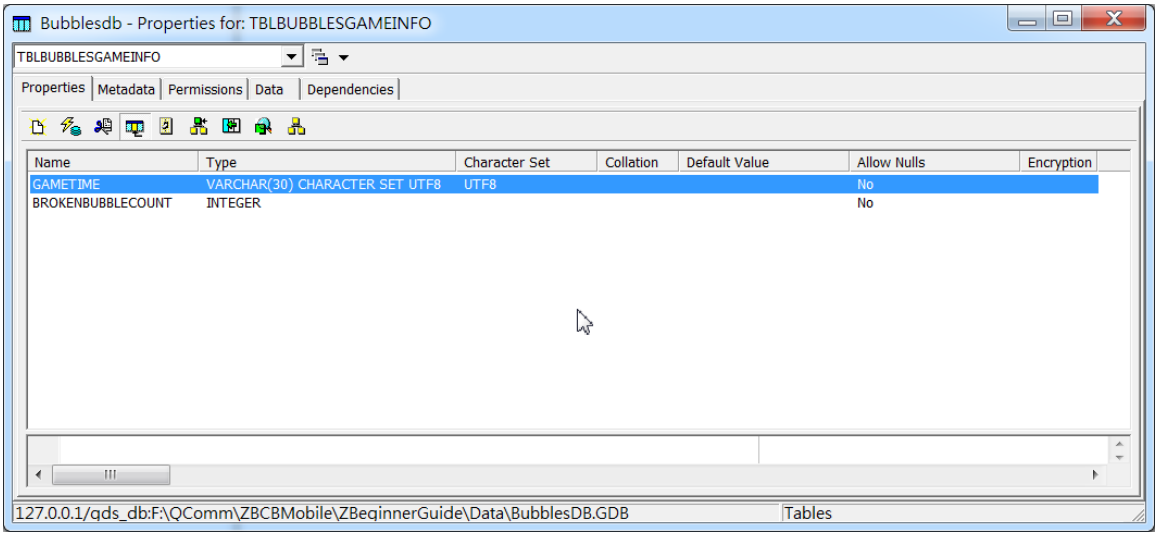
8-6 把遊戲資訊儲存到資料庫吧

C++Builder 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。

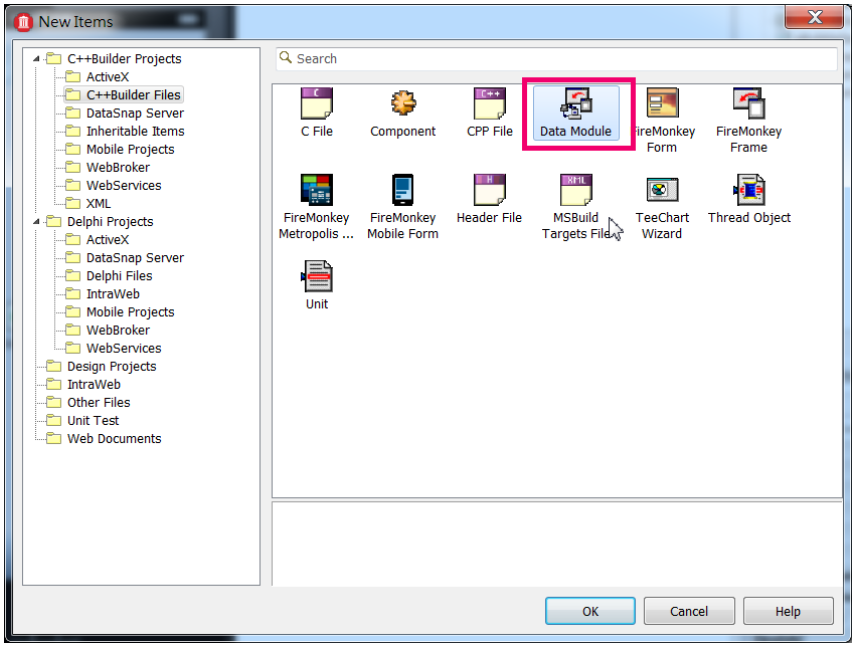
要使用 FireDAC 存取資料庫非常的簡單，基本上只需要使用 4 個 FireDAC 元件即可，下面的表格說明了本小節使用的 FireDAC 元件：

元件	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 等候游標的元件

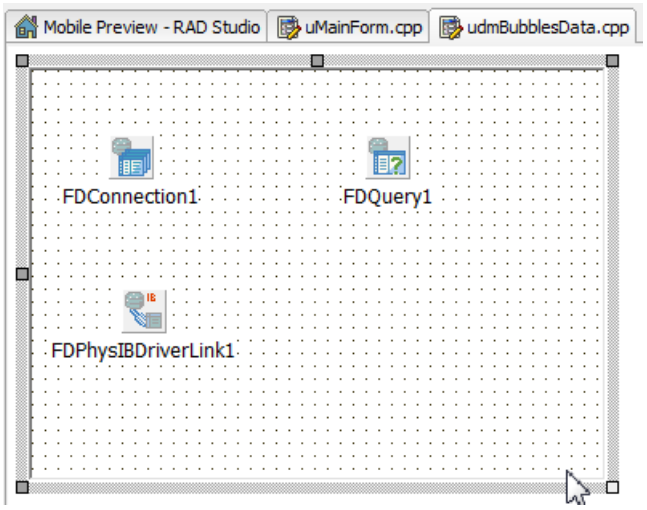
在開始之前我們需要建立一個範例 InterBase 資料庫讀者可以使用 IDE 中的 Explorer 或是 IBConsole 建立 BUBBLESDB.GDB 範例資料庫，在其中建立 TBLBUBBLESGAMEINFO 資料表並且在其中建立下面的 2 個欄位物件(讀者也可以在本書的範例中找到這個資料庫)：



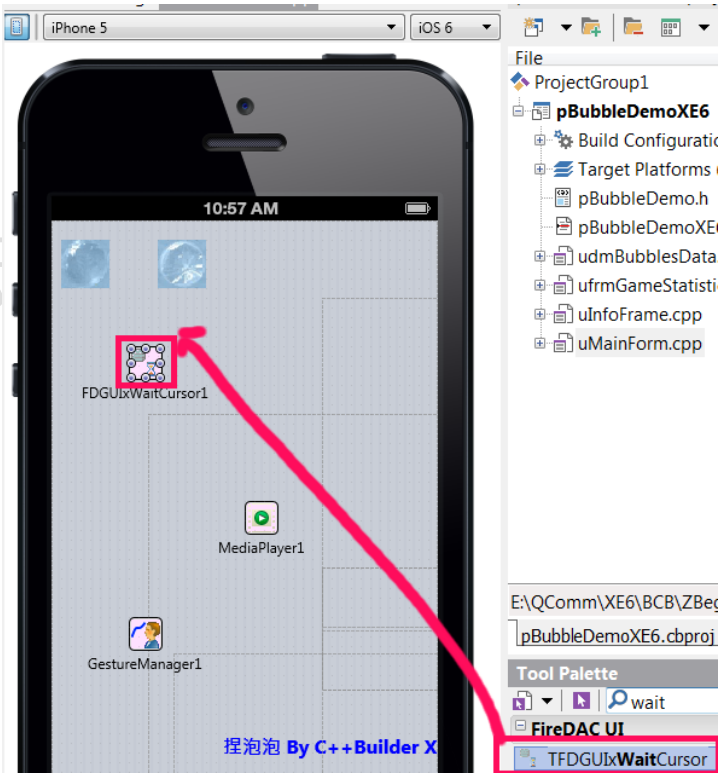
接著在 IDE 中為範例專案加入一個資料模組以便在其中使用 FireDAC 元件存取資料庫，您可以在 IDE 的 New Items 對話盒中選擇建立資料模組，如下所示：



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 元件，如下所示：

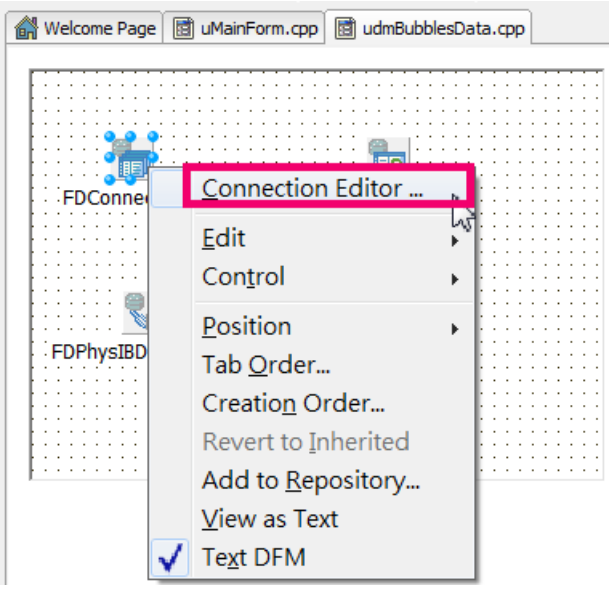


再把 TFDPhysIBDriverLink 元年放入到主表單之中：

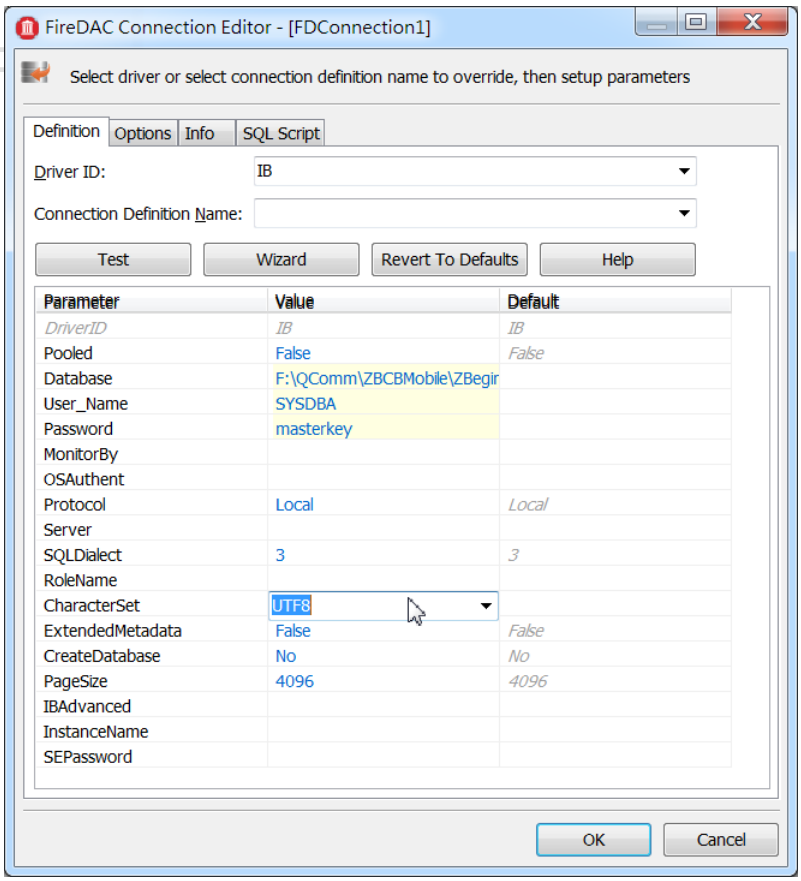


回到資料模組，現在我們要用 TFDConnection 來連結 InterBase ToGo 資料庫，由於現在我們是在設計時期因此可以先直接使用 TFDConnection 連結 InterBase ToGo，等設計好基本的工作之後再於執行時期動態連結真正部署到手機中的 BUBBLESDB.GDB 範例資料庫。

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，請在下面的元件編輯器 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 元件的 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。

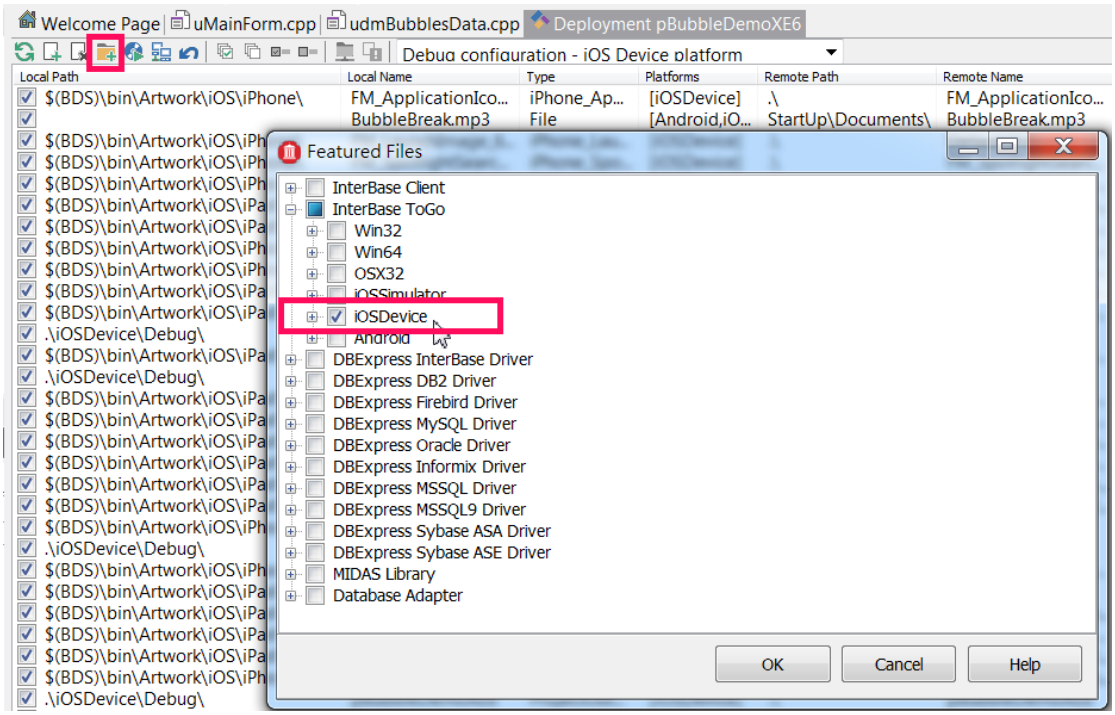
接著在資料模組的 OnCreate 事件處理函式和 OnDestry 事件處理函式中撰寫如下的程式碼開啟和 InterBase ToGo 的連結並且取得 TBLBUBBLESGAMEINFO 中的資料：

```
void __fastcall TDataModule1::DataModuleCreate(TObject *Sender)
{
  FDConnection1->Connected = true;
  FDQuery1->Active = true;
}

void __fastcall TDataModule1::DataModuleDestroy(TObject *Sender)
{
  FDConnection1->Connected = false;
}
```

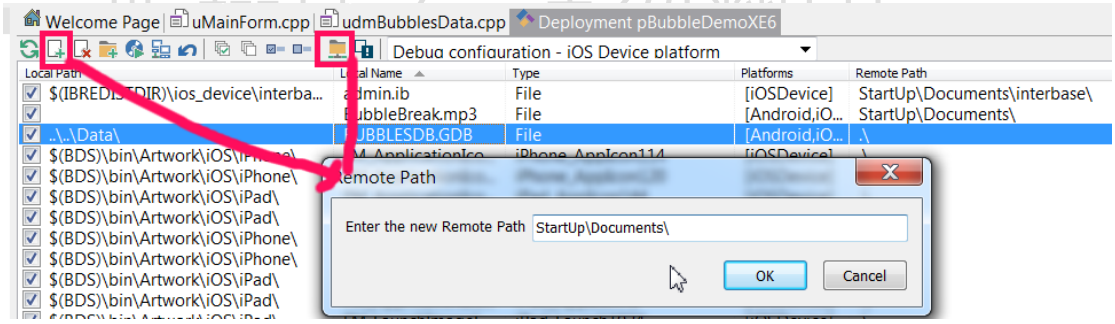
再為 TFDConnection 元件的 OnBeforeConnect 事件處理函式撰寫如下的程式碼，在 OnBeforeConnect 我們需要從 App 的沙盒 Documents 目錄中載入 BUBBLESDB.GDB 範例資料庫：

```
void __fastcall TDataModule1::FDConnection1BeforeConnect(TObject
*Sender)
{
  FDConnection1->Params->Values["Database"] =
System::Ioutils::TPath::GetDocumentsPath() + PathDelim
+"BUBBLESDB.GDB";
}
```

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 沙盒的 Documents 目錄中。請在 IDE 中先啟動部署精靈，先點選上方的 Add Featured Files 快捷鍵加入要在此 iOS App 中使用 InterBase ToGo 的功能，C++Builder 就會在部署 App 時也順便部署 InterBase ToGo 的相關函式庫：



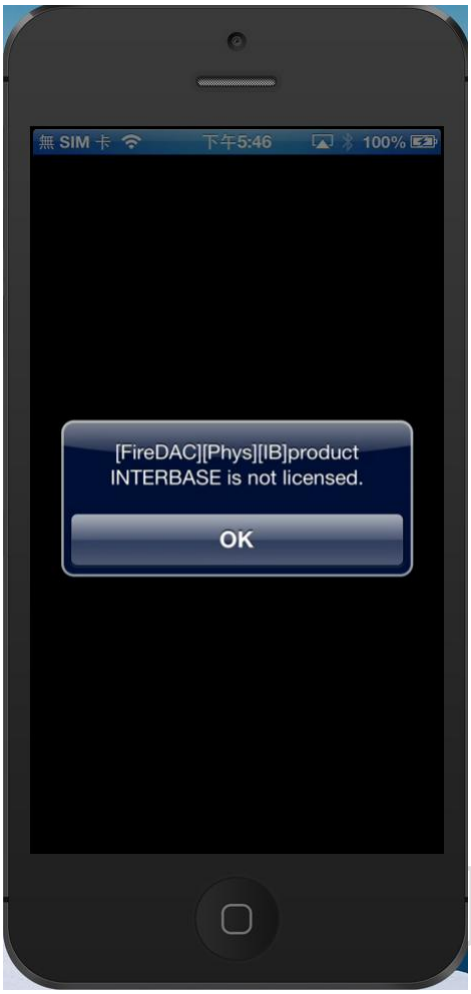
接著點選 Add Files 快捷鍵加入部署 BUBBLESDB.GDB 到 StartUp\Documents\目錄中：



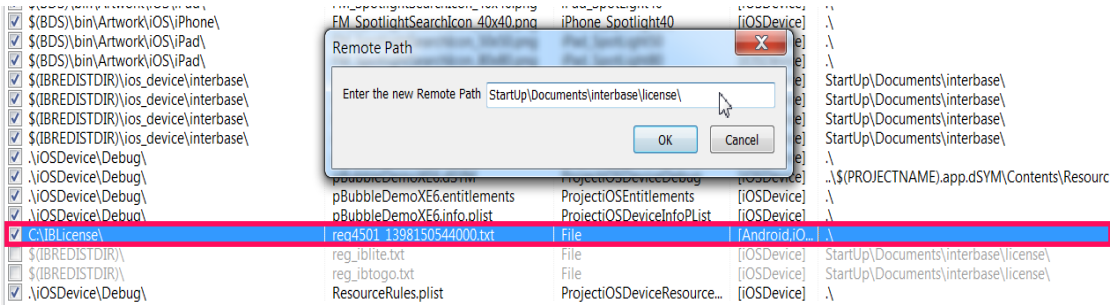
此外您需要到下面的ＵＲＬ取得 InterBase ToGo 的部署授權檔一起部署：

```
http://docwiki.embarcadero.com/RADStudio//en/IBLite_and_IBToGo_Test_Deployment_Licensing
```

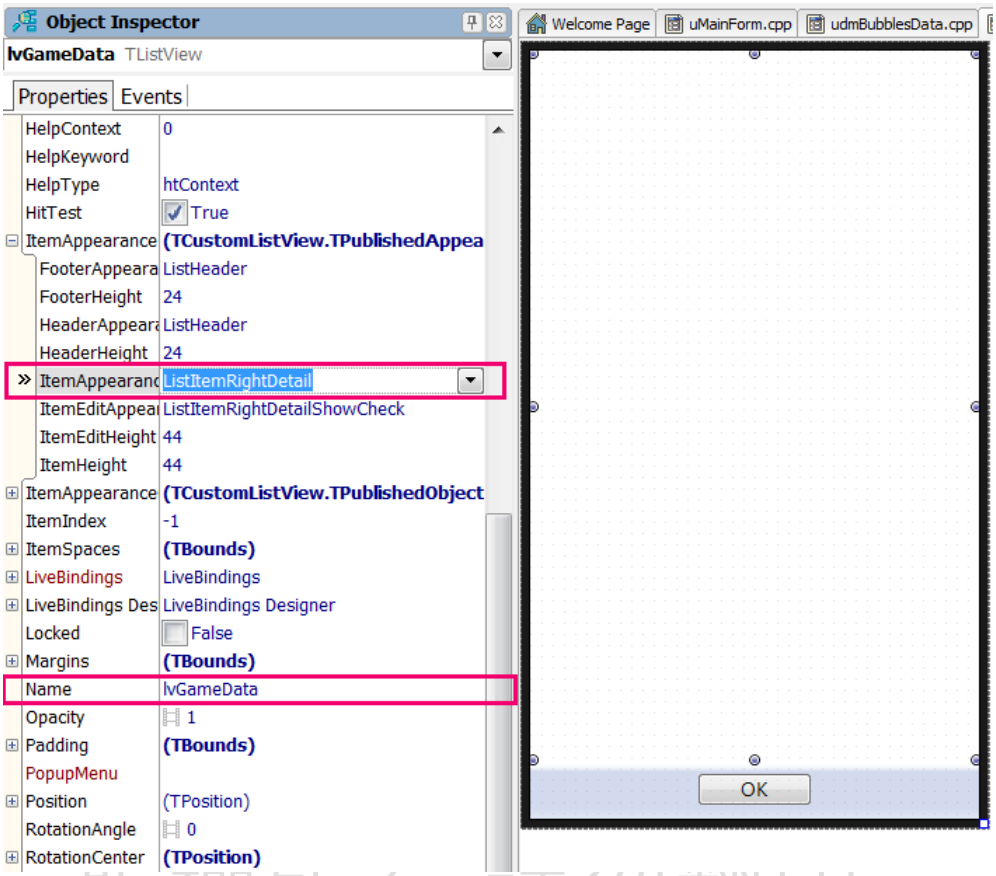
否則在此範例 App 執行時您便會看到下面的錯誤訊息：



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

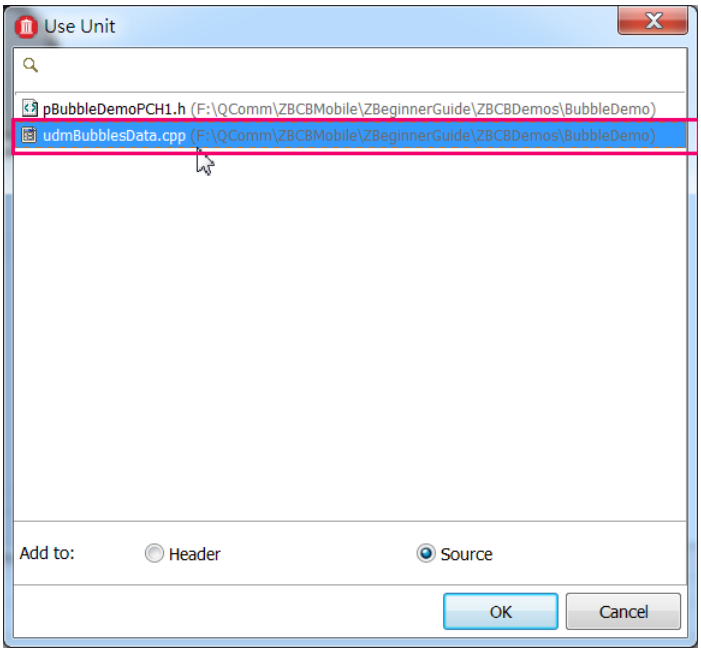


好了， 現在我們還需要一個ＵＩ來顯示資料庫中的資料，請於專案中再建立一個新的 Frame 物件設定它的 Name 特性值為 frmGameData，在其中放入 TListView，ToolBar 和 TButton 元件，設定 TListView 的 Align 特性值為 alClient，再設定它的 ItemAppearance 特性值為 ListItemRightDetail，設定它的 Name 特性值為 lvGameData，如下所示：



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：

```
void TfmMainForm::ResetGame()
{
  WriteGameInfo();
  WriteGameData(
  SetupGameInfoFrame();
  for (int iRow = 0; iRow < IROWS; iRow++)
  {
     for (int iCol = 0; iCol < ICOLS; iCol++)
     {
       if (BubblePoppedStatus[iRow][iCol])
       {

     pBubbles[iRow][iCol]->MultiResBitmap->Assign(imgOriNormalBubble->
MultiResBitmap);
         BubblePoppedStatus[iRow][iCol] = false;
       }
     }
  }
}
```

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掐破的泡泡數傳給它：

```
void TMainForm::WriteGameData()
{
  dmGameData->WriteGameData(Now(), GetBrokenBubles());
}
```

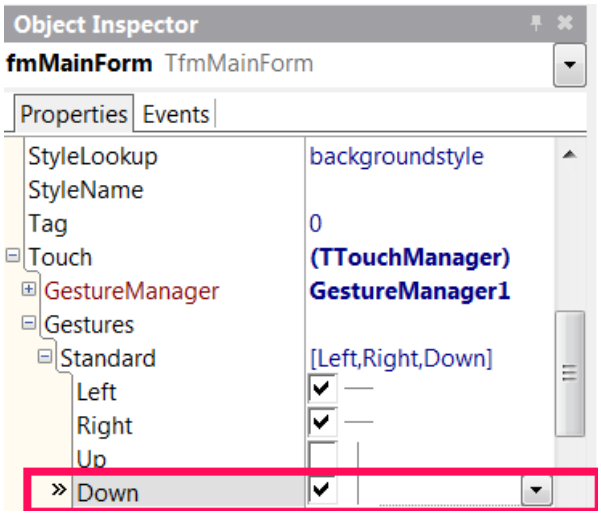
資料模組的 WriteGameData()方法非常簡單，它使用 TFDQuery 元件把資料新增到 TBLBUBBLESGAMEINFO 資料表中：

```
void TdmGameData::WriteGameData(const TDateTime dt, const int iBubbles)
{
    FDQuery1->Insert();
    FDQuery1->Fields->Fields[0]->Value = dt;
    FDQuery1->Fields->Fields[1]->Value = iBubbles;
    FDQuery1->Post();
    FDQuery1->Refresh();
}
```

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)
            {
              FillGameData();
              faniGameDataY->Enabled = false;
              faniGameDataY->StartValue = -iScreenHeight;
              faniGameDataY->StopValue = 0;
              faniGameDataY->Enabled = true;
              Handled = true;
            }
```

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：

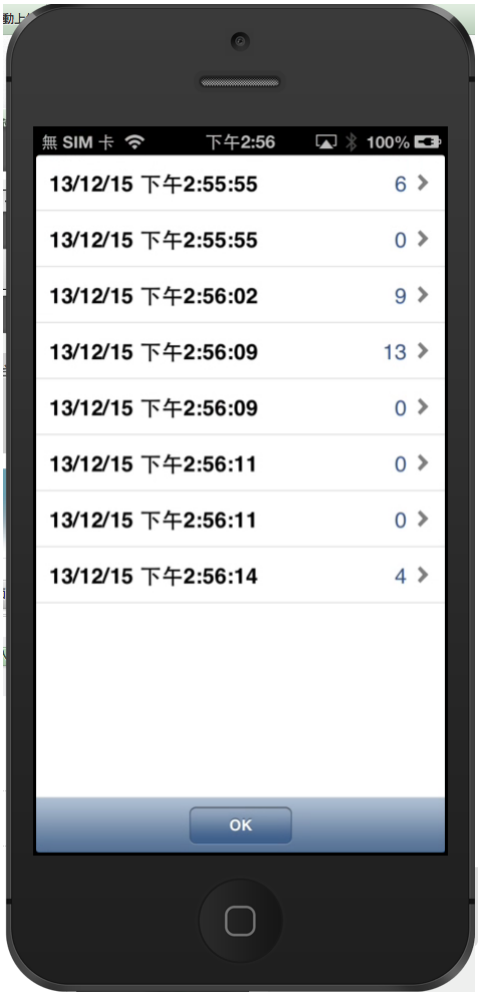
```
void TMainForm::SetupGameInfoFrame()
{
  frmGameInfo1->Position->X = Layout2->Width;
  frmGameInfo1->Position->Y = 0;
  frmGameInfo1->Parent = this;

  frmGameStatistics1->Position->X = -Layout2->Width;
  frmGameStatistics1->Position->Y = 0;
  frmGameStatistics1->Parent = this;

  frmGameData1->Position->X = 0;
  frmGameData1->Position->Y = -iScreenHeight;
  frmGameData1->Parent = this;
}
```

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：

```
void TMainForm::FillGameData()
{
  frmGameData1->lvGameData->Items->Clear();
  frmGameData1->lvGameData->Items->BeginUpdate();
  try
  {
```

```
    dmGameData->FDQuery1->First();
    while (!dmGameData->FDQuery1->Eof)
    {
      TListViewItem *plvi = frmGameData1->lvGameData->Items->Add();
      plvi->Text = dmGameData->FDQuery1->Fields->Fields[0]->Value;
      plvi->Detail = dmGameData->FDQuery1->Fields->Fields[1]->Value;
      dmGameData->FDQuery1->Next();
    }
  }
  __finally
  {
      frmGameData1->lvGameData->Items->EndUpdate();
  }
}
```

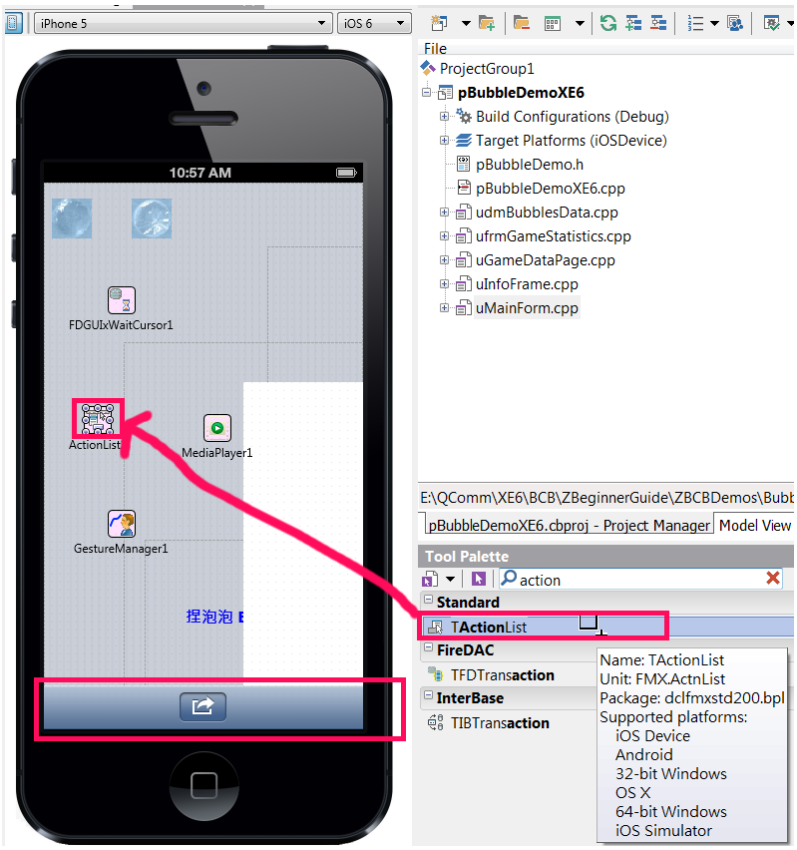
現在請編譯範例 App 點選泡泡再搖動手機儲存並重新開始遊戲,如此反覆數次之後再使用手指從手機上方往下方滑動，就可以看到類似如下的畫面，範例 App 的遊戲資訊果然成功的儲存到 InterBase ToGo 資料庫並且能夠顯示出來了：



8-7 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 C++Builder 中的 ShareSheet 功能就可以輕易的完成，而 ShareSheet 功能則內含在 TActionList 元件。

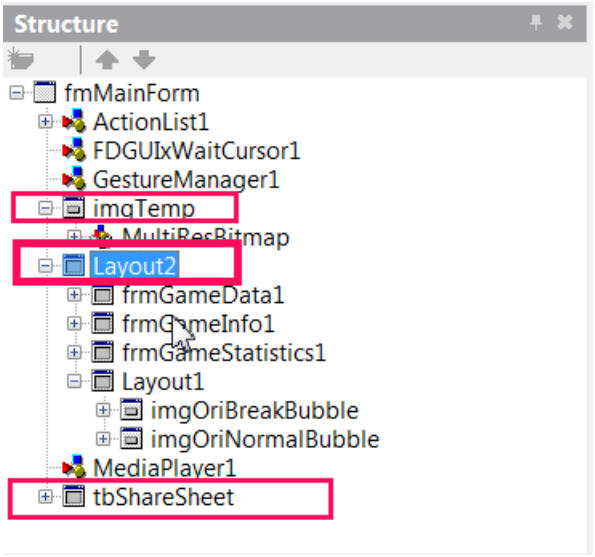
首先在主表單中加入一個 ToolBar 元件設定它的 Align 特性值為 alBottom，再於其中放入一個 TButton 元件設定它的 StyleLookup 特性值為 actiontoolbuttonbordered 以及 Name 特性值為 btnShareGameInfo，再放入一個 TActionList 元件，此時主表單如下所示：



接著繼續在主表單中加入一個 TLayout 元件，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

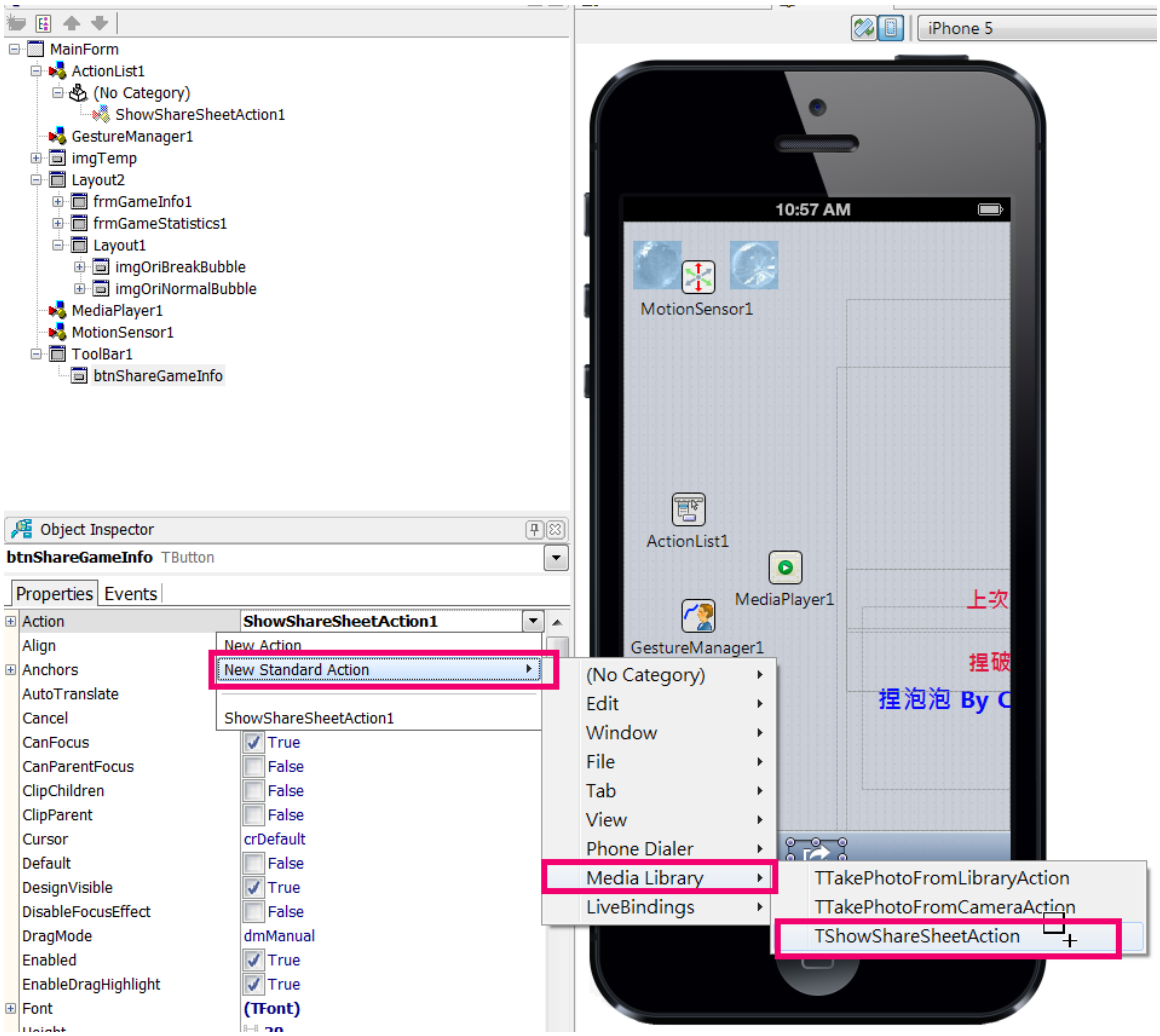
最後再於 IDE 左上方的架構視窗中移動前面 3 個 Frame 物件以及 Layout1 物件於新的 Layout2 物件之中，此時架構視窗會顯示主表單中所有元件的關係如下所示：



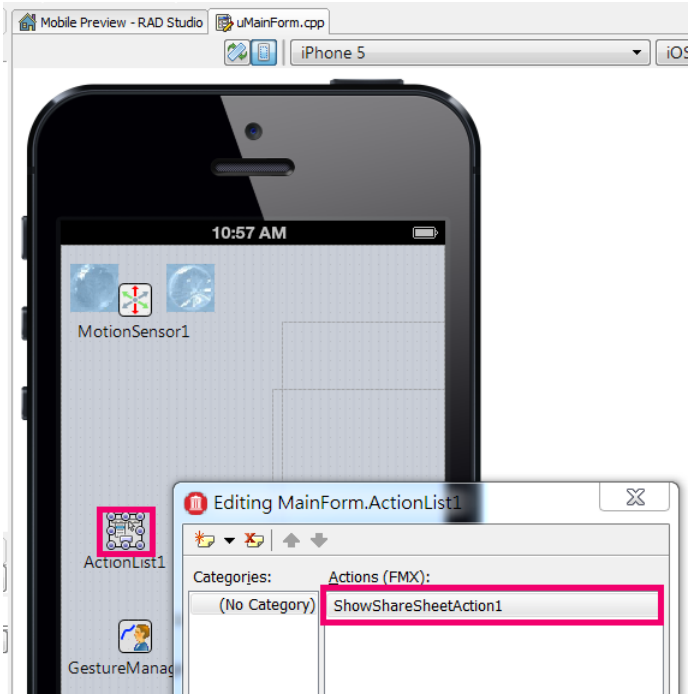
把前面 3 個 Frame 物件以及 Layout1 物件移動到新的 Layout2 物件之中是因為稍後我們要把手機遊戲畫面截取下來並且以圖形的方式分享出去，您很快會看到如何做到。

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。

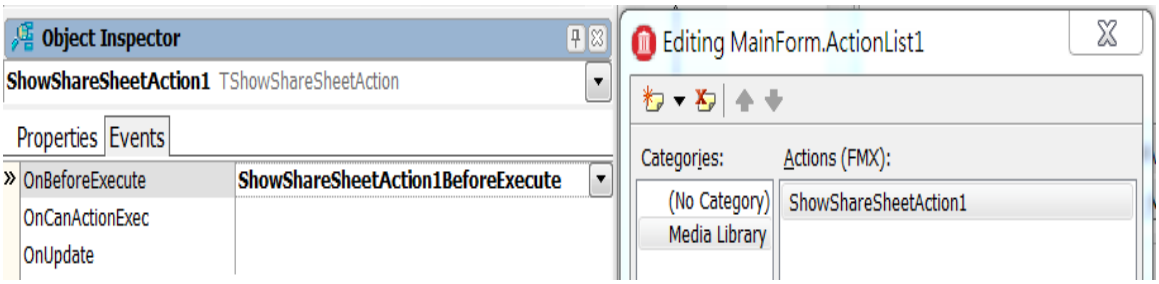
點選了 btnShareGameInfo 之後在物件檢視器中於它的 Action 特性中點選 New Standard Action | Media Library | TShowShareSheetAction 選項讓 btnShareGameInfo 的被點選時就觸發 TShowShareSheetAction 的動作，也就是分享的動作，如下所示：



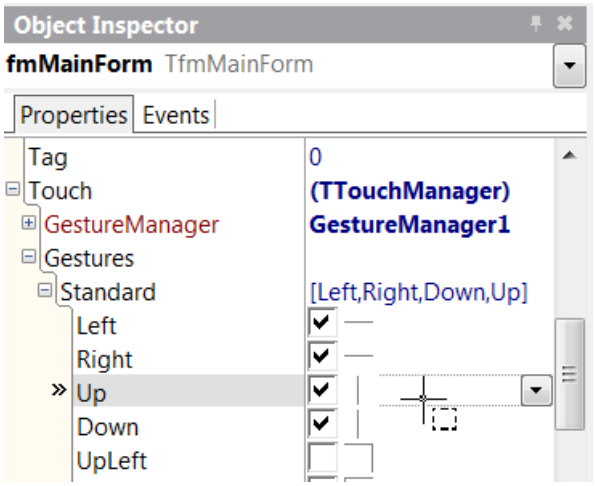
再雙擊主表單中的 TActionList 元件，此時在其中就會自動產生一個 TShowShareSheetAction 物件，如下所示。



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援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。在下面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)
          {
            GetScreenImage();
            SetupToolBarPosition(true);
          }
```

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：

```
void TMainForm::GetScreenImage()
{
  imgTemp->Bitmap->Assign(Layout2->MakeScreenshot());
}
```

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    void TMainForm::SetupToolBarPosition(const bool bVisible)
002    {
003      if (bVisible)
```

```
004     {
005     ToolBar1->Align = TAlignLayout::alBottom;
006     ToolBar1->BringToFront();
007     }
008     else
009     {
010     ToolBar1->Align = TAlignLayout::alNone;
011     ToolBar1->Position->Y = Layout1->Height;
012     }
013     ToolBar1->Visible = bVisible;
014   }
```

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 提昇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。


```
void TfmMainForm::SetupGameInfoFrame()
{
  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;
}
```

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。

```
void __fastcall TMainForm::ShowShareSheetAction1BeforeExecute(TObject
*Sender)
{
    ShowShareSheetAction1->TextMessage = "這次捏破了" +
IntToStr(GetBrokenBubles()) + "泡泡";
    ShowShareSheetAction1->Bitmap->Assign(imgTemp->Bitmap);
    SetupToolBarPosition(false);
}
```

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：

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：

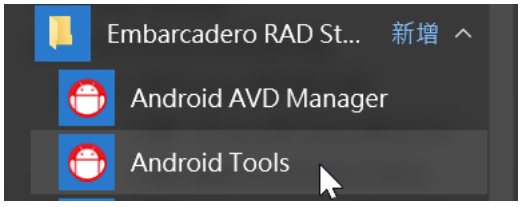
最近如果點選上面畫面中的發佈按鈕之後就可以真的在 Facebook 的網頁中看到如下的執行結果畫面，我們成功的使用 C++Builder 的 ShareSheet 功能分享範例 App 的遊戲資訊了，Cool！

C++Builder for Android入門指引手冊

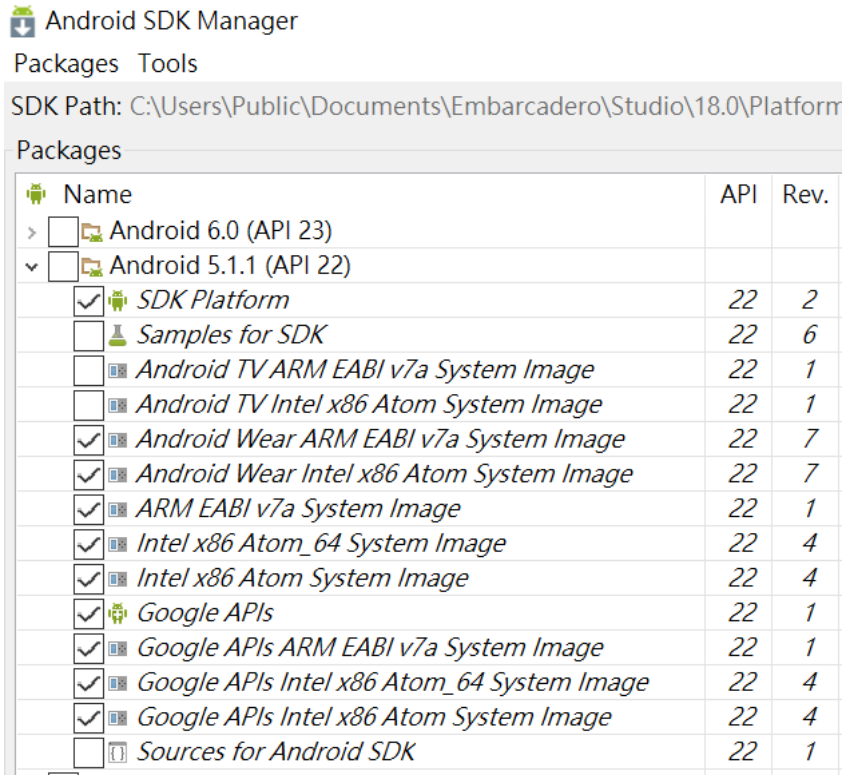
強調的功能就是一套原始程式碼就可同時開發多個平台的應用程式，因此在前面已經說明了如何使用開發 iOS App，那麼您現在就可以使用相同的技巧來開發 Android App。不過開發 Android 和開發 iOS 不同的地方是在開發 Android 之前您需要先安裝好 Android SDK 和 Android NDK，並且在 C++Builder IDE 中設定好開發環境。

9.安裝和設定 BCB for Android 開發環境

由於 Google 授權政策的改變因此安裝時不再會同時安裝 Android SDK 和 Android NDK，讀者可在安裝完後在它的程式群組中執行 Android Tools 程式安裝 Android SDK 和 Android NDK:



例如筆者使用 Android Tools 程式安裝了 Android 5.1.1:



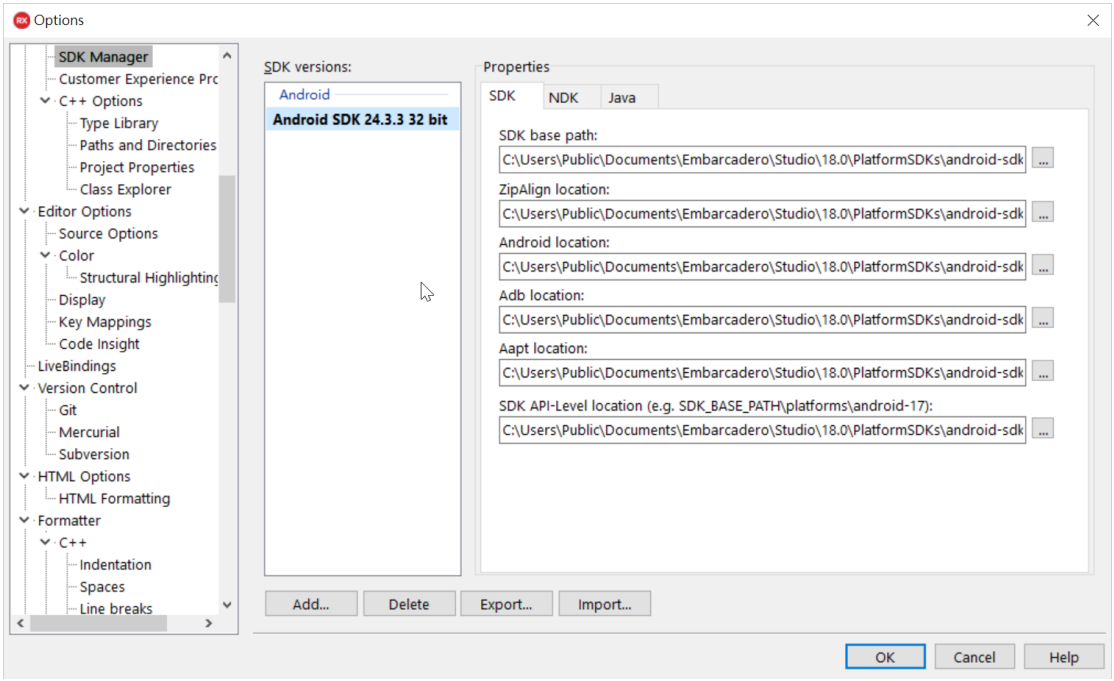
完成安裝 Android SDK 和 Android NDK 後需要 C++Builder IDE 中設定 Android SDK 和 Android NDK 的安裝路徑，以便讓 IDE 能夠找到相關的檔案和工具。例如 Android Tools 把 Android SDK 安裝在：

```
c:\Users\Public\Documents\Embarcadero\Studio\18.0\PlatformSDKs\android-sdk-windows\
```

Android NDK 安裝在：

```
c:\Users\Public\Documents\Embarcadero\Studio\18.0\PlatformSDKs\android-ndk-r9c\
```

接著請回到的 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：

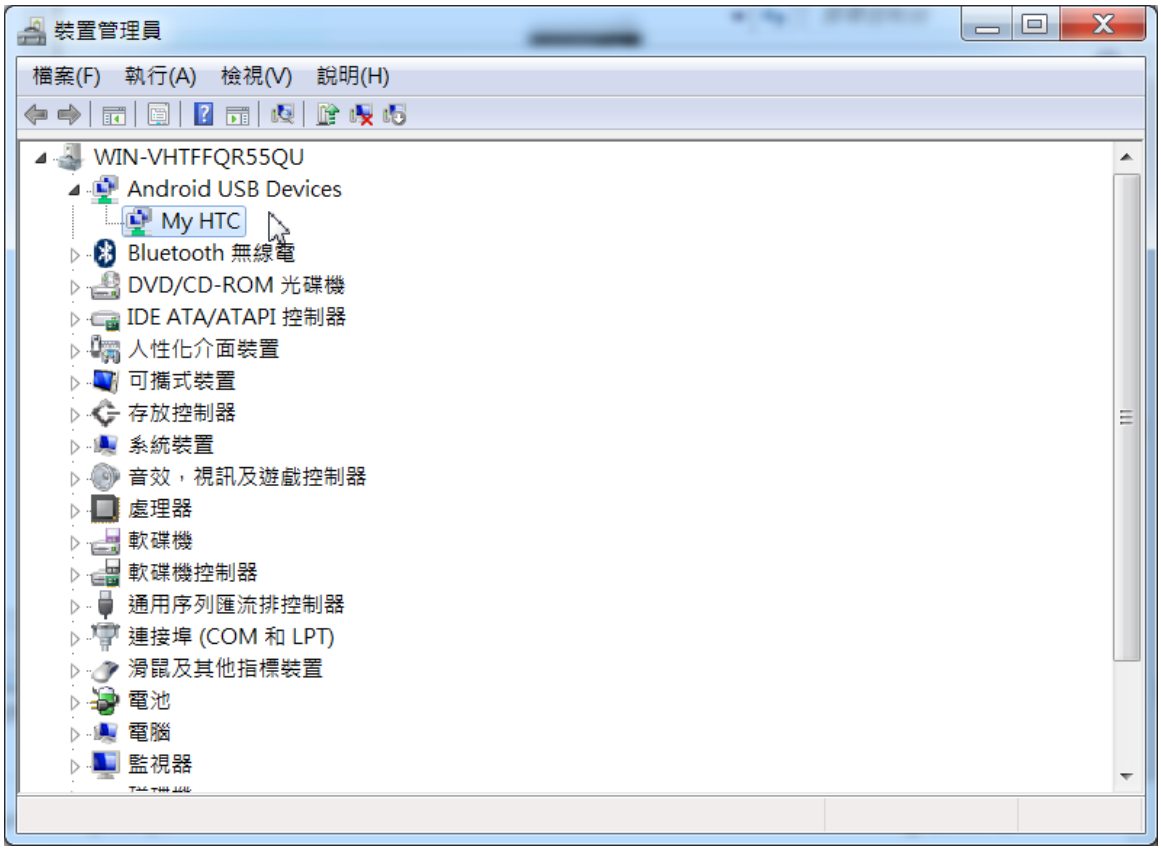


我們只需要把上 SDK 工具設定成 Android Tools 安裝的路徑即可。

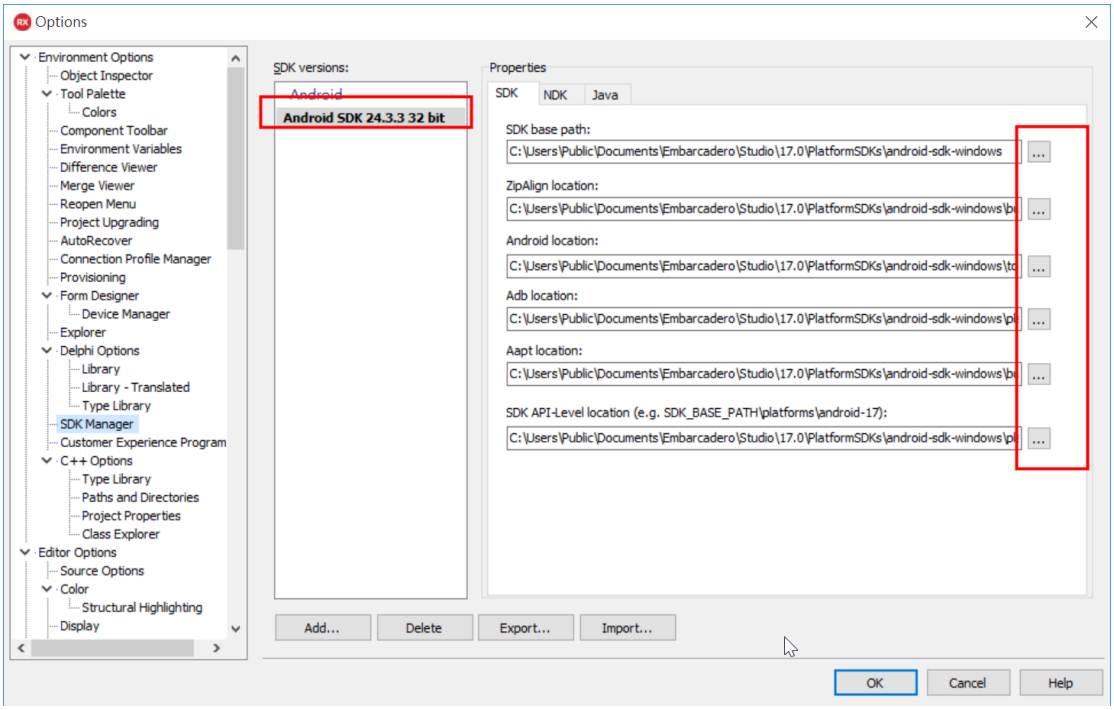
之後讀者需要安裝使用開發的 Android 手機的驅動程式，才能讓 C++Builder IDE 部署和測試。讀者需要到您的手機廠商網站下載，例如筆者是使用 hTC Incredible S，因此筆者到 hTC 下列的 URL 下載驅動程式：

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

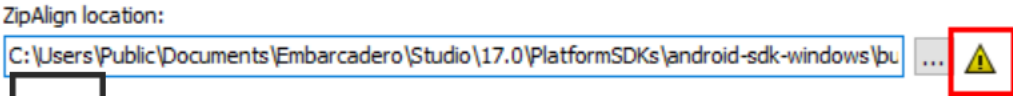
一旦安裝了 hTC 的 Sync Manager 之後筆者在作業系統的裝置管理員中就可以看到 HTC 手機：



在 XE7 之後到的版本中，當您安裝完 C++Builder 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\android-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。

10.開發 C++Builder for Android App

從之後 C++Builder 正式支援 Android 的開發，而且 iOS 的程式碼可以再使用於 Android 平台，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 平台中執行。

在本小節中我們將示範如何使用 C++Builder 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，我們將使用台北市政府提供的公共資訊做為範例。

台北市政府在下面的 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 中台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 C++Builder 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
- 使用部份關鍵字搜尋旅館

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

10-1 開發查詢臺北市旅館 App

在下面的 URL 中：

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

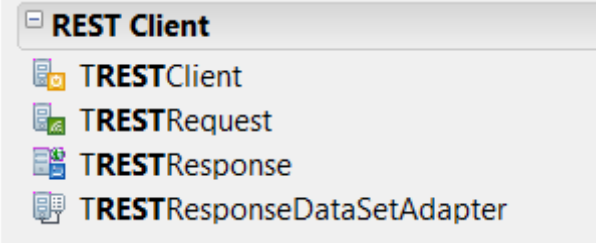
提供的旅館資訊是使用 JSON 封裝的資料，下面是取得結果的部份資料：

```
 [{"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。

瞭解了這個台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 C++Builder 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 客戶端以使從任何提供 REST 公共服務的來源取得需要的資料。

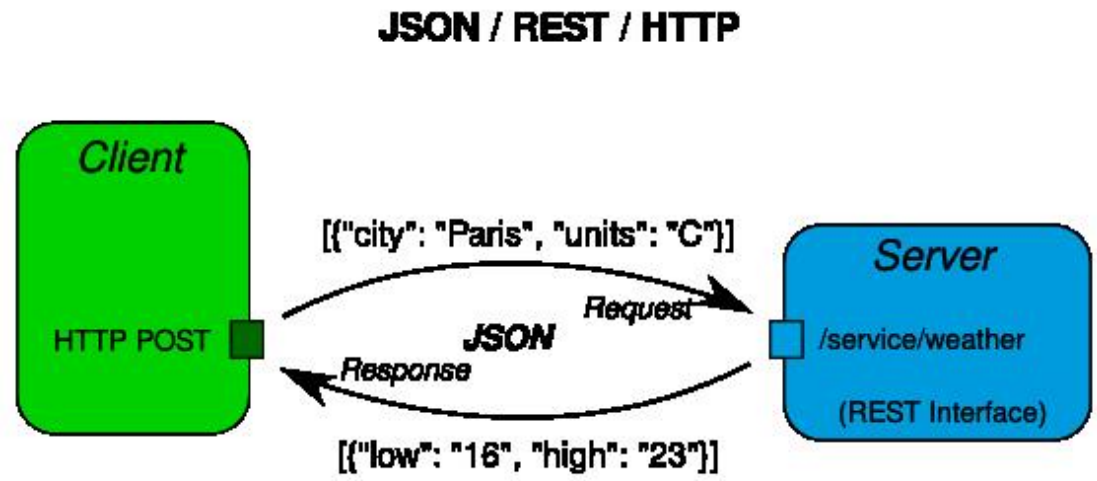


下面的表格說明了上面 4 個 REST Client 元件組的功能：

元件	說明
TRESTClient	指定提供 REST 服務的 REST 服務器
TRESTRequest	提出 REST 請求向 REST 服務器要求服務
TRESTResponse	REST 服務器執行結果回傳到此元件
TRESTResponseDataSetAdapter	把 TRESTResponse 元件中的 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。在下圖說明了一個 RESTful 架構的基本執行流程。左方的 REST 客戶端藉由 HTTP/HTTPS 向 REST 服務器要求服務，這個要求的服務也是使用 JSON 封裝的，在 REST 服務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 客戶端。

因此我們可以使用上表中的 TRESTClient 元件指定右方 REST 服務器的所在地，使用 TRESTRequest 提出要求服務，右方 REST 服務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。

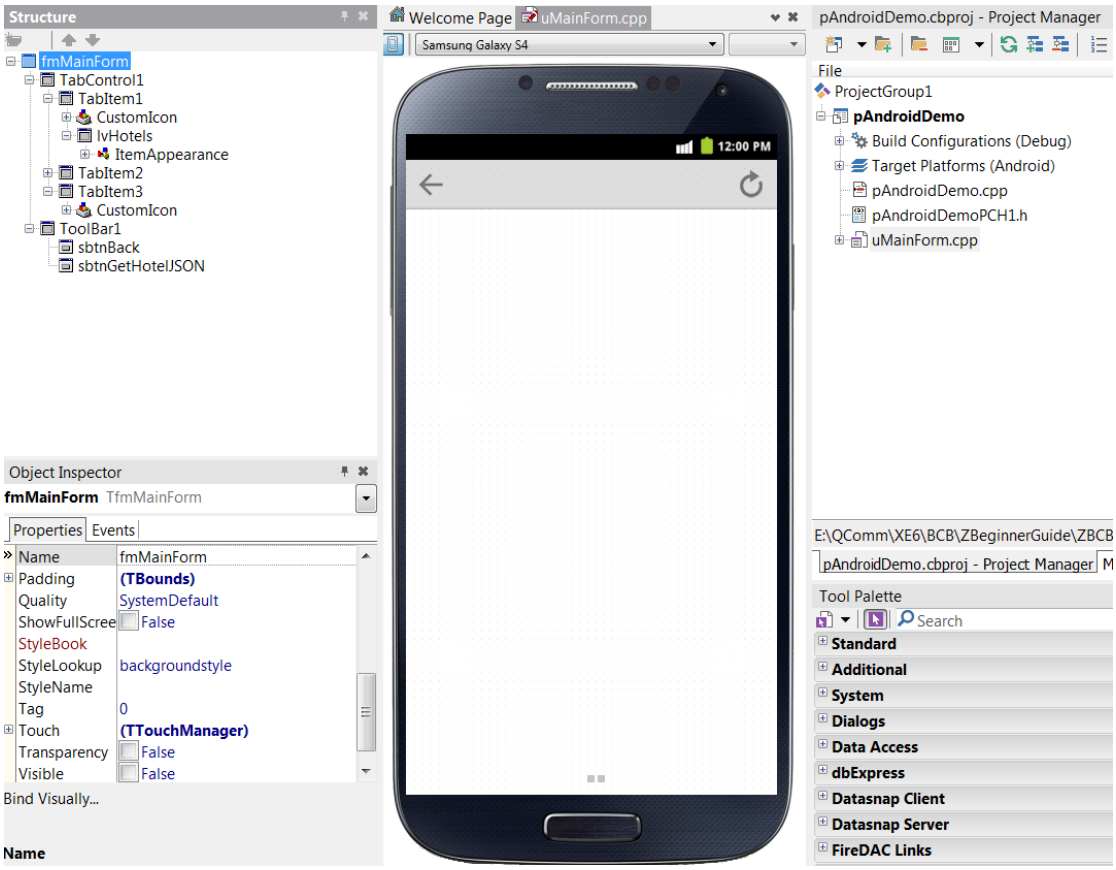


瞭解了上面的概念之後我們就可以輕易的使用 REST Client 中的元件完成查詢台北市旅館資訊的工作：

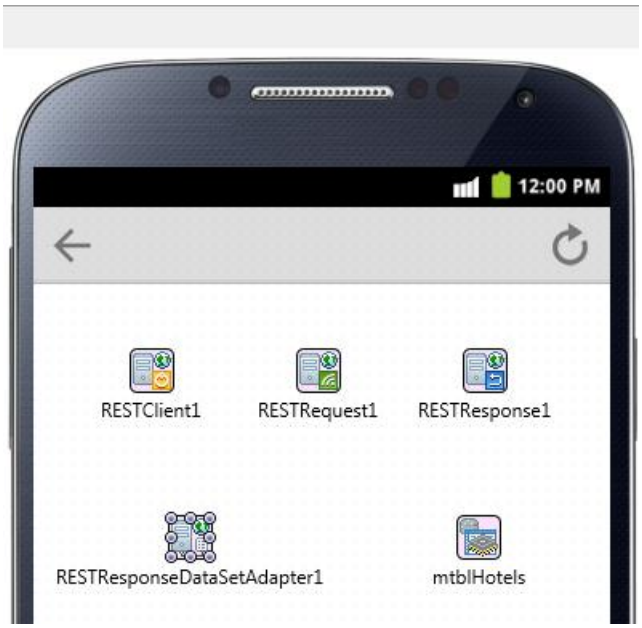
1. 使用 TRESTClient 元件設定指向 http://data.taipei.gov.tw/opendata/apply/NewDataContent?oid=4418C600-D7F5-46D6-BB1F-1DA29DAB91E9 取得服務
2. 使用 TRESTRequest 元件提出 REST 請求
3. REST 請求結果會自動回傳到 TRESTResponse 元件
4. 再使用 TRESTResponseDataSetAdapter 元件把 TRESTResponse 元件中的 REST 執行結果換成資料集

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

現在請使用 C++Builder 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 元件並且設定此 TListView 元件的 ItemAppreance | ItemAppeance 子特性值為 ListItemRightDetail。最後再放入 ToolBar 元件並且在其中放入 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 元件：

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

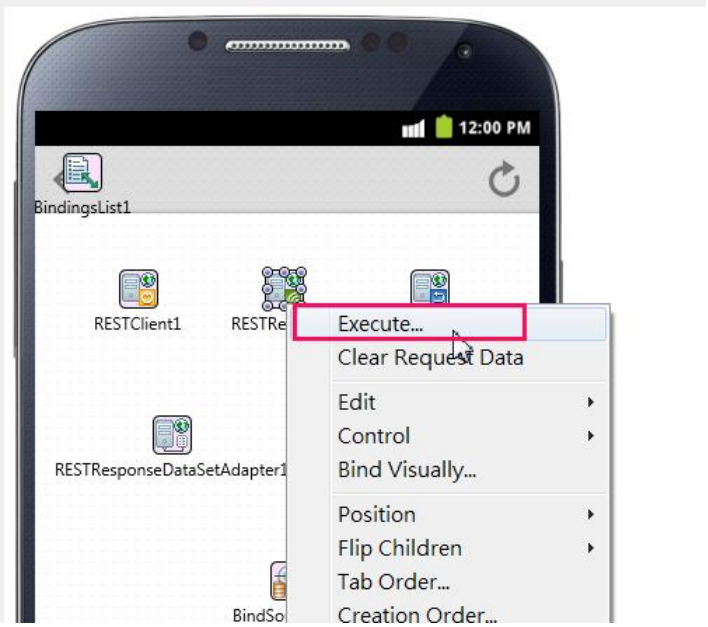
TFDMemTable 元件：

特性	特性值
Name	mtblHotels

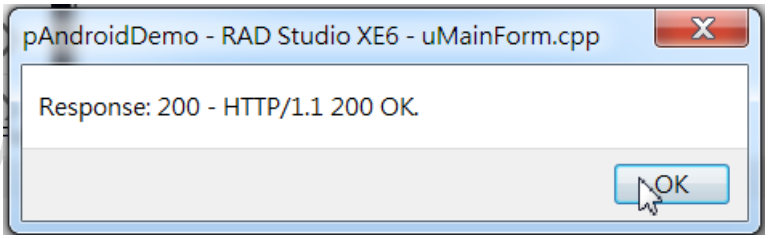
RESTResponseDataSetAdapter1 元件：

特性	特性值
ResponseJSON	RESTResponse1
DataSet	mtblHotels

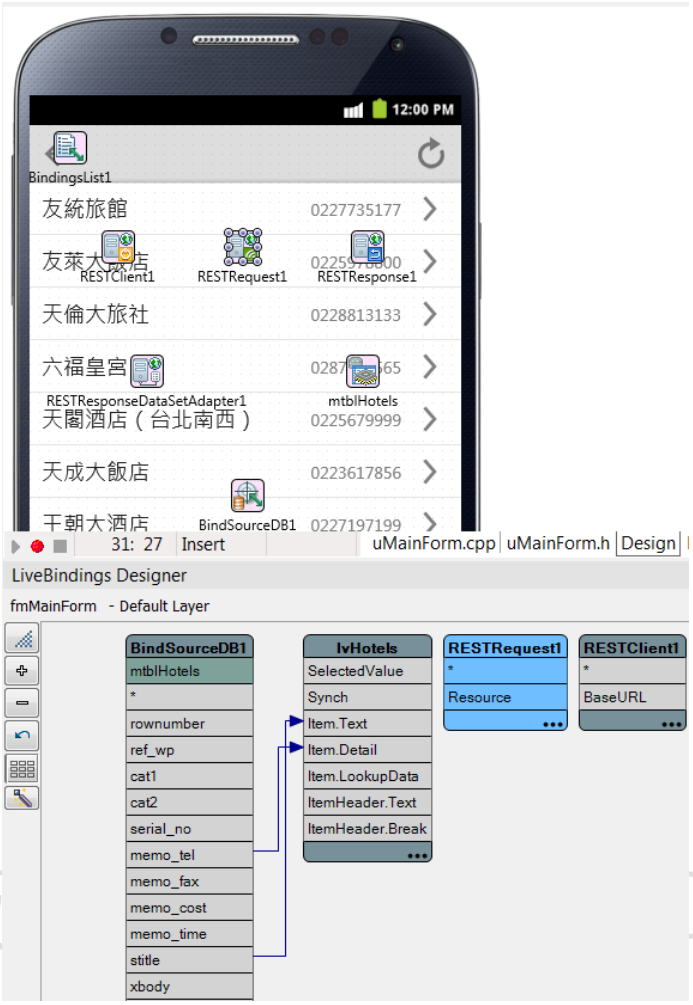
接著我們就可以開始設計台北市旅館顯示在 lvHotels 之中了，為了方便設計 UI，現在請點選主表單中的 RESTReuqest1 元件，點選滑鼠右鍵選擇 Execute...選項(請確定您的網路連結在工作中)：



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 元件中：



接著設定 RESTResponseDataSetAdapter1 元件的 Active 特性值為 true，開啟 LiveBinding 設計家，把 mtblHotels 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 元件中看到台北市旅館的資訊了：



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：

```
void __fastcall TfmMainForm::FormActivate(TObject *Sender)
{
  RESTResponseDataSetAdapter1->Active = false;
}
```

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：

```
void __fastcall TfmMainForm::sbtnGetHotelJSONClick(TObject *Sender)
{
  GetTaipeiHotelsJSON();
}
```

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 以顯示最及時的台北市旅館的資訊：

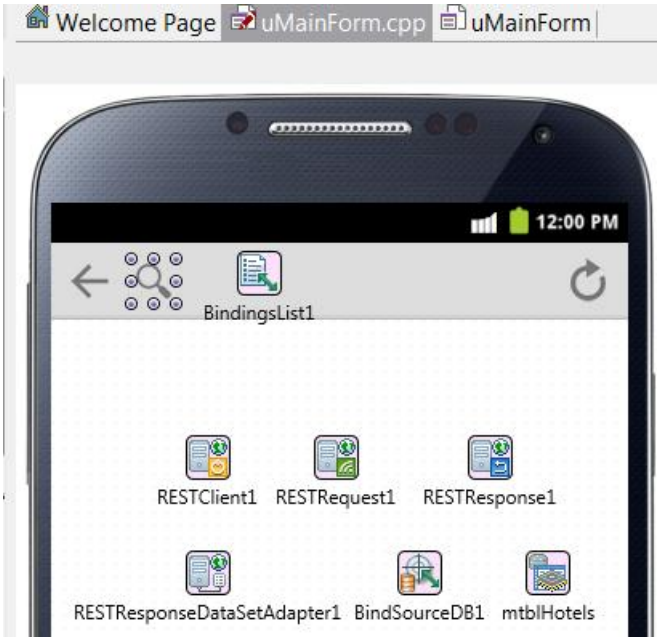
```
void TfmMainForm::GetTaipeiHotelsJSON()
{
  RESTRequest1->Execute();
  RESTResponseDataSetAdapter1->Active = true;
}
```

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台北市旅館的資訊，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：

10-2 加入查詢旅館功能

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，讓我們試著加入查詢的功能，讓這個範例 App 能夠允許使用者藉由寫出旅館的部份名稱後就可以自動幫助使用者查詢旅館。

回到主表單並且在 ToolBar 中放入一個新的 TSpeedButton 做為搜尋旅館功能：

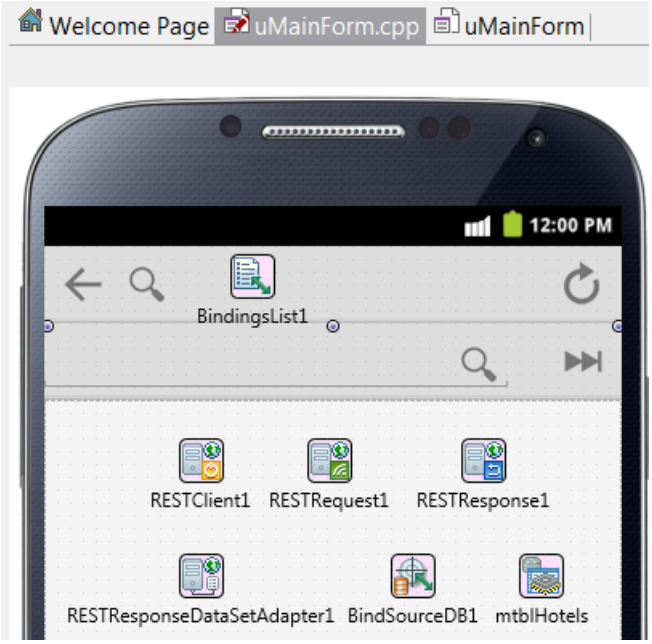


在這個 TSpeedButton 的 OnClick 事件處理函式中撰寫如下的程式碼：

```
void __fastcall TfmMainForm::sbtnSpeechSearchClick(TObject *Sender)
{
  if (!mtblHotels->Active)
    GetTaipeiHotelsJSON();
  TabControl1->ActiveTab = TabItem2;
  edtHotelName->SetFocus();
}
```

如果此時尚未取得所有旅館資料就先呼叫 GetTaipeiHotelsJSON()方法，接著顯示 TabControl 的第 2 個 TabItem 頁面。

在主表單的 TabControl 的第 2 個 TabItem 中再放入一個 ToolBar，一個 TEdit，一個 TSpeedButton 和一個 TWebBrowser 元件：



在 TEdit 元件的右方的搜尋 TSpeedButton 的 OnClick2 事件處理函式中撰寫如下的程式碼：

```
void __fastcall TfmMainForm::SearchEditButton1Click(TObject *Sender)
{
  if (edtHotelName->Text != "")
    SearchHotel(edtHotelName->Text);
}
```

SearchEditButton1Click 先判斷使用者是否有輸入任何旅館名稱的字元，如果有的話就呼叫 SearchHotel()方法在 mtblHotels 中搜尋旅館資訊。

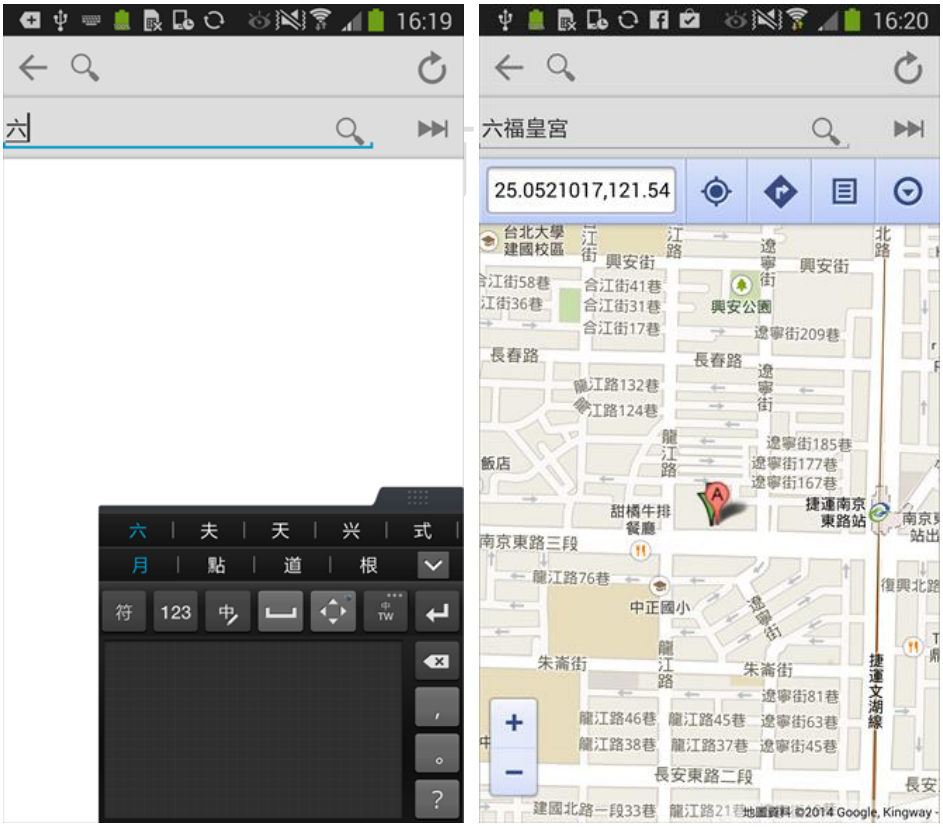
SearchHotel()方法使用 TFDMemTable 元件的 LocateEx()方法在 mtblHotels 中搜尋資料，一旦找到了使用者輸入的旅館名稱之後就呼叫 DisplaySearchedHotel()方法顯示目標旅館：

```
void TfmMainForm::SearchHotel(const String sData)
{
  Firedac::Comp::Dataset::TFDDataSetLocateOptions Opts;
  Opts << lxoCaseInsensitive;
  Opts << lxoPartialKey;

  if (mtblHotels->LocateEx("stitle", sData, Opts) )
    DisplaySearchedHotel();
}
```

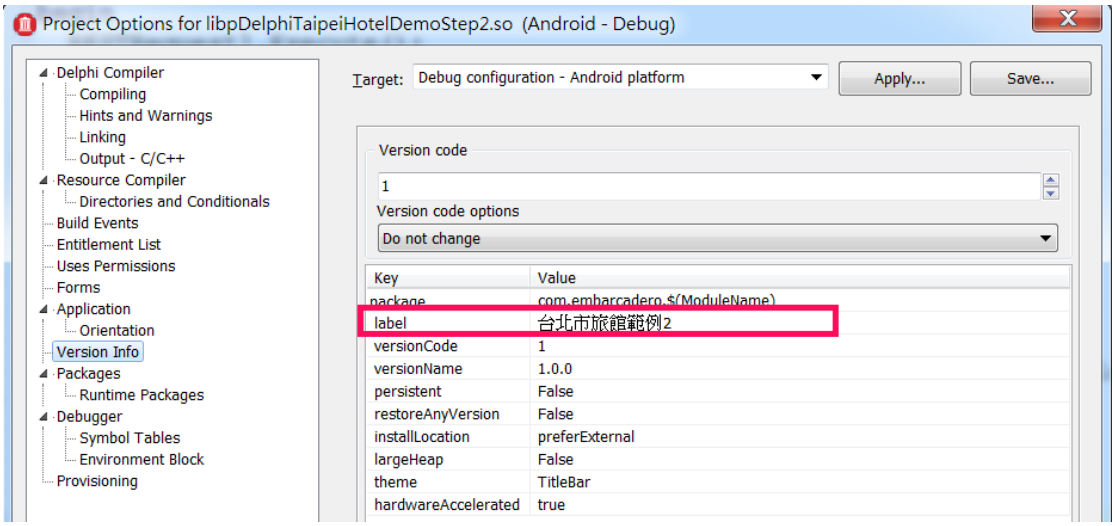
DisplaySearchedHotel()方法藉由從 mtblHotels 中取出搜尋旅館的經/緯度資訊再使用 TWebBrowser 元件顯示目標旅館的所在地：

```
void TfmMainForm::DisplaySearchedHotel()
{
  edtHotelName->Text = mtblHotels->FieldByName("stitle")->Value;
  String sHotelLongitude = mtblHotels->FieldByName("longitude")->Value;
  String sHotelLatitude = mtblHotels->FieldByName("latitude")->Value;
  String URLString = Format("https://maps.google.com/maps?q=%s,%s",
ARRAYOFCONST((sHotelLatitude, sHotelLongitude)));
  WebBrowser1->Navigate(URLString);
  TabControl1->ActiveTab = TabItem2;
}
```

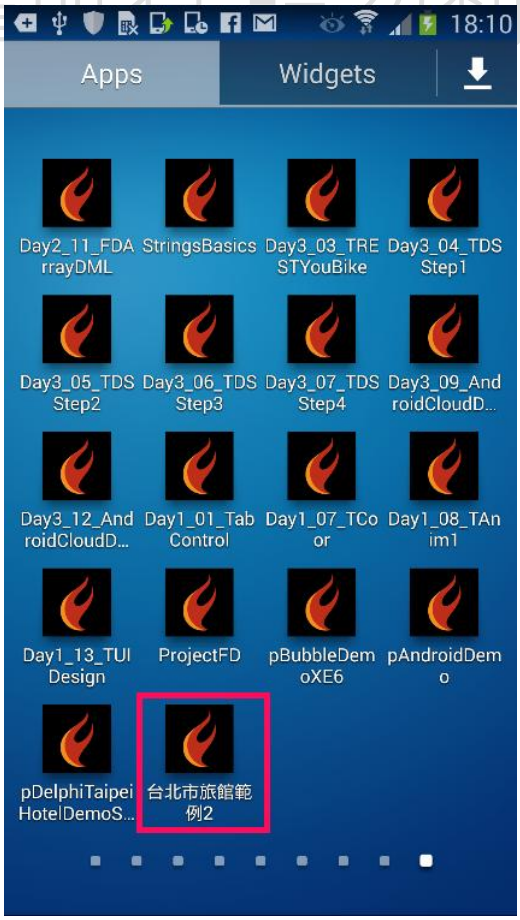
下圖是筆者在 S4 上搜尋六福皇宮的畫面，讀者可以看到筆者只輸入了”六”之後點選旁邊的搜尋按鈕之後就可以正確的找到”六福皇宮”了：



到現在為止當我們部署範例 App 到手機中時看到的範例 App 名稱都是英文的專案名稱，現在讓我們為它設定中文的部署名稱吧。請在專案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"台北市旅館範例 2":

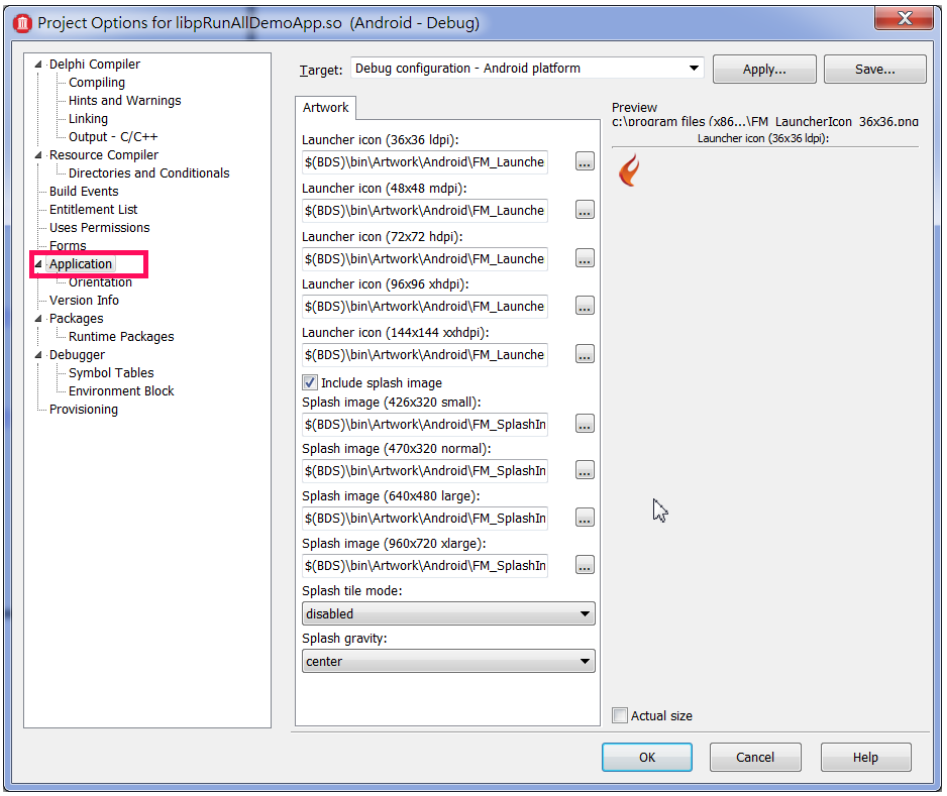


現在您可再次部署此範例 App 到手機中，您應該就可以看到中文的 App 名稱了，例如下圖就是此範例 App 部署到 S4 手機中的結果畫面：

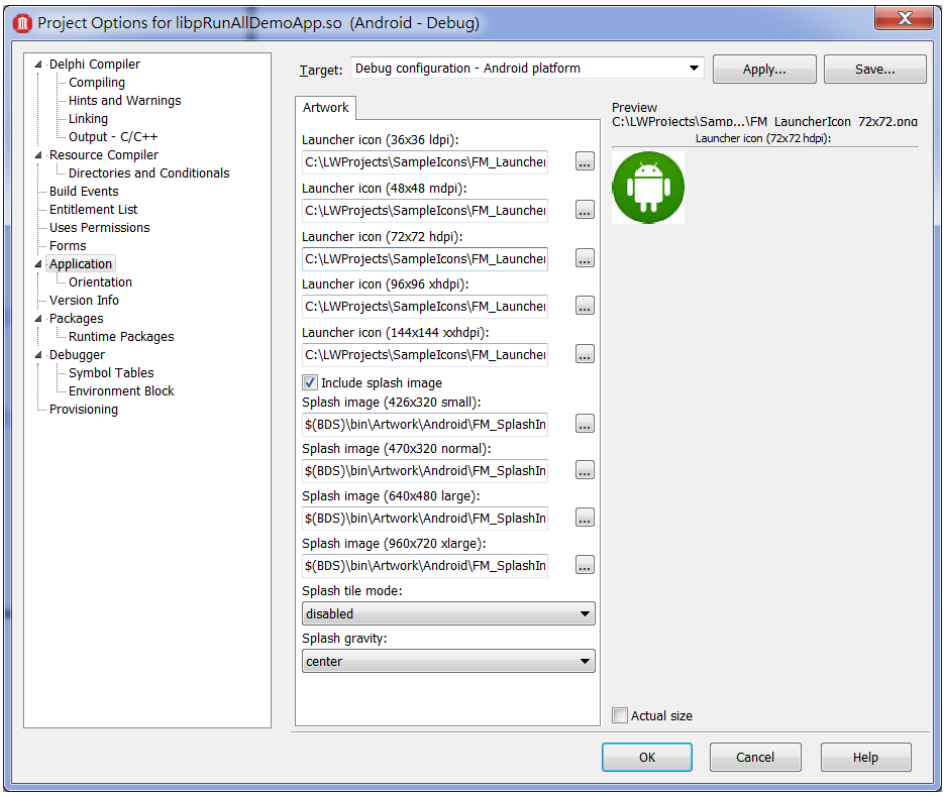


10-3 為您的 App 設計個圖像吧

當您使用 C++Builder 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製化的圖像來代表您的 App 而不是使用 C++Builder 內定的圖像。在中要為 App 更改客製化圖像非常的簡單，請點選 Project | Options 選單，在顯示的對話盒 Application 項目中可以看到 C++Builder 內定使用的各種大小的 App 圖像，如下所示：



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製化圖像。例如下圖就是筆者在此對話盒 Application 項目中載入客製化圖像：

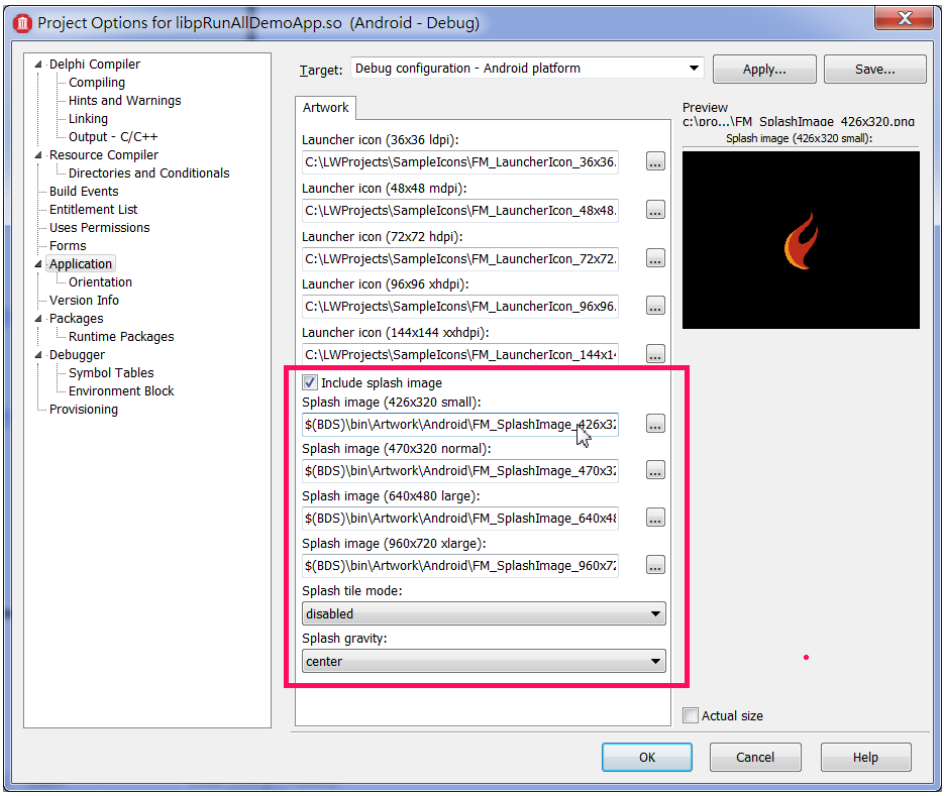


接著重新編譯並且部署您的 App，您就可以在手機中看到您的 App 現在使用了您自己的客製化圖像：

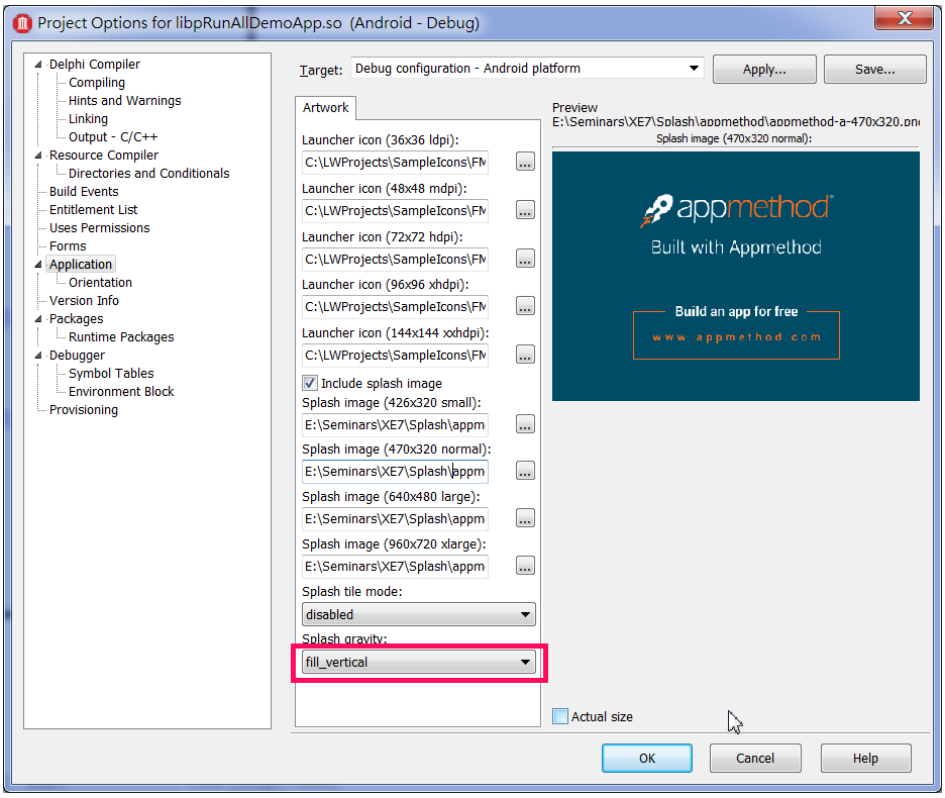
10-4 為您的 App 加入啟動畫面吧

除了 App 圖像之外，也允許開發人員為 App 加入啟動時的歡迎畫面，由於編譯出的 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，在剛才加入客製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製化影像即可，例如下圖就是載入客製化影像並且設定使用 fill_vertical 方式顯示：



部署此 App 並且執行就可以看到如下的啟動畫面：

10-5 為您的 App 加入 Provisioning 資訊吧

當您開發完並且決定部署您的 App 並且如上加入了客製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。

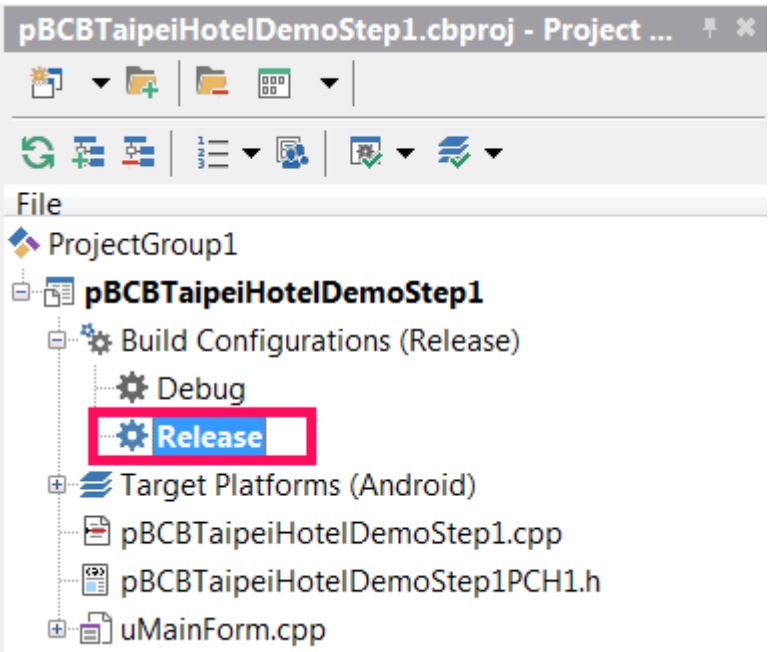
要對 App 進行 部署開通服務，您需要完成下列的工作：

1. 設定 Build Configurations 為 Release
2. 開啟 Target Platforms 設定為 Android 平台並在 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。

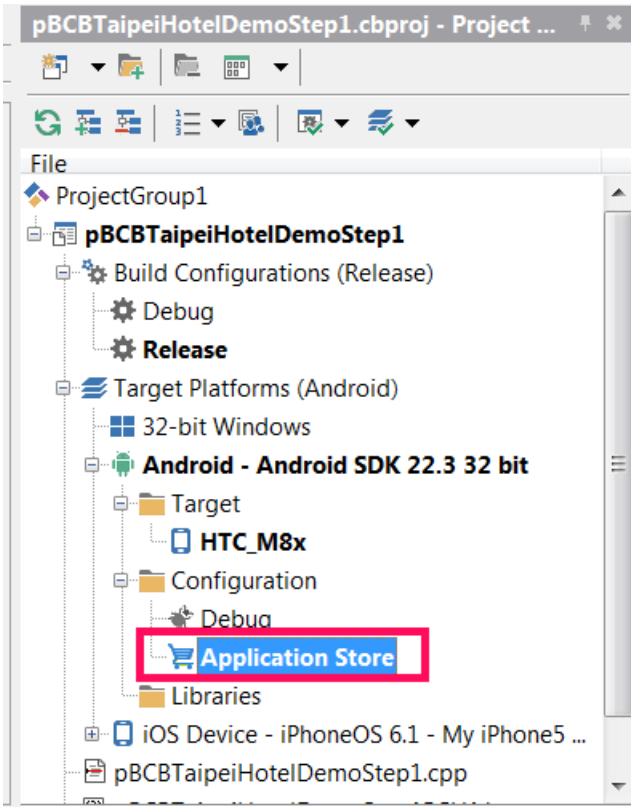
設定 Build Configurations 為 Release

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，請在專案管理員中雙擊 Build Configurations 中的 Release 節點：



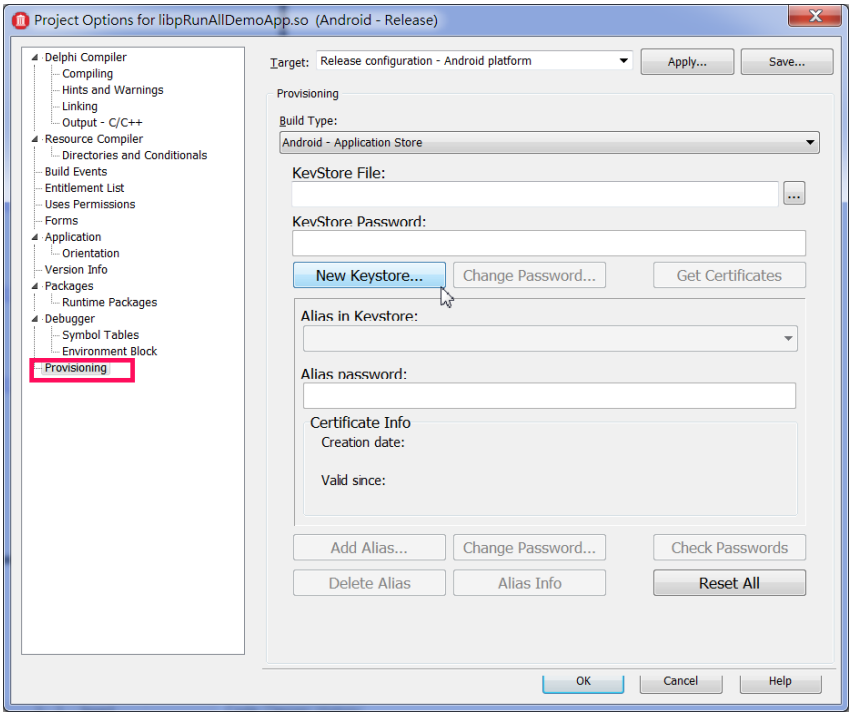
開啟 Target Platforms 設定為 Android 平台並在 Configuration 中選擇 Application Store

接著同樣在請在專案管理員中雙擊 Target Platforms | Configuration 中的 Application Store 節點:

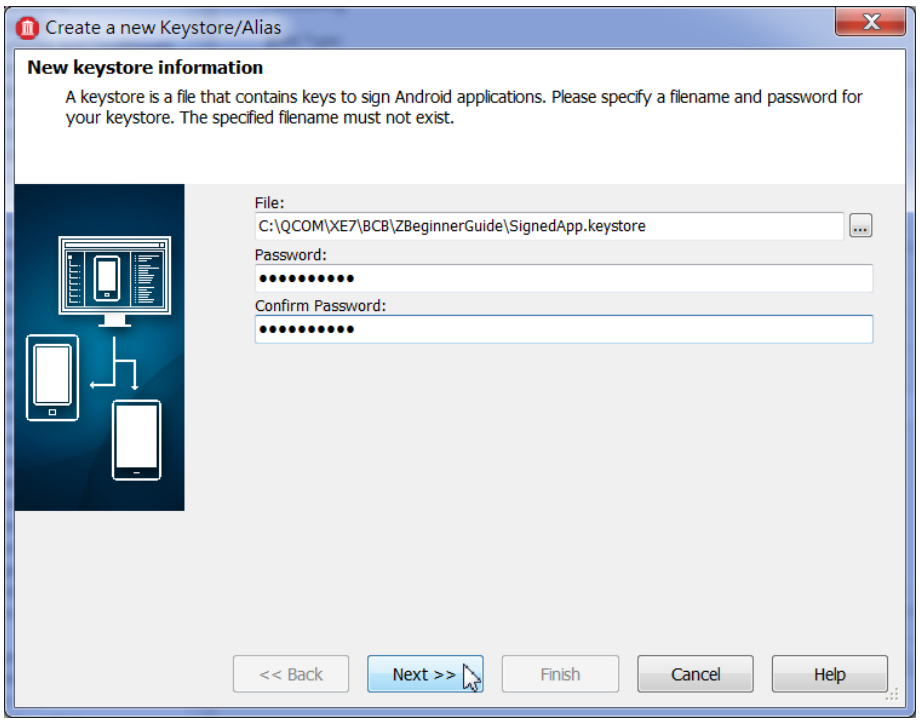


到 Project > Options > Provisioning 頁面填寫必要的資訊

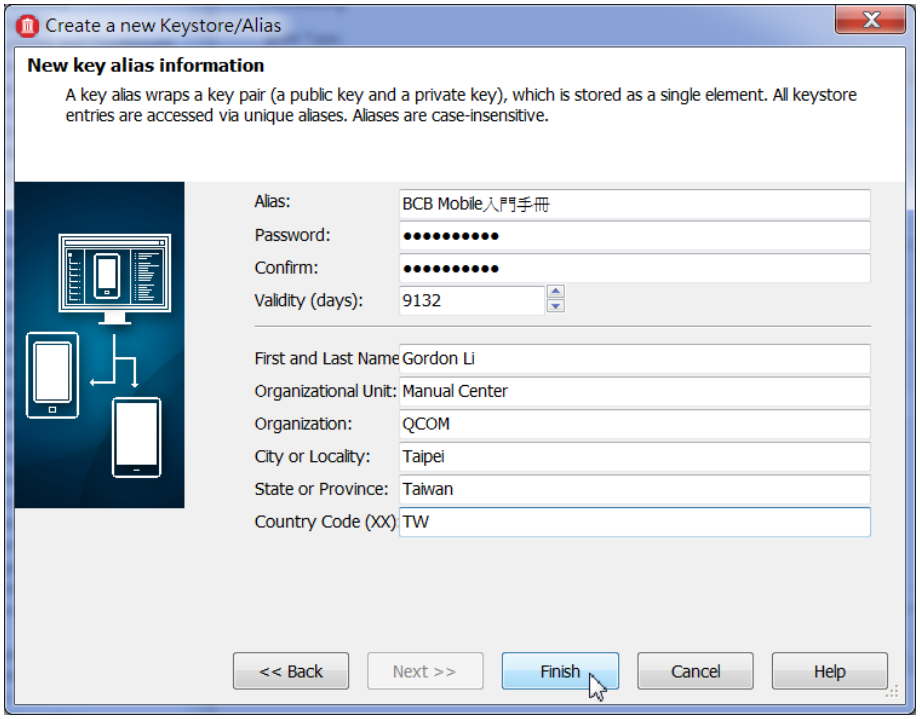
點選 Project | Options 選單，再點選 Provisioning 節點：



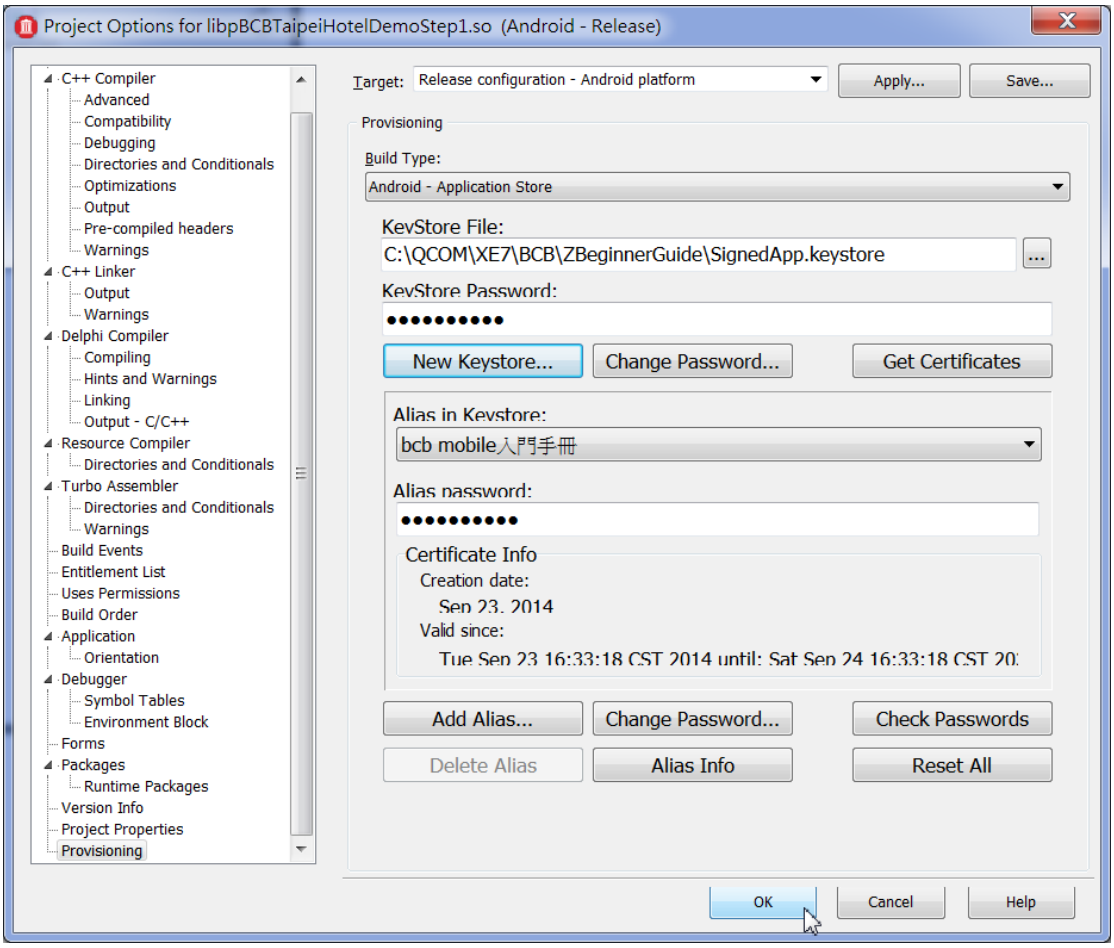
接著點選其中的 New Keystore...按鈕，選擇一個要建立 Keystore 檔的目錄和檔名並且輸入密碼：



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：

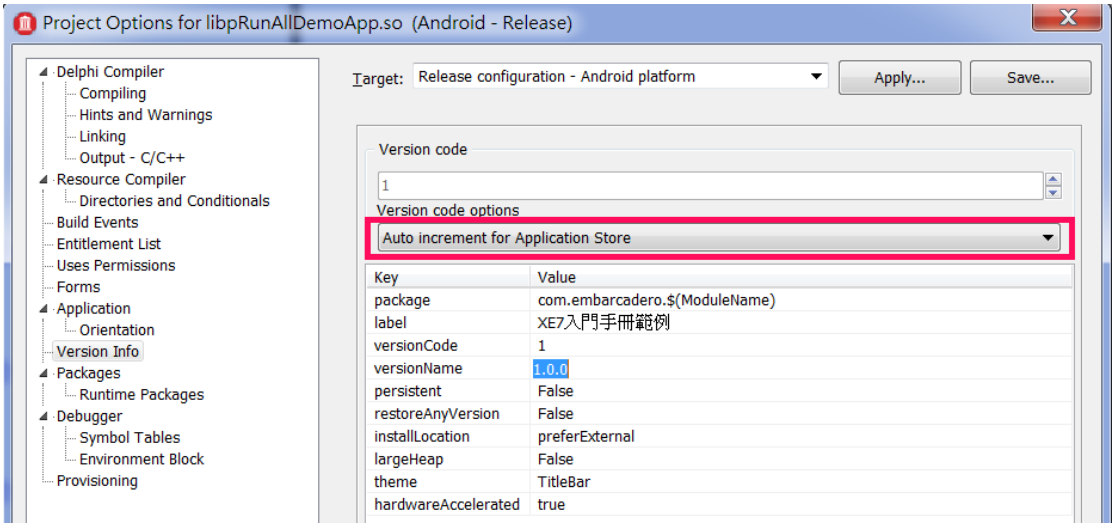


點選 Finish 按鈕之後就可以回到 Provisioning 節點並且看到剛才設定的資訊都已經自動帶入到 Provisioning 的各個欄位中了。剛才建立的 Keystore 檔是這個 App 的 Provisioning 資訊，請像 App 的原始程式一樣妥善保存好。



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 您的專案 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：

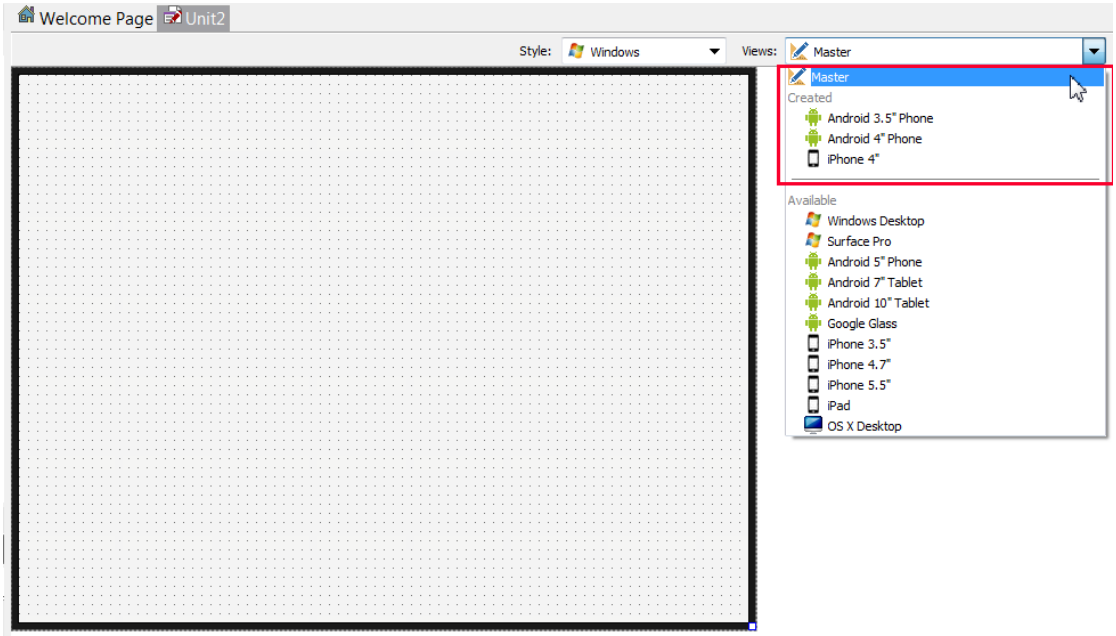
11 新功能

C++Builder 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 C++Builder 支援多平台的開發能力，但如何在多個平台開發時能夠減少 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。

11-1 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台開發的時間進而提昇生產力。在中 MultiView 繼續獲得了強化，加入了 Multi-Device 功能允許開發人員可同時檢視所有開發平台的 UI 介面。

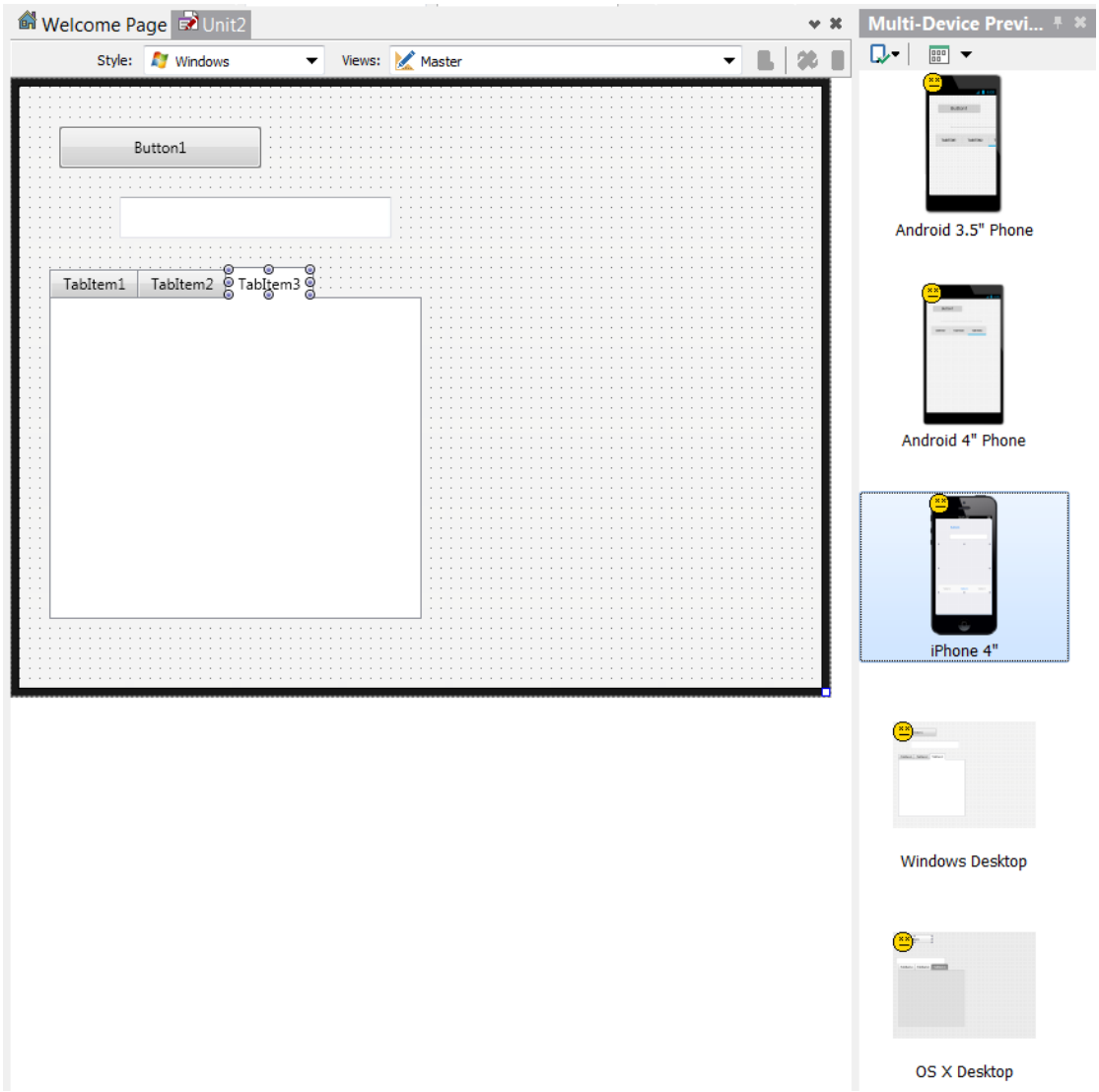
Multi-Device 在使用上非常的簡單，只要在開發專案時開啟 Multi-Device 設計視窗即可。例如下圖是一個 Multi-Device 專案，它同時要開發 Android 3.5 吋，Android 4 吋和 iPhone 吋的設備：



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。而且只要使用滑鼠雙擊 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。

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。



11-2 使用分散式版本控制工具-Git

C++Builder 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，到了對於 Git 的支援就比較完整了。本小節的內容將說明如何在 C++Builder IDE 中使用 Git 進行版本控制功能。

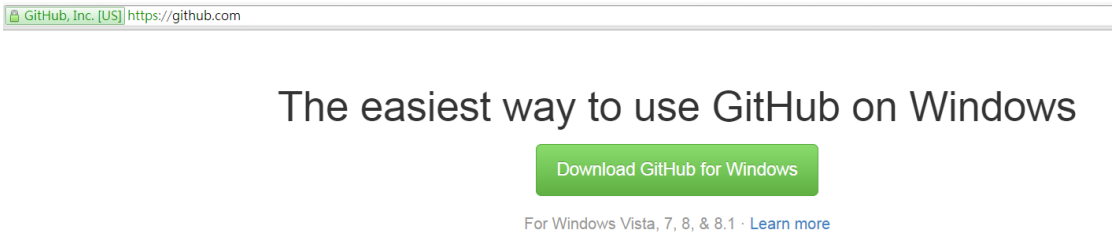
要在 IDE 中使用 Git，開發人員需要進行下面的步驟：

1. 下載和安裝 Git

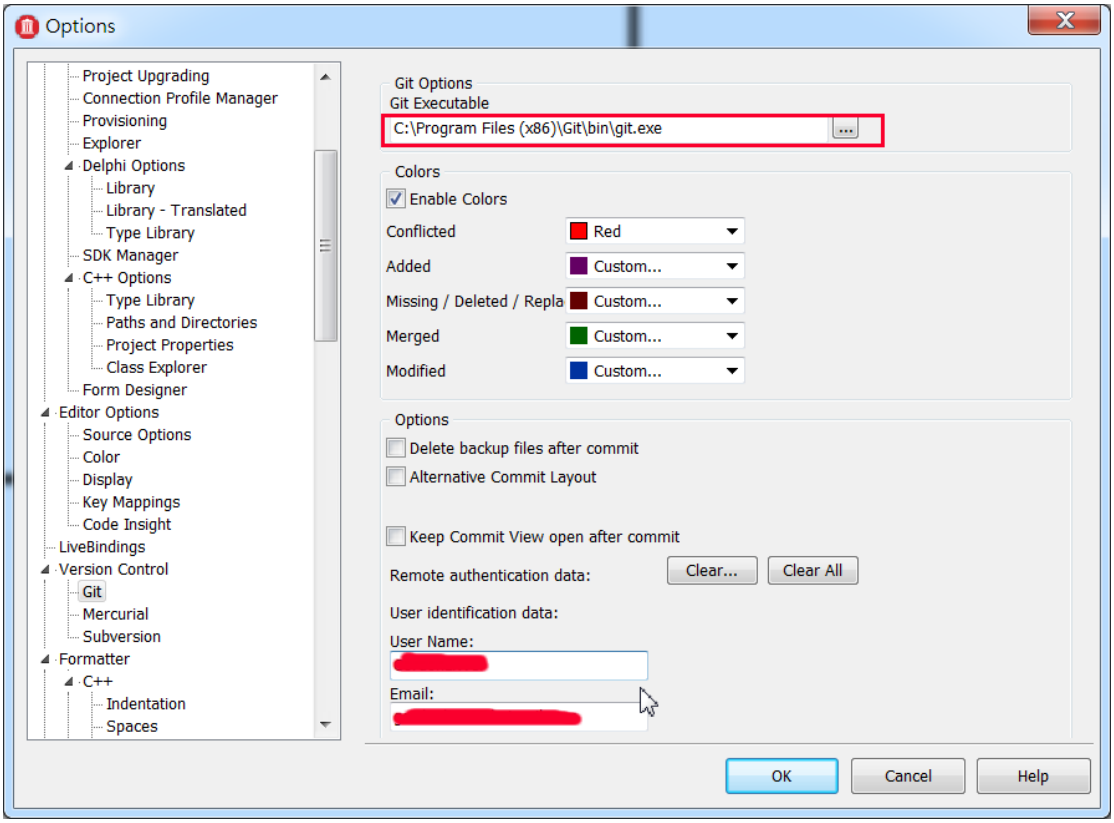
2. 在 IDE 中設定 Git

3. 申請帳號

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



下載 Git 之後請安裝它，安裝完之後請到 C++Builder 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

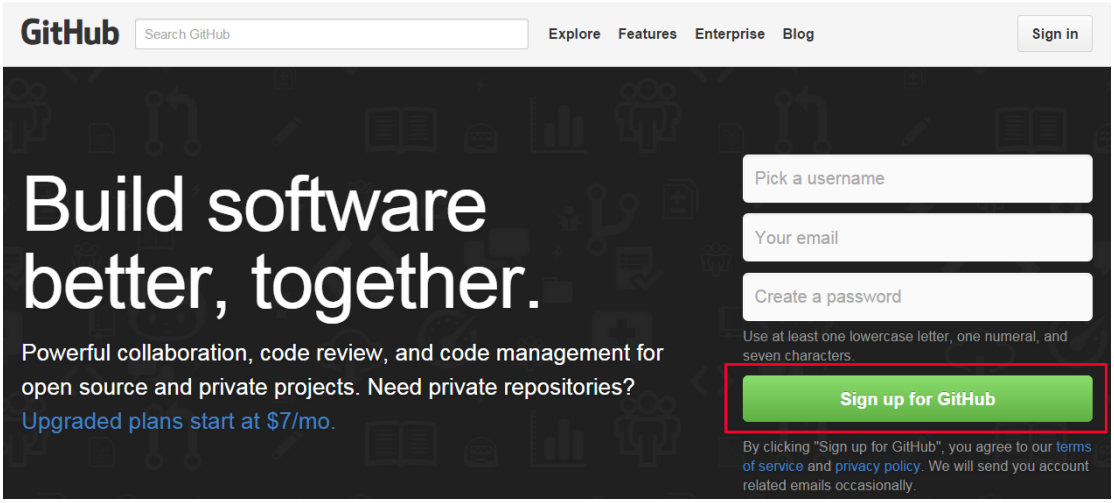
```
https://github.com/
```

申請使用 GitHub，或是到

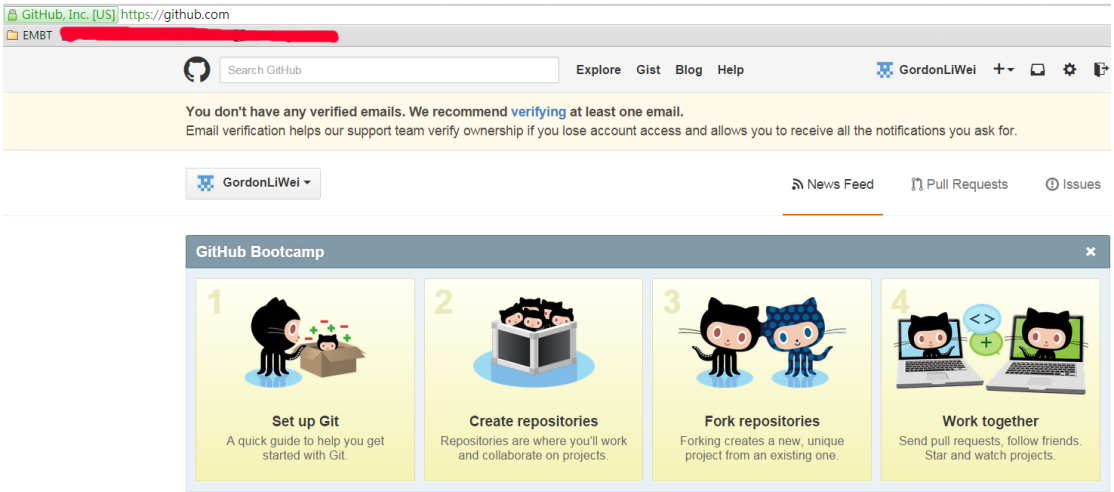
```
https://bitbucket.org/
```

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

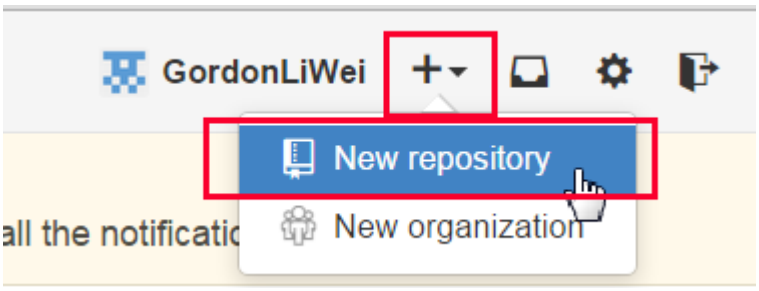
首先到 https://github.com/點選如下圖的”Sign up for GitHub”並申請帳號：



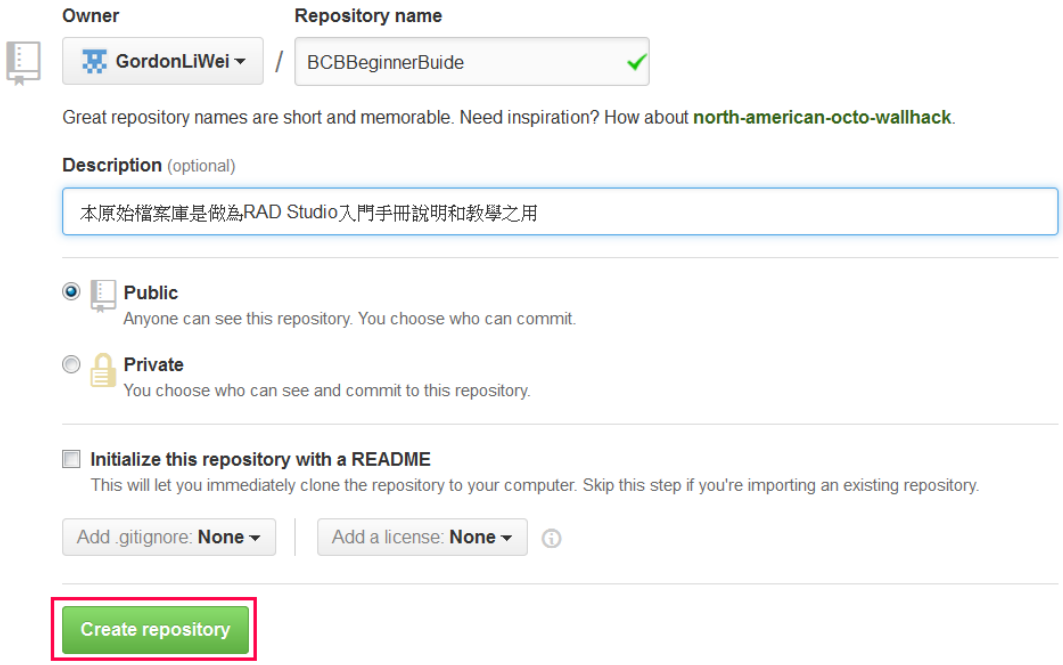
在成功申請帳號並登入之後可看到如下的畫面，不過申請完帳號後請記得回到 IDE 的 Tools | Options | Version Control | Git 類別輸入您的帳號資訊：



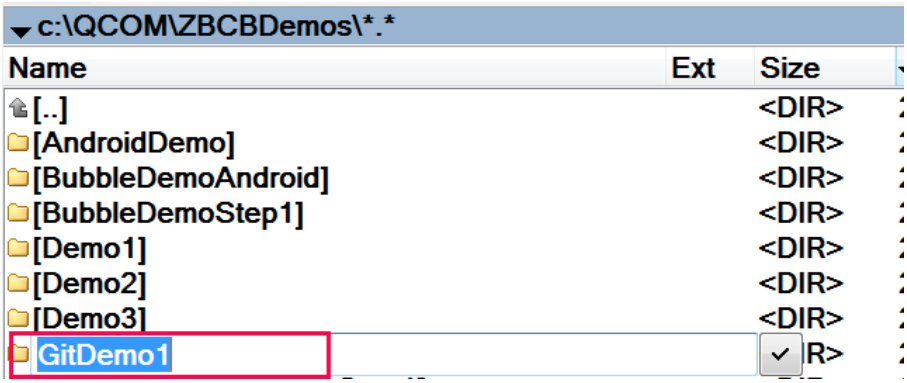
此時可點選您帳號名稱右邊的”+”號以建立一新的程式庫，如下所示：

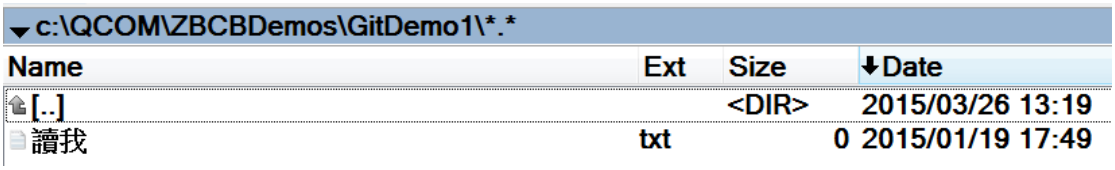


點選"+"號之後會看到如下的類似畫面，您需要為新的程式庫取一名稱和輸入一簡短的說明之後點選下方的"Crate repository"按鈕真正的建立程式庫：

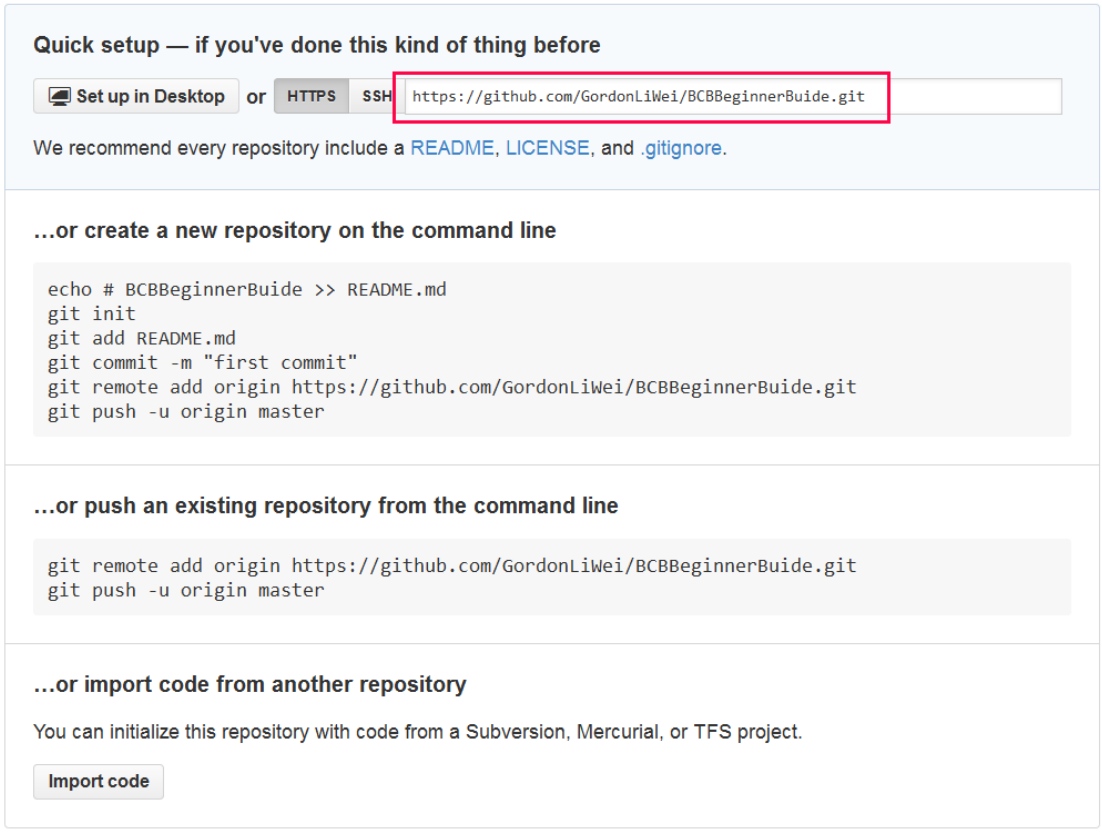


回到你的電腦中，我們現在需要在本機中先建立一個文字檔並簽入到遠端的剛才建立的程式庫中。下圖是筆者在本機的 GitDemo1 目錄中建立一個"讀我.txt"文字檔：



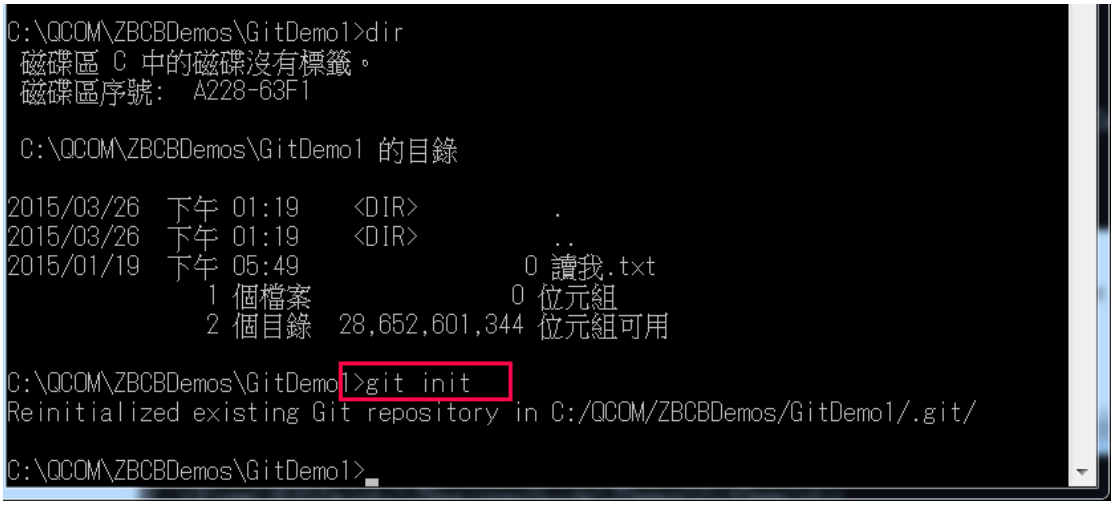


事實上當您建立了程式庫時 GitHub 便會顯示如下的內容教導您如何把一個檔案簽入到程式庫中：

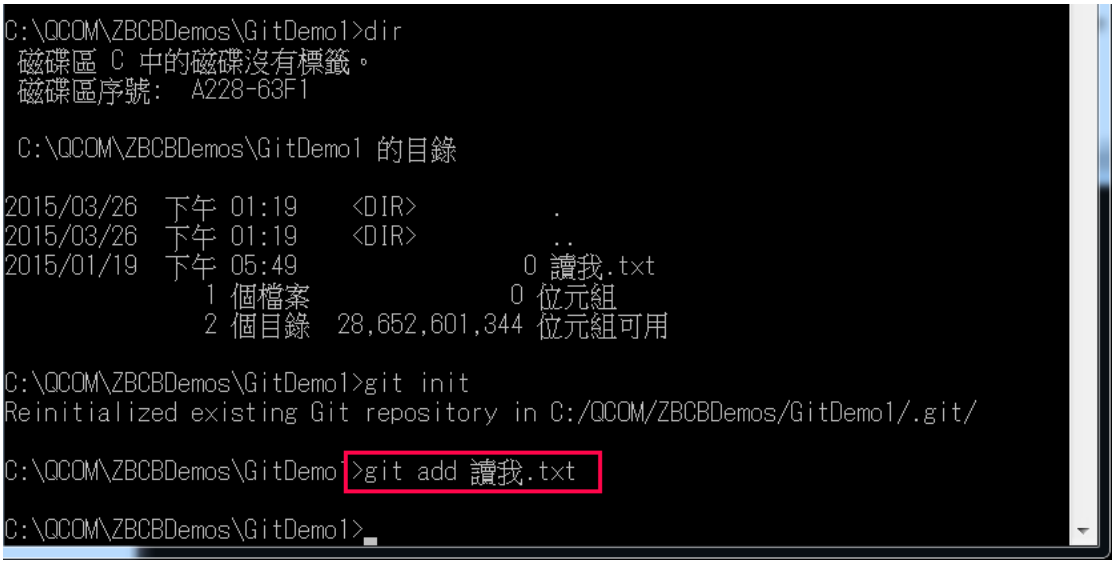


現在就讓我們使用上面的說明把 GitDemo1 目錄中”讀我.txt”文字檔簽入到遠端的 GitHub 中。

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：



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：

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端的程式庫中，在這一步驟中您可能需要輸入登入資訊：

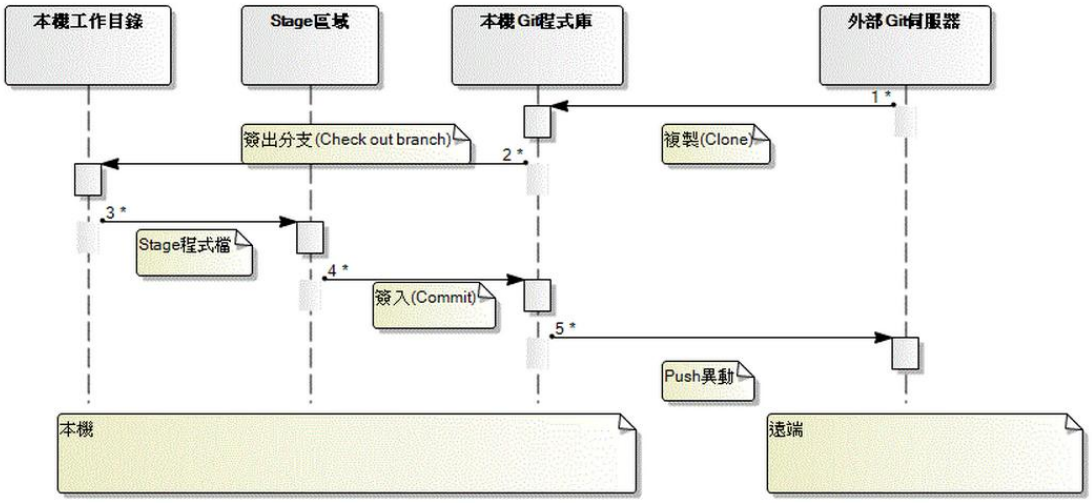
```
C:\QCOM\ZBCBDemos\BCBGitDemo1>git remote add origin https://github.com/GordonLiW
ei/BCBBeginnerBuide.git

C:\QCOM\ZBCBDemos\BCBGitDemo1>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), 228 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/GordonLiWei/BCBBeginnerBuide.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

C:\QCOM\ZBCBDemos\BCBGitDemo1>
```


最後回到瀏覽器並查看遠端 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 C++Builder IDE 來進行專案的版本控制了，而我們使用的程式庫就是剛才在遠端於 GitHub 中建立的程式庫。但在說明如何在 C++Builder 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 C++Builder IDE 中執行的動作是什麼意思。

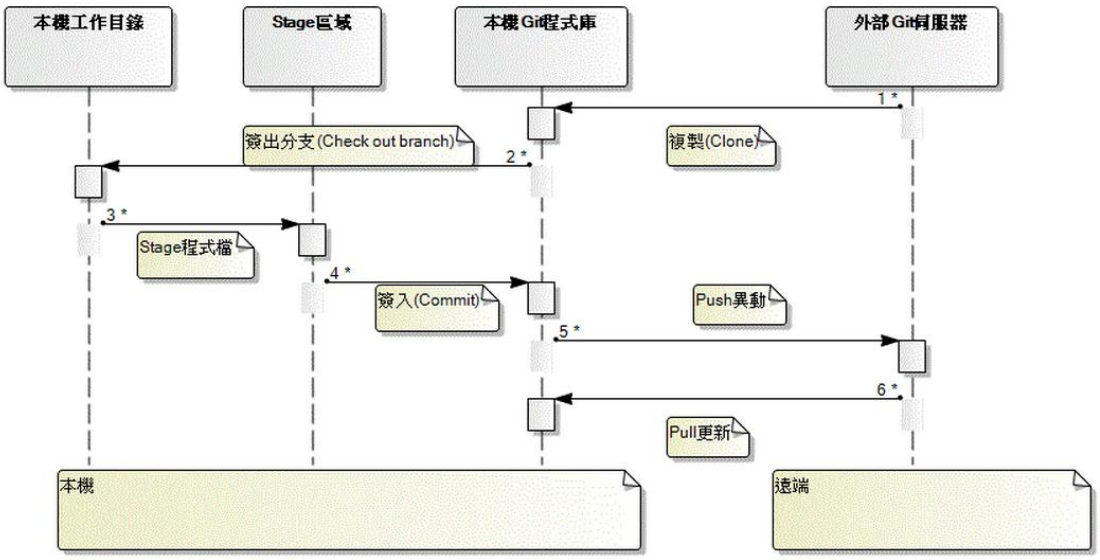
傳統的版本控制工具，例如 StarTeam 和 SubVersion 等都是屬於中央控制的 C/S 架構，每一個客戶端的開發人員從版本控制伺服器簽出程式開發和異動，最後再簽入回版本控制伺服器。但 Git 是屬於分散式的版本控制系統，Git 除了有遠端版本控制伺服器之外(即 GitHub)，Git 也會在本機建立本機程式庫，Git 可同步遠端版本控制伺服器和本機的程式庫。

讓我們使用下面的圖形簡單說明 Git 基本的運作原理 ，在前面我們於 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。



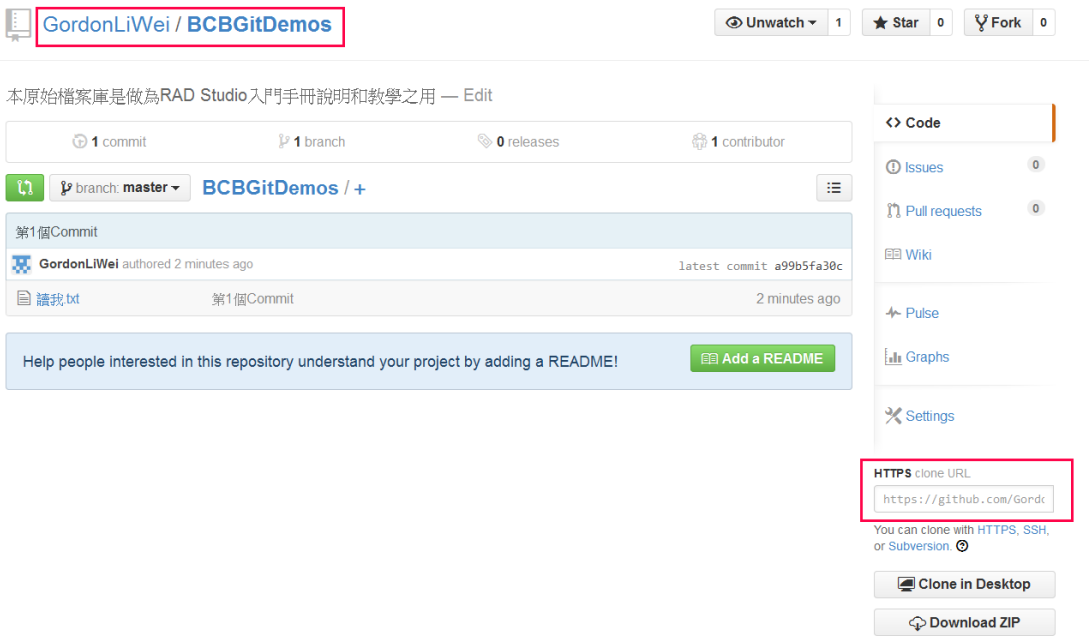
Git 基本運作原理

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：



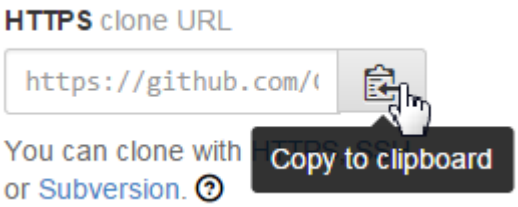
從上面的簡單說明讀者應該可以瞭解為什麼現在 Git 這麼流行，因為 Git 提供了分散式的版本控制能力，又能整合遠端 Git 伺服器和本機 Git 程式庫進行團隊開發。

在具備了 Git 基本的觀念後，我們就可以開發解釋如何在中使用 Git，筆者在 GitHub 中建立了一個”BCBGitDemos”程式庫做為上圖中的遠端伺服器：

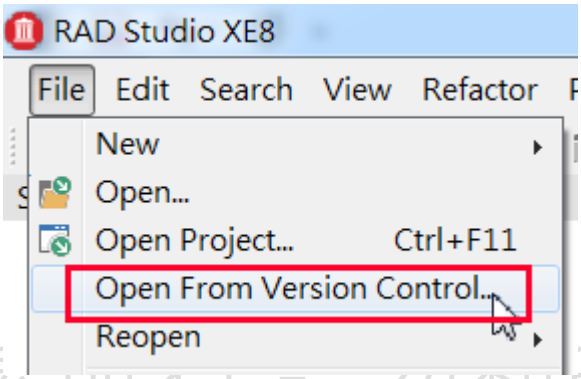


請注意上圖中這個遠端程式庫目前的分支是”Master”，而在上圖右下角的 "HTTPS clone URL”處即是指向此遠端程式庫的 URL，稍後在 C++Builder IDE 中將使用這個 URL 複製(clone)程式到本機 Git 程式庫，也就是前圖” Git 基本運作原理”中的第 1 步驟。

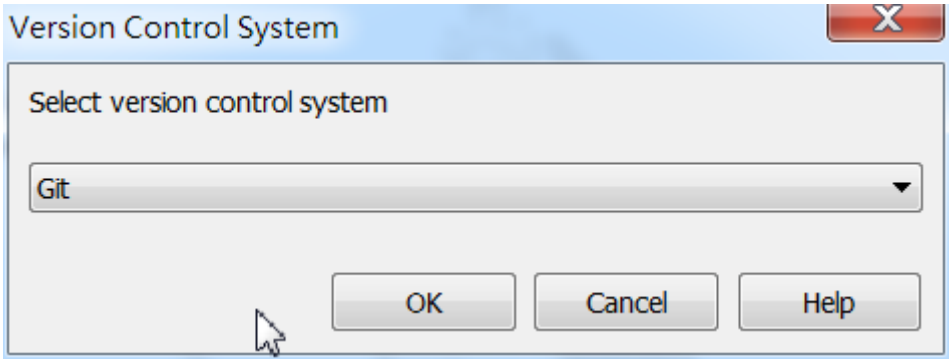
在 GitHub 的”BCBGitDemos”程式庫頁面中點選拷貝”BCBGitDemos”程式庫 URL：



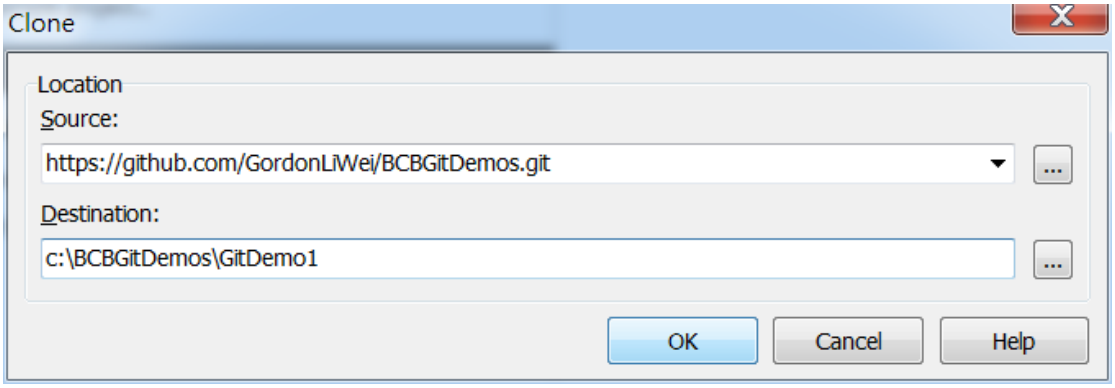
現在回到 C++Builder IDE，點選 File | Open From Version Control...選項：



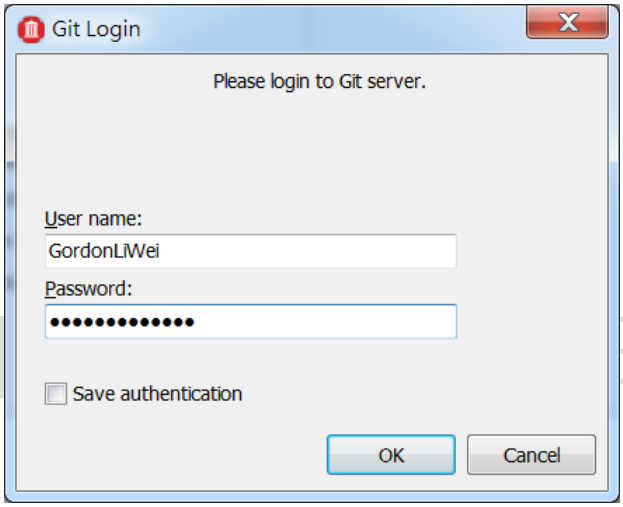
從 Version Control System 對話盒中選擇 Git：



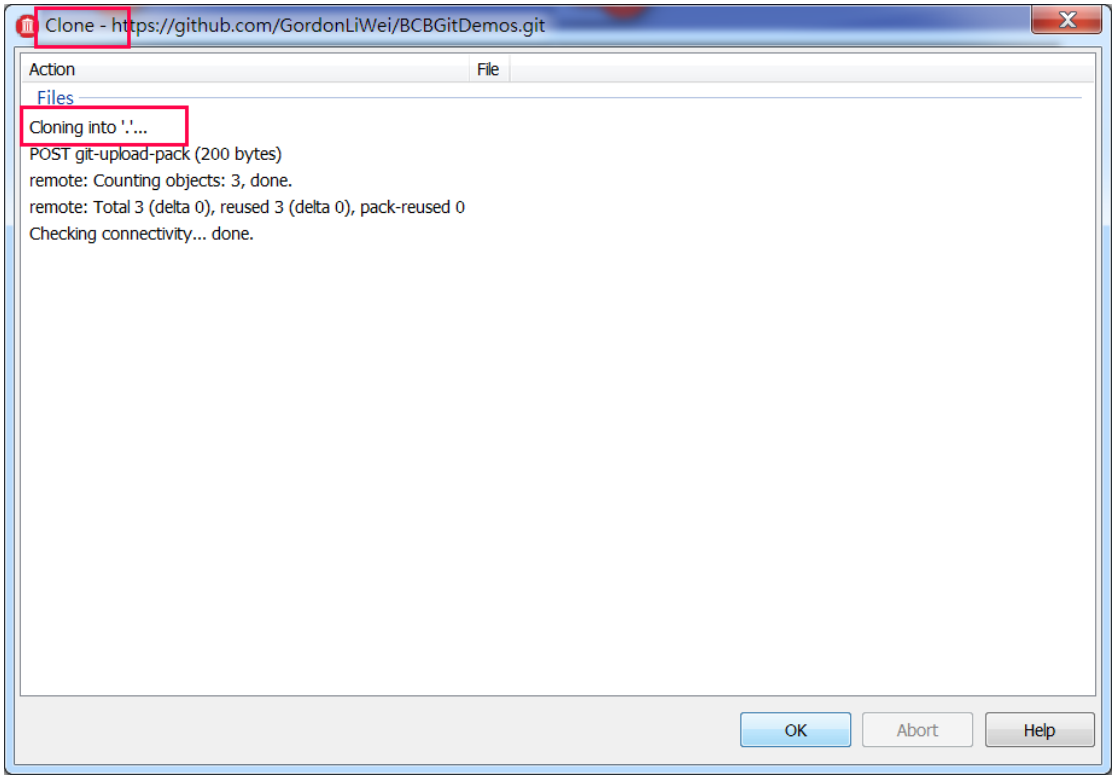
點選上圖的 OK 按鈕後在 Clone 對話盒中的 Source 欄位中貼上前面拷貝的”BCBGitDemos”程式庫 URL，再於 Destination 欄位中輸入複製(clone)到本機 Git 程式庫的目錄：



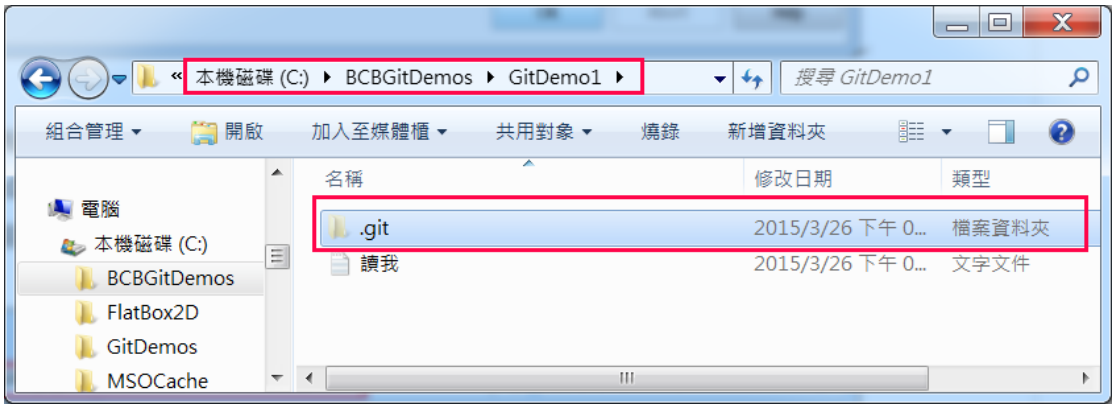
最後點選 OK 按鈕，就可以看到 C++Builder IDE 顯示如下的複製對話盒，此時 IDE 會要求您登入 Git：



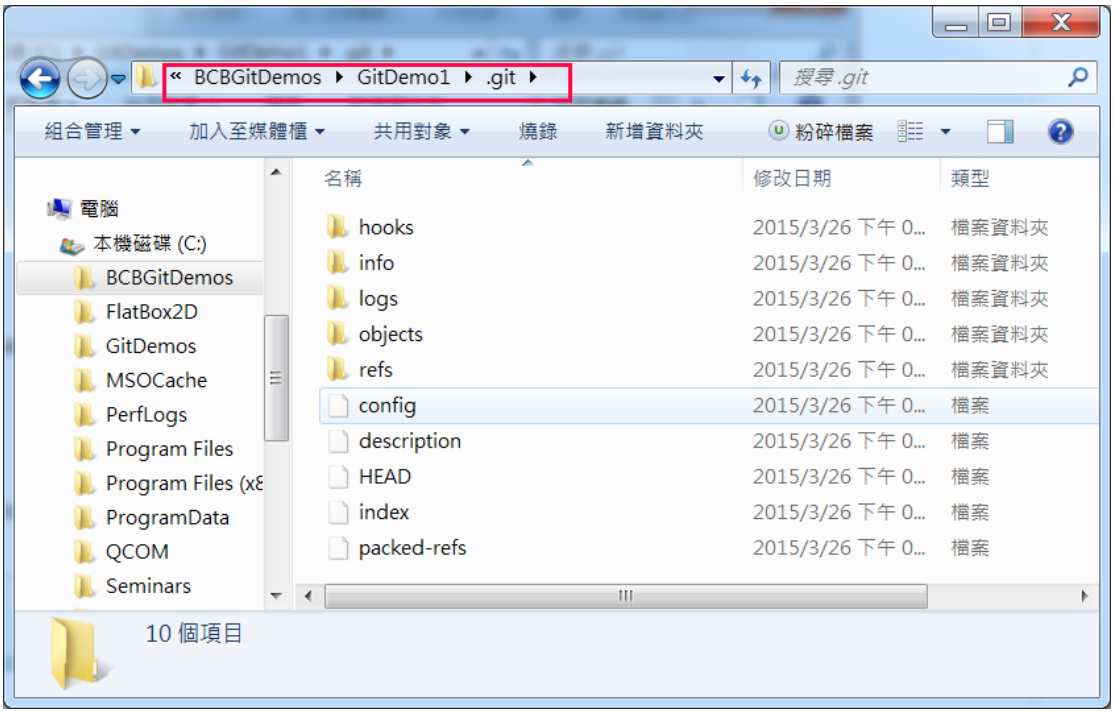
輸入前面申請建立的 Git 帳號資訊後再點選 OK 按鈕，您可以看到它顯示 Git 正在複製 (cloning) 遠端的程式庫內容到本機的 c:\BCBGitDemos\GitDemo1 目錄中。這印證了 C++Builder IDE 正在執行前圖" Git 基本運作原理"中的第 1 步驟：



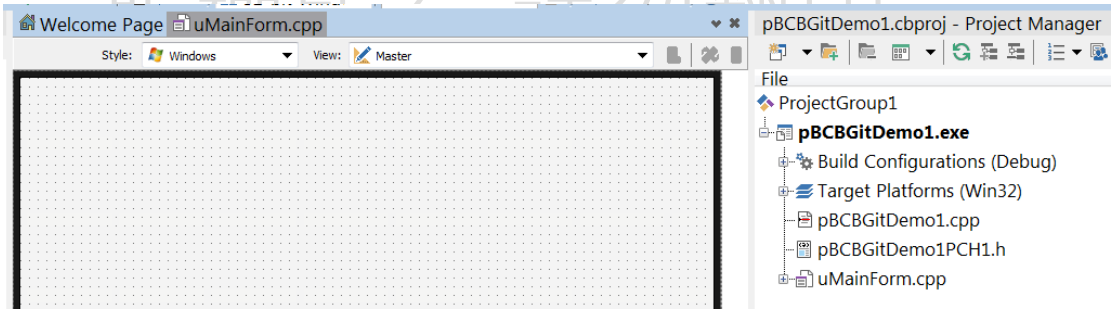
點選 OK 按鈕之後到 c:\BCBGitDemos\GitDemo1 目錄中查看，我們可以看到在 c:\BCBGitDemos\GitDemo1 目錄中果然複製了遠端"讀我.txt"而且在目錄中出現了一個 .git 子目錄：



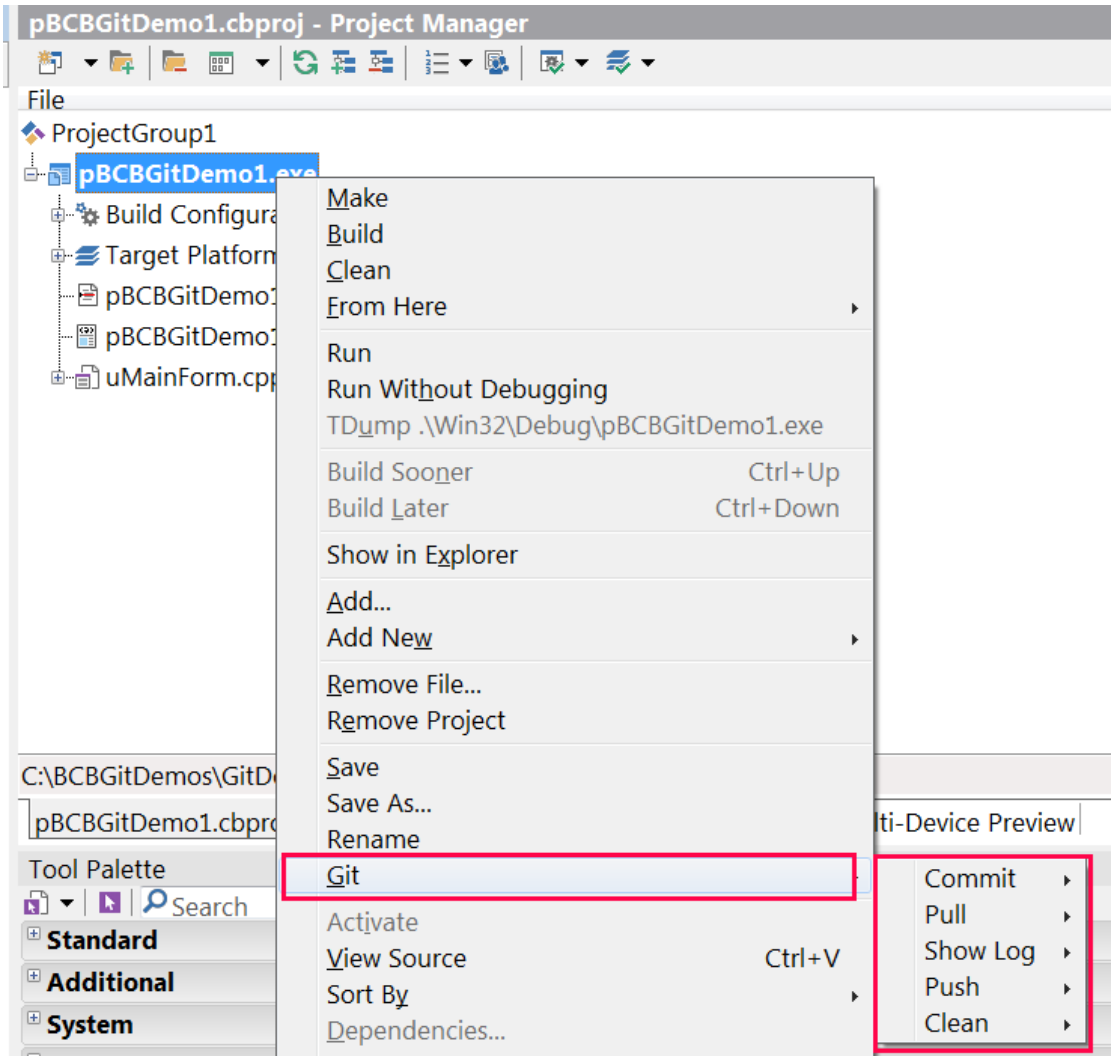
.git 子目錄就是 Git 在 c:\BCBGitDemos\GitDemo1 目錄中建立的本機程式庫，我們如果到.git 子目錄中就可以看到下面的內容，.git 子目錄是由 Git 管理的本機程式庫：



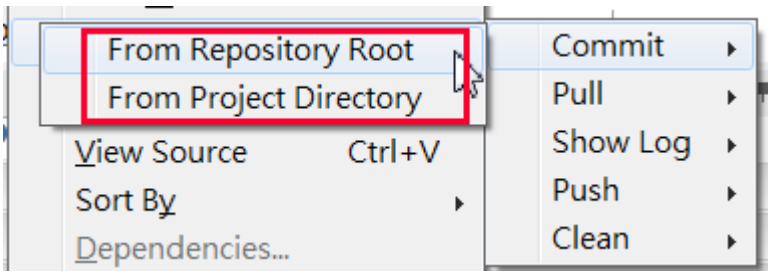
讓我們建立一個 Multi-Device 專案並儲存在 c:\BCBGitDemos\GitDemo1 目錄，如下所示：



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：



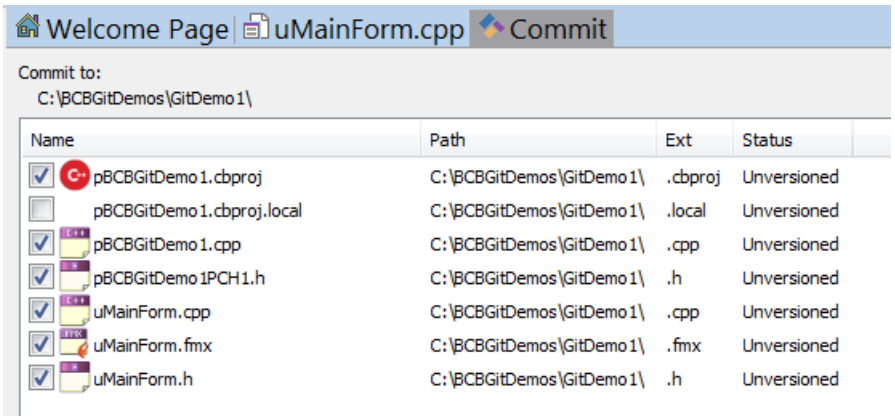
在 Git 選項中有 Commit 子選項，在其中有 2 個命令”From Repository Root”和”From Project Directory”：



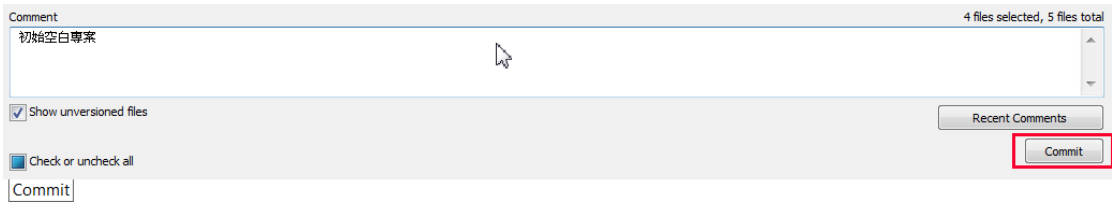
現在讓我們把這個專案簽入到本機的 Git 程式庫中，由於現在本機中本機的 Git 程式庫和 C++Builder 專案是在同一目錄中，因此上圖中的”From Repository Root”和”From Project Directory”2 個命令效果是一樣的，所以請讀者點選上圖中的 Git | Commit | From Repository Root 選項，準備把專案簽入到本機的 Git 程式庫。

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 6 個檔案，目前每一個檔案的狀態欄位都是 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。

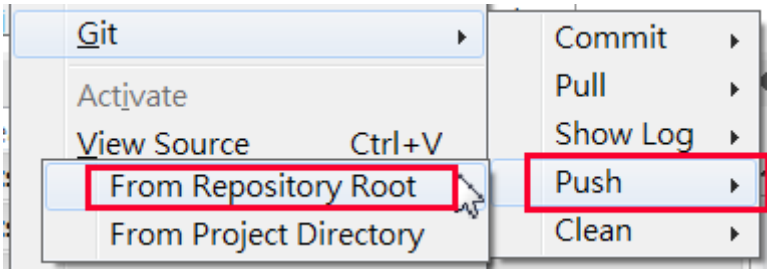


點選"Commit"按鈕就是前圖" Git 基本運作原理"中的第 3 和第 4 步驟。

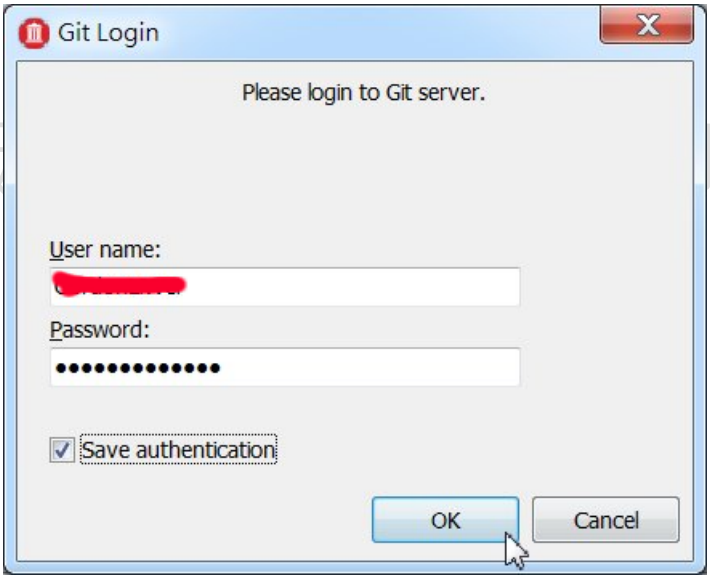
點選"Commit"按鈕之後 IDE 會執行 Git 的 Commit 命令，成功之後 IDE 的訊息視窗就會顯示一簽入代號如下所示代表簽入成功：

請注意現在我們只是把專案檔案簽入到本機程式庫中，並沒有把專案推播到遠端 Git 伺服器中，因此其他團體成員無法共享現在的開發成果，因此如果現在去 GitHub 的 BCBGitDemos 程式庫我們看不到剛才簽入的專案檔案。

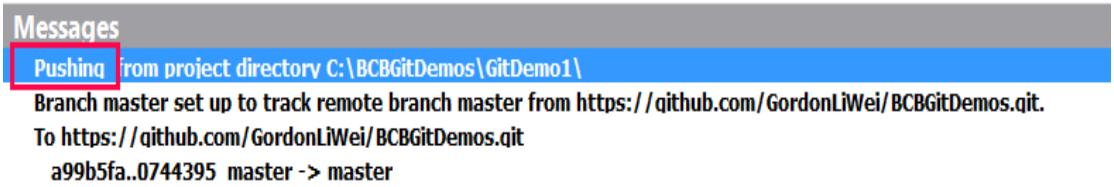
因此現在讓我們把此專案推播到遠端 Git 伺服器中，以便進行團隊開發。請點選 Git | Push | From Repository Root 選項：



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：

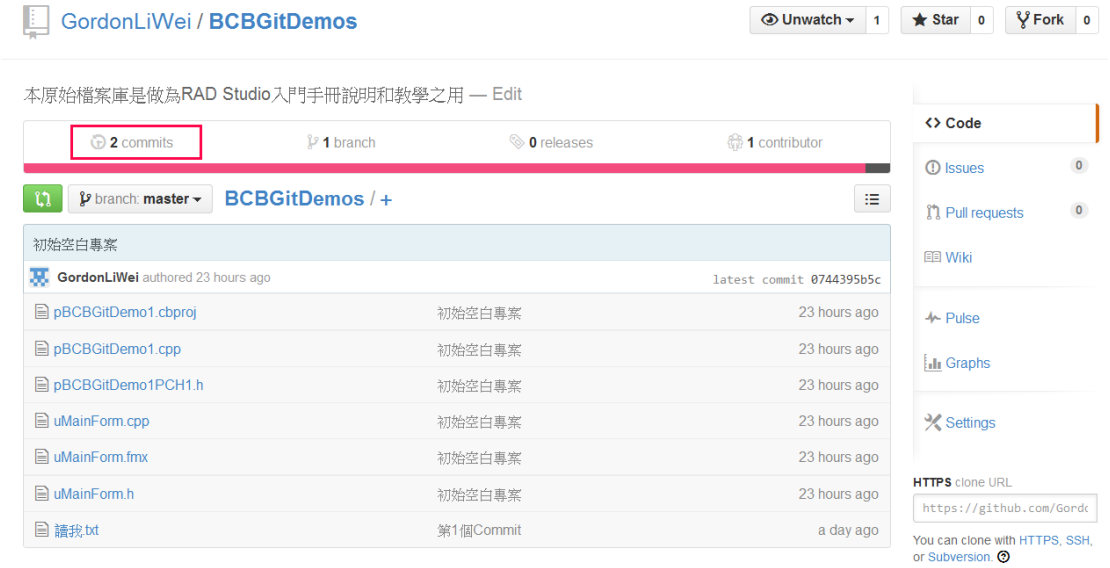


點選 OK 按鈕之後 IDE 便會執行 Git 的 Push 命令，成功之後 IDE 在訊息視窗會顯示推播成功的資訊，從下圖可以看到 IDE 的確是執行 Push 命令，而且會顯示從本機的 master 分支推播到 GitHub 的程式庫中 BCBGitDemos 程式庫的 master 分支。

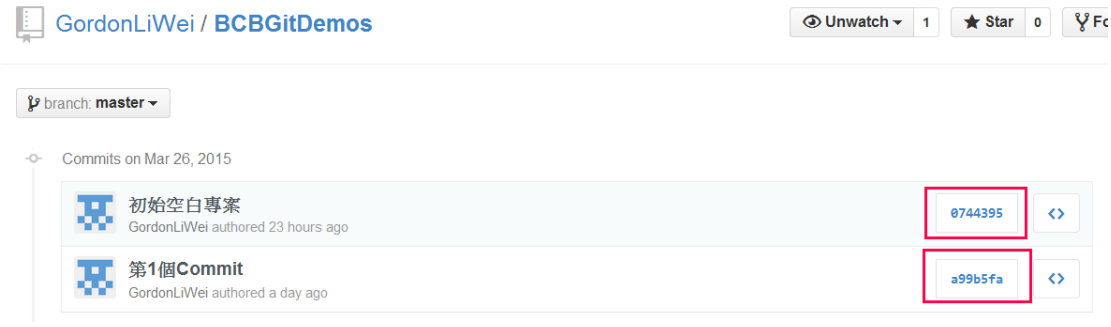


這個動作就是前圖”Git 基本運作原理”中的第 5 步驟。

現在回到 GitHub，重新整理 BCBGitDemos 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：

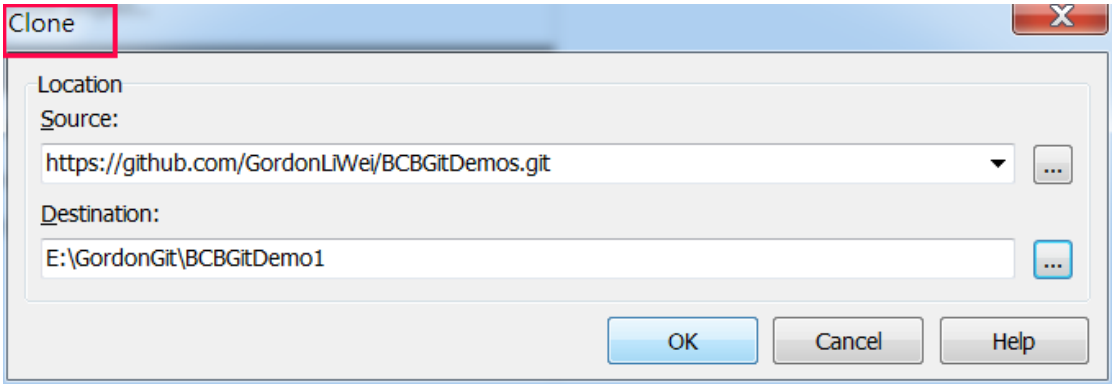


請注意上圖中顯示我們有 2 次簽入的動作，第 1 次是簽入”讀我.txt”檔案，第 2 次當然就是剛才簽入專案的動作。如果我們點選上圖中的 2 commits，那麼就可以看到如下的結果：



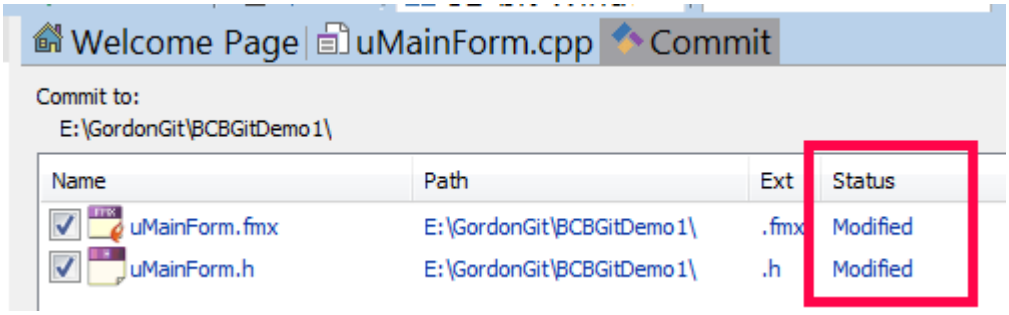
我們可以看到 2 次簽入的註解和每次的簽入代號。

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 伺服器中的專案，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，就像前面說明的流程一樣，

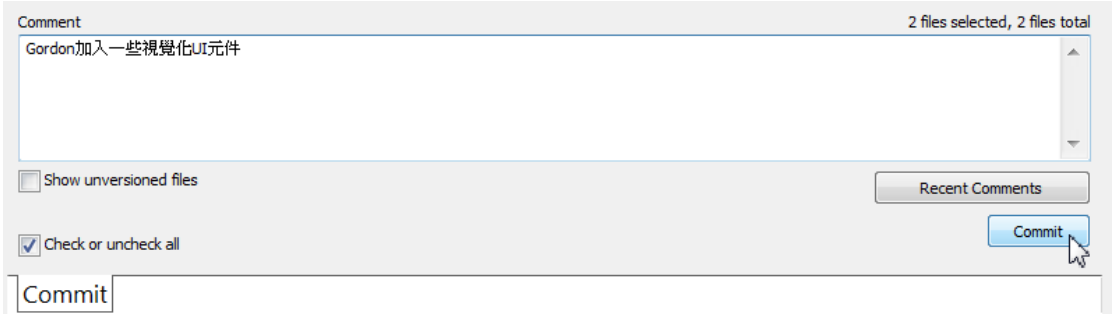


然後 Gordon 在複製到本機的專案中加入如下的 UI 元件：

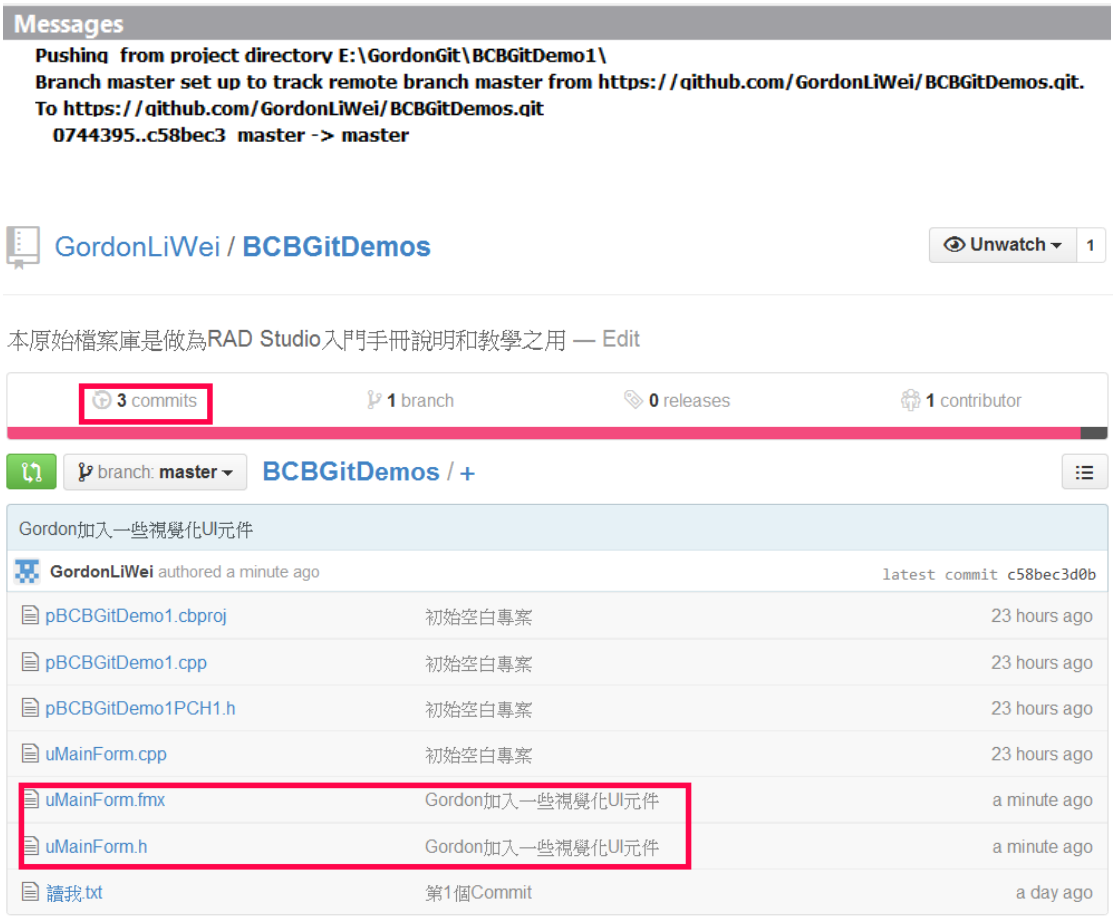
接著 Gordon 點選 Git | Commit | "From Repository Root"選項就可以在 IDE 中看到下面的 2 個專案檔案被修改過了：



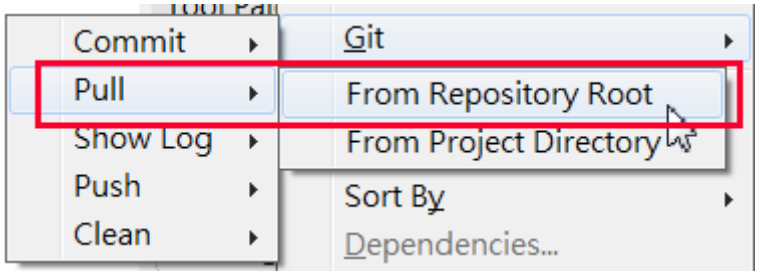
Gordon 輸入新的簽入說明再點選 Commit 按鈕簽入修改的檔案：



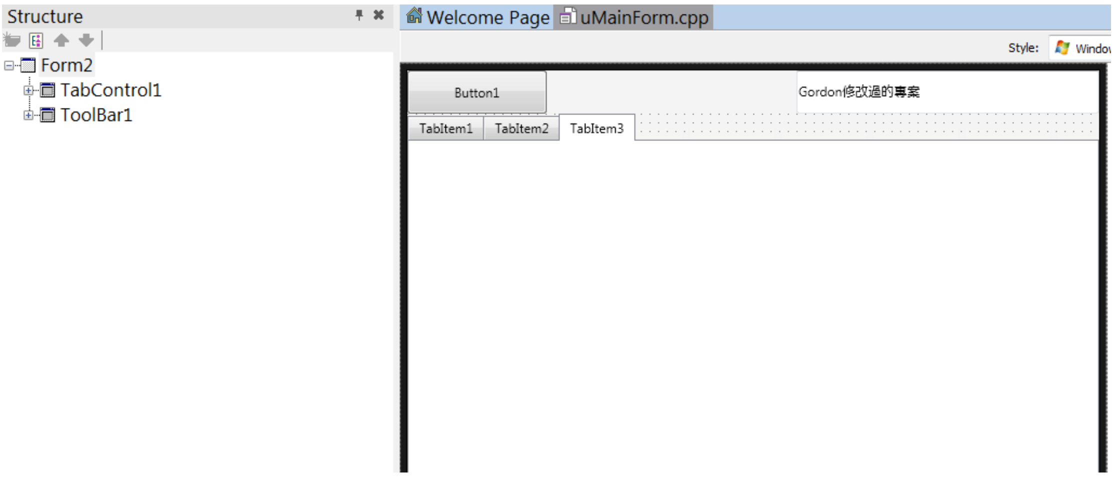
再點選 Git | Push | From Repository Root 選項把修改的專案推播回遠端 Git 伺服器，之後我們就可以看到類似下面的結果，Gordon 修改的專案果然簽回到遠端的 Git 伺服器了：

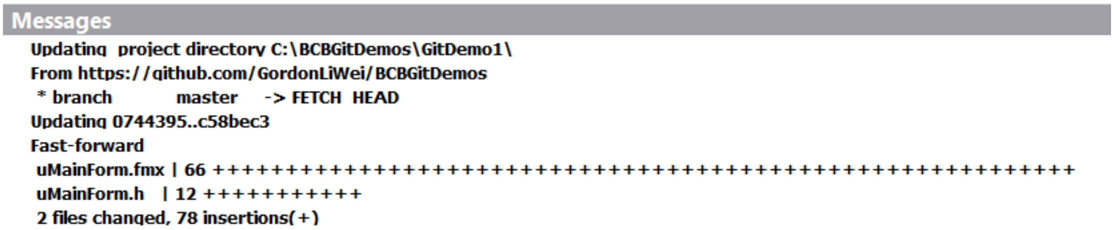


現在我們再回到原先的機器中，我們想繼續 Gordon 的修改工作，因此請在 IDE 中點選 Git | Pull | From Repository Root 把專案的異動更新回原來的本機中：



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。

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。

11-3 使用 DUNITX 單元測試框架

C++Builder 從很早的版本即使用 DUNIT 框架來支援單元測試，到了開始支援 DUNITX 框架來支援單元測試，因此從 C++Builder 開發同時支援 DUNIT 和 DUNITX。開始使用 DUNITX 主要是因為：

1. DUNITX 支援跨平台的單元測試

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 C++Builder 類別都可以成為測試類別。如此一來 C++Builder 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 C++Builder 可開發多個平台，因此為了減少需要在多個平台測試/除錯的成本，跨平台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。

11-3-1 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 C++Builder 2010 版即開始支援，讀者可以在下面的 URL 下載 DUNITX 框架：

```
https://github.com/VSoftTechnologies/DUnitX
```

到了 C++Builder 版正式加入到 C++Builder 的 IDE 中。

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]屬性標註
Fixture Setup 方法	無	使用[SetupFixture] 屬性標註或是使用建構元(Constructor)
Test Setup 方法	覆載基礎測試類別的 Setup 方法	使用 [Setup] 屬性標註
Test TearDown 方法	覆載基礎測試類別的 TearDown 方法	使用 [TearDown] 屬性標註
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)] 屬性標註
Asserts	Check(X),	Assert 類別
Asserts on Containers(IEnumerable<T>),	人工撰寫	Assert.Contains*, Assert.DoesNotContain*, Assert.IsEmpty*
使用正規表示法 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。

11-3-2 如何成為 DUNITX 測試框架的測試類別

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 C++Builder 類別都可以成為測試類別

因此下面的範例類別 TMyTestObject 就是一個 DUNITX 測試框架的測試類別：


```
#include <DUnitX.TestFramework.hpp>
#include <stdio.h>

#pragma option --xrtti

class __declspec(C++Builderrtti) TMyTestObject : public TObject
{
public:
  virtual void __fastcall SetUp();
  virtual void __fastcall TearDown();

__published:
  void __fastcall Test1();
  void __fastcall Test2();
};
```

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 C++Builder 的 DUNITX 中由於沒有像 C++Builder 的語言屬性來標誌測試方法，因此 C++Builder 的測試方法都需要宣告在類別的 **__published** 部分，因此下面的 TestPushedMessageCount 和 TestUnPushedMessageCount 方法就成為了測試方法。

```
class __declspec(C++Builderrtti) TMyTestObject : public TObject
{
public:
  virtual void __fastcall SetUp();
  virtual void __fastcall TearDown();

__published:
  void __fastcall TestPushedMessageCount();
  void __fastcall TestUnPushedMessageCount();
};
```


也非常的簡單。

在 C++Buidler 中 DUNITX 測試類別必須加入 2 個表頭檔案：

```
#include <DUnitX.TestFramework.hpp>
#include <stdio.h>
```

此外也必須使用#pragma 開啟 rtti 功能：

```
#pragma option -xrtti
```

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] 屬性標註的方法就是 Setup 方法，DUNITX 在執行測試之前都會先執行使用[Setup] 屬性標註的方法。而使用[TearDown] 屬性標註方法就是 TearDown 方法，DUNITX 在執行完測試方法之後最後會執行使用 [TearDown] 屬性標註的方法。但由於 C/C++語言沒有屬

性標註機製，因此在 C++Builder 中是使用 SetUp()方法和 TearDown()方法來代替：

```
 class __declspec(C++Builderrtti) TMyTestObject : public TObject
{
public:
  virtual void __fastcall SetUp();
  virtual void __fastcall TearDown();

__published:
  void __fastcall TestPushedMessageCount();
  void __fastcall TestUnPushedMessageCount();
};
```

規則 4 使用 Dunitx::Testframework::Assert 類別測試執行結果

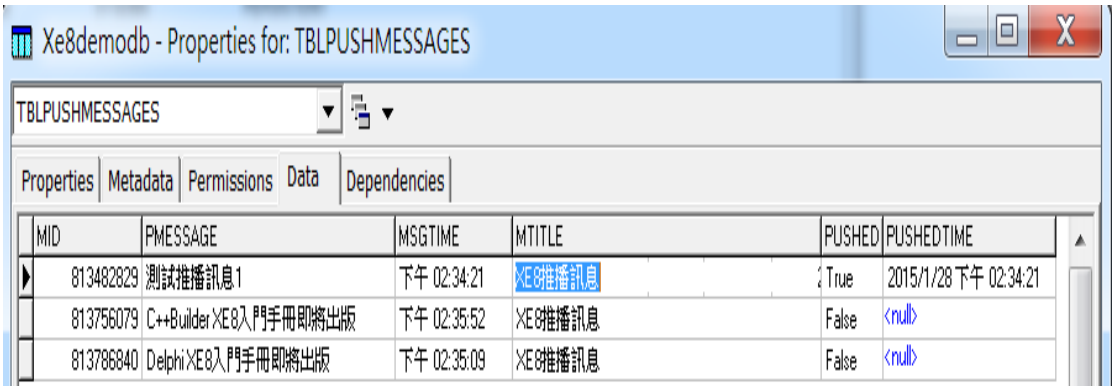
使用 DUNITX 框架最主要的目地當然就是測試開發人員撰寫的程式碼是否正確，每當我們實作了一個方法之後就應該撰寫一個測試方法來測試實作方法的正確性，實作第 2 個方法之後再撰寫第 2 個測試方法來測試實作方法的正確性並且不斷的一直執行所有先前的測試方法看看後來加入的實作方法有沒有影響先前的實作程式碼(依照敏捷開發方式您應該反過來，先撰寫測試方法再實作方法)。

在 DUNITX 框架的 Assert 類別中提供了許多的方法讓程式師可以進行各種不同的測試，下面的表格列出了這些經常使用的測試方法：

功能	說明
Pass	檢查函式是否可正常執行
Fail	檢查函式是否執行失敗
AreEqual	檢查項目是否相等
AreNotEqual	檢查項目是否不相等
AreSame	檢查 2 個項目是否相同
AreNotSame	檢查 2 個項目是否不相同
Contains	檢查項目是否包含在一個串列物件中
DoesNotContain	檢查項目是否不包含在一個串列物件中
IsTrue	檢查條件是否為真
IsFalse	檢查條件是否為假
IsEmpty	檢查項目的數值是否是空白

IsNotEmpty	檢查項目的數值是否不是空白
IsNull	檢查項目是否是 null
IsNotNull	檢查項目是否不是 null
WillRaise	檢查一個匿法是否會產生例外
StartsWith	檢查字串是否以特定的子字串開頭
InheritsFrom	檢查是否繼承自特定的類別
IsMatch	檢查項目是否符合特定的樣式

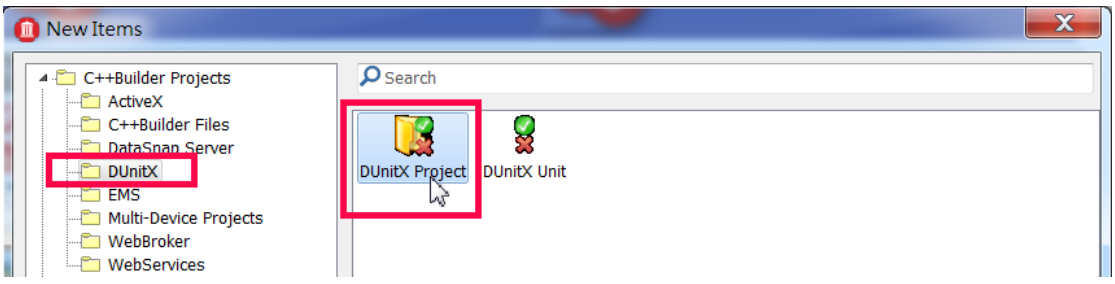
讓我們使用一個資料表 TBLPUSHMESSAGES 來說明，一開始 TBLPUSHMESSAGES 中包含了下面的資料：



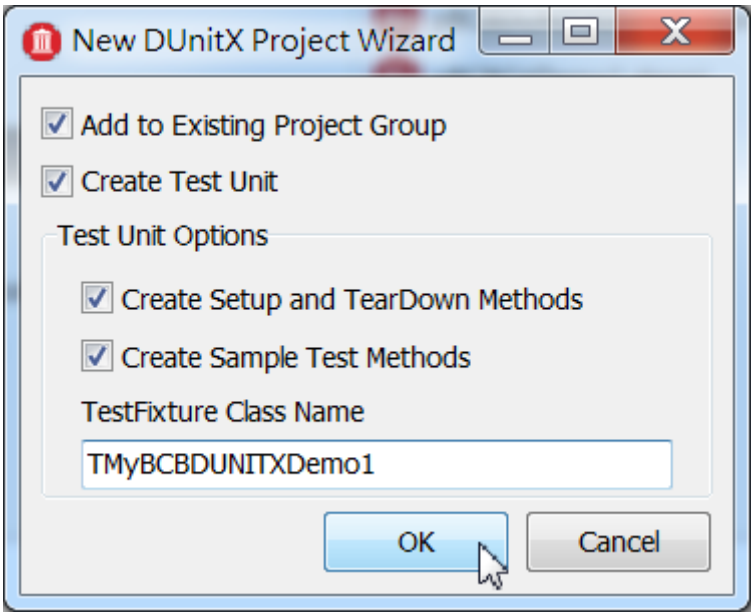
TBLPUSHMESSAGES 一共有 3 筆資料，其中已經推播的資料筆數是 1，另 2 筆資料是尚未推播的(PUSHED 欄位)。接下來讓我們直接使用 C++Builder IDE 中的 DUNITX 功能來說明如何進行測試。

11-3-3 使用 DUNITX 單元測試框架

在 C++Builder IDE 中使用 DUNITX 框架非常簡單，只需點選 File|New|Other...選項再於 C++Builder Projects|DUnitX 項目中可看到 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：



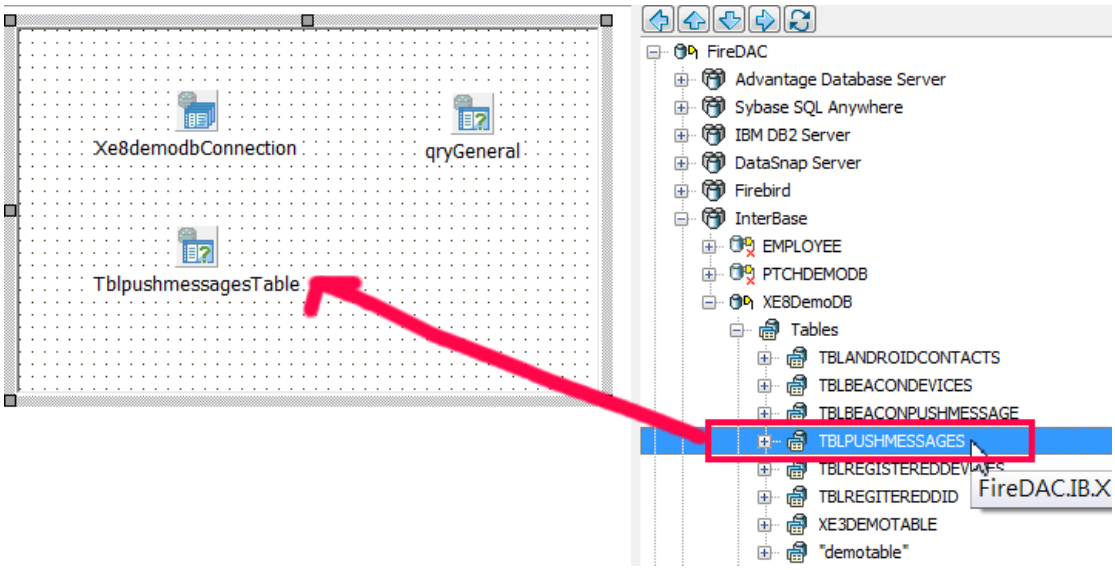
點選上面的選項和 OK 按鈕後 IDE 便會產生如下的程式碼，除了前面介紹的 SetUp()和 TearDown()方法之外 DUnitX 精靈也會產生 2 個樣本測試方法：


```
#pragma option --xrtti
class __declspec(C++Builderrtti) TMyBCBDUNITXDemo1 : public TObject
{
public:
  virtual void __fastcall SetUp();
  virtual void __fastcall TearDown();

__published:
  void __fastcall Test1();
  void __fastcall Test2();
};


void __fastcall TMyBCBDUNITXDemo1::SetUp()
{
}

void __fastcall TMyBCBDUNITXDemo1::TearDown()
```

```
{
}

void __fastcall TMyBCBDUNITXDemo1::Test1()
{
  // TODO
  String s("Hello");
  Dunitx::Testframework::Assert::IsTrue(s == "Hello");
}

void __fastcall TMyBCBDUNITXDemo1::Test2()
{
  // TODO
  String s("Hello");
  Dunitx::Testframework::Assert::IsTrue(s == "Bonjour"); // This
fails for illustrative purposes
}

static void registerTests()
{
  TDUnitX::RegisterTestFixture(__classid(TMyBCBDUNITXDemo1));
}
#pragma startup registerTests 33
```


我們現在想實作一個資料模組讓它可以處理 TBLPUSHMESSAGES 資料表中的資料，也由於我們會撰寫實作程式碼，因此我們要使用 DUNITX 框架來測試這些實作程式碼。

首先請在範例專案中加入一個資料模組，再於 IDE 的 Data Explorer 中把 TBLPUSHMESSAGES 資料表拖入到資料模組 ，再加入一個名為 qryGenral 的 TFDQuery 元件：



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。

11-4 App Tethering

App Tethering 技術允許開發人員使用軟體連結 PC 和各種客戶端的設備，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。

藉由 App Tethering 技術在不同硬體平台中的軟體可以：

1. 互相遠端控制執行行動命令

2. 傳遞和共享資料

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絡區域中設備的連結，到了 XE7 App Tethering 可藉由直接提供 IP 地址連結也允許使用藍牙協定連結，到了 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 允許不同硬體平台中的軟體不同硬體平台中的軟體，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
共享資源	使用資源方式共享資料，並提供資料更新的能力
以暫時資源一次傳送資料	從一平台 App 一次傳遞資料到另一平台的 App

由於 C++Builder 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：

元件	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 元件
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

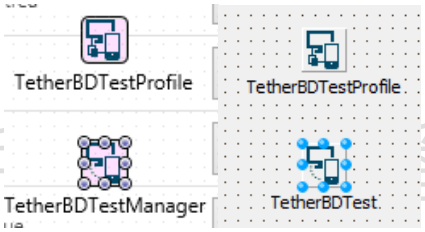
現在我們就可以開始說明 BDShoppingList 是如何工作的。請在 C++Builder IDE 中開啟下面目錄中的

```
c:\Users\Public\Documents\Embarcadero\Studio\16.0\Samples\CPP\RTL\Tethering\BDShoppingList\
```

BDShoppingList.groupproj 專案群組。這個專案群組中包含了一個 Windows 端的 VCL 應用程式和一個能在 Android/iOS 平台中執行的 App。在下面的小節中將說明 Windows 端和 Android/iOS 平台中的 App 如何藉由 App Tethering 技術共同執行運作。

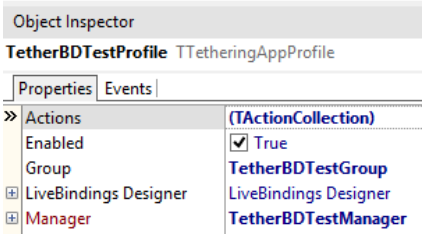
11-4-1 手機端 TetherDBClient 如何工作

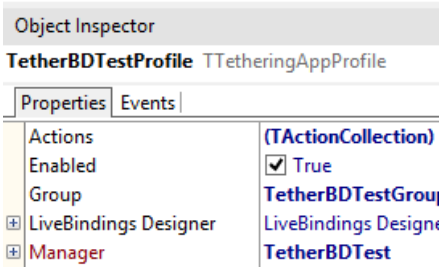
開啟 TetherDBClient 專案的主表單，您會看到下面左方的 TetheringManager 和 TetheringAppProfile 元件，如何此時再開啟 Windows 端的 VCL 應用程式的主表單就會看到到下方右邊同樣的 2 個元件：



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：





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：

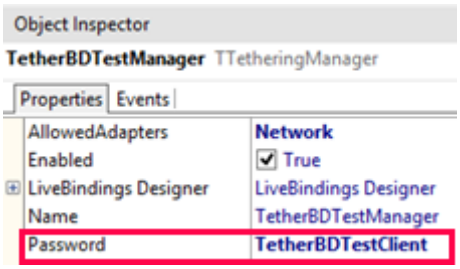
```
  void __fastcall AutoConnect(unsigned Timeout, const
System::UnicodeString ATarget = System::UnicodeString())/*
overload */;
  void __fastcall AutoConnect(const System::UnicodeString ATarget
= System::UnicodeString())/* overload */;
#ifndef _WIN64
  void __fastcall AutoConnect(unsigned Timeout, const
System::DynamicArray<System::UnicodeString> ATargetList)/*
overload */;
  void __fastcall AutoConnect(const
System::DynamicArray<System::UnicodeString> ATargetList)/*
overload */;
#else /* _WIN64 */
  void __fastcall AutoConnect(unsigned Timeout, const
System::TArray__1<System::UnicodeString> ATargetList)/* overload
*/;
  void __fastcall AutoConnect(const
System::TArray__1<System::UnicodeString> ATargetList)/* overload
*/;
#endif /* _WIN64 */

  void __fastcall DiscoverManagers(unsigned Timeout, const
System::UnicodeString ATarget = System::UnicodeString())/*
```

```
overload */;
  void __fastcall DiscoverManagers(const System::UnicodeString
ATarget = System::UnicodeString())/* overload */;
#ifndef _WIN64
  void __fastcall DiscoverManagers(unsigned Timeout, const
System::DynamicArray<System::UnicodeString> ATargetList)/*
overload */;
  void __fastcall DiscoverManagers(const
System::DynamicArray<System::UnicodeString> ATargetList)/*
overload */;
#else /* _WIN64 */
  void __fastcall DiscoverManagers(unsigned Timeout, const
System::TArray__1<System::UnicodeString> ATargetList)/* overload
*/;
  void __fastcall DiscoverManagers(const
System::TArray__1<System::UnicodeString> ATargetList)/* overload
*/;
#endif /* _WIN64 */
```


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 個方法於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 的連結對象。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：

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

void __fastcall
TForm1::TetherBDTestManagerRemoteManagerShutdown(const TObject
*Sender,  const UnicodeString ManagerIdentifier)
{
  CheckRemoteProfiles();
}
```

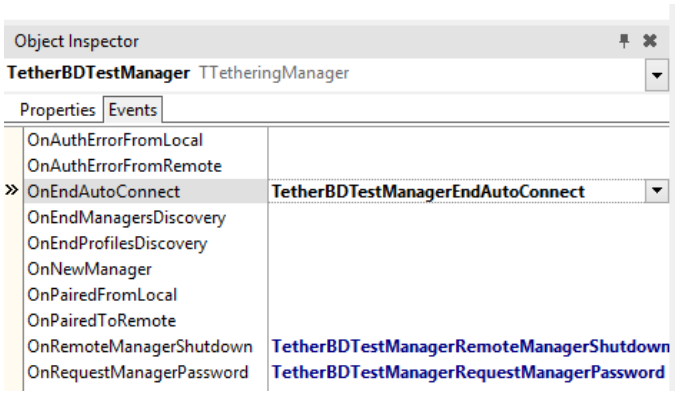
您也可以使用密碼來保護合法的連結：



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：

```
void __fastcall
TForm1::TetherBDTestManagerRequestManagerPassword(const TObject
*Sender,
const UnicodeString RemoteIdentifier, UnicodeString &Password)
{
  Password = "TetherBDTest";
}
```


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。

而在遠端一方，例如下面的 Windows 端 TetherDatabase 專案中，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。

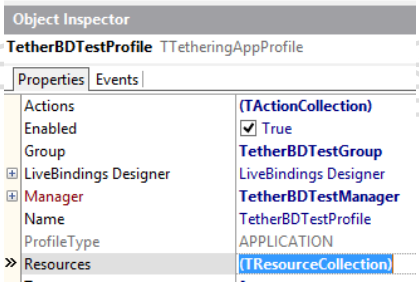
11-4-2 Windows 端 TetherDatabase 如何工作

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 一個共享的資源讓手機端 TetherDBClient App 可擷取遠端 Windows 的 TetherDatabase 應用程式的執行結果。因此 Windows 端 TetherDatabase 專案提供了：

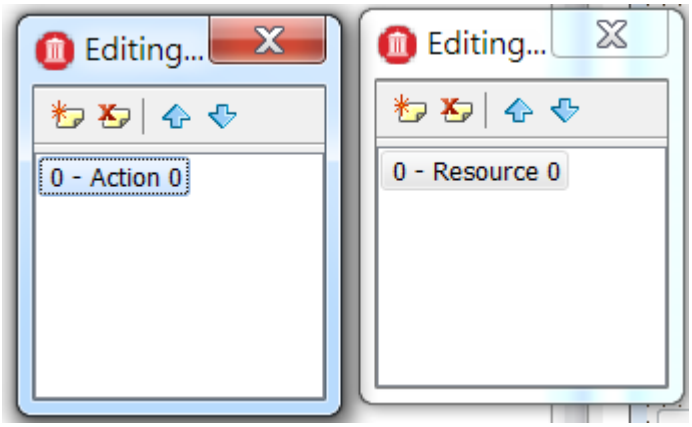
1. 一個輸出執行命令

2. 一個共享資源

要提供輸出執行命令和共享資源，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：

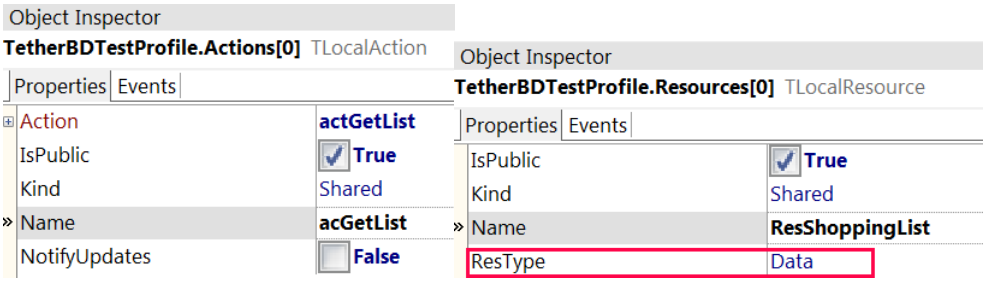


在它的 Actions 和 Resources 特性中提供了一個命令物件 TAction 和一個共享資源物件 TLocalResource：

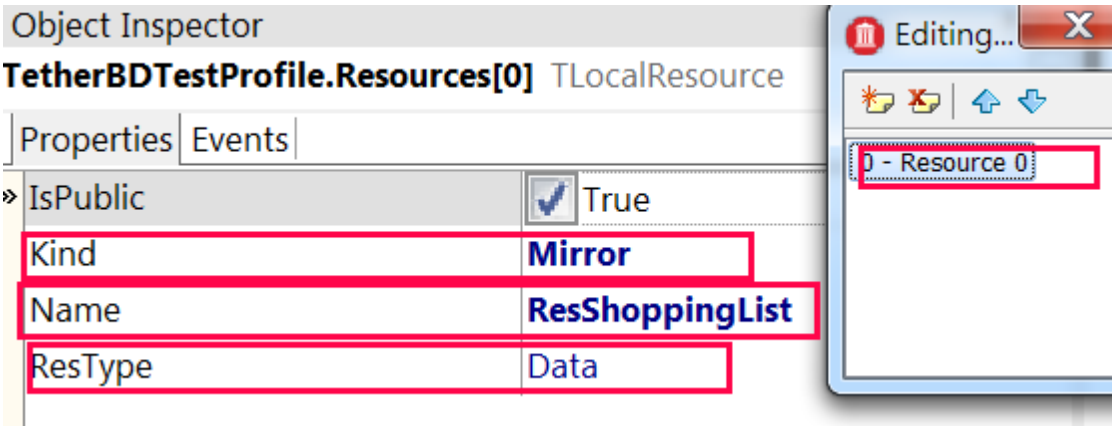


命令物件在被遠端手機 App 呼叫執行之後就會把執行結果以共享資源方式讓雙方可存取，因此共享資源物件 TLocalResource 的 ResType 的型態是設定

為 Data，Kind 特性設定為”Shared”，而且共享資源是命名為”ResShoppingList”：

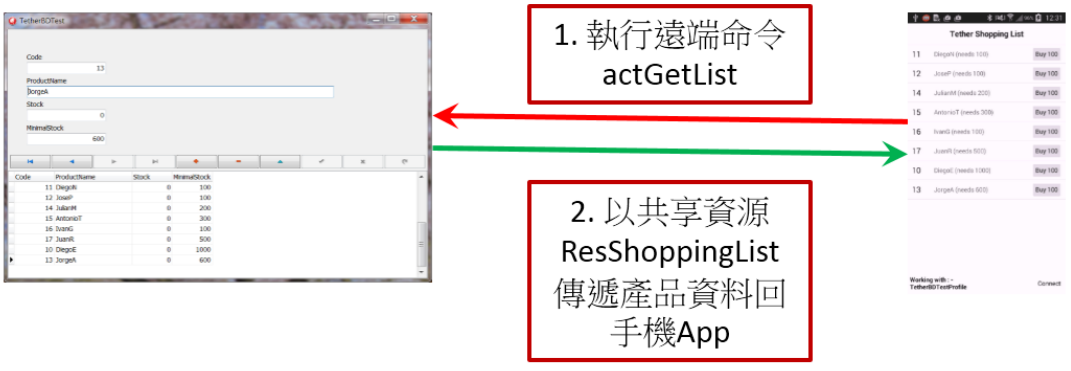


因此在手機端 TetherDBClient 專案中的 TetherBDTestProfile 元件其 Resources 特性中也要加入一個命名為”ResShoppingList”的共享資源物件但其 Kind 特性設定為”Mirror”：



11-4-3 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：



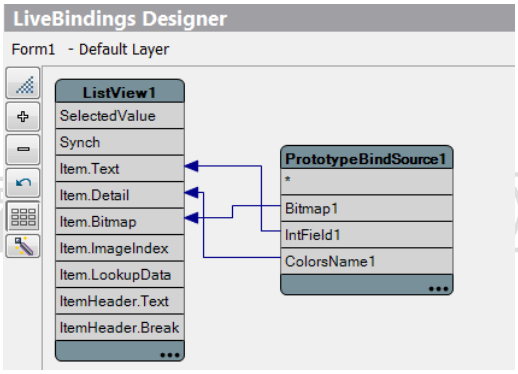
手機中的 App 可以藉由 App Tethering 技術看到需要購買那麼產品，為什麼？這是因為…

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 (!FIsConnected)
    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享資源中，接著 App Tethering 就會觸發 TetherDBClient 專案端的 OnResourceReceived 事件，TetherDBClient 專案端再使用 LiveBinding 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享資源中等待手機端去存取執行結果：

```
void __fastcall
TForm1::TetherBDTestProfileResources0ResourceReceived(const
TObject *Sender,
        const TRemoteResource *AResource)
{
  TStringList * lStrings = NULL;
  ListView1->Items->Clear();
  if(AResource->Value.AsString != "NONE") {
```

```
      lStrings = new TStringList();
      try
      {
          lStrings->Delimiter = ':';
          lStrings->DelimitedText = AResource->Value.AsString;
          for(int i = 0; i < lStrings->Count; i++){
              TListViewItem * lItem = ListView1->Items->Add();
              TStringList * itemParts = new TStringList();
              try
              {
                  itemParts->Delimiter = '-';
                  itemParts->DelimitedText = lStrings->Strings[i];
                  lItem->Text = itemParts->Strings[1];
                  lItem->Detail = itemParts->Strings[0] + " (needs
" + itemParts->Strings[2] + ")";
              }
               __finally
              {
                  delete itemParts;
              }
          }
      }
       __finally
      {
          delete lStrings;
      }
  }
}
```


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：

```
void __fastcall TForm1::ListView1ButtonClick(const TObject
*Sender, const TListViewItem *AItem,
        const TListItemSimpleControl *AObject)
{
  TetherBDTestProfile->SendString(TetherBDTestManager->RemotePro
files->Items[0],
      "Buy item", AItem->Text);
}
```


TetherDatabase 專案端

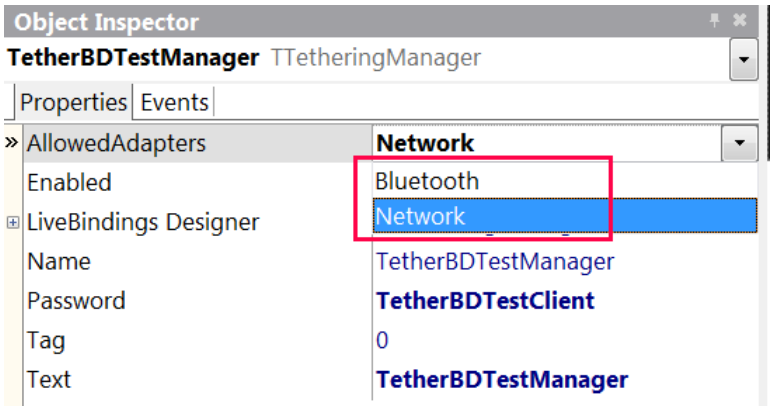
此時 Windows 端的 TetherDatabase 專案就會觸發 OnResourceReceived 事件，接著就執行採購和更新資料的工作：

```
void __fastcall
TForm2::TetherBDTestProfileResources0ResourceReceived(const
TObject *Sender,
        const TRemoteResource *AResource)
{
  if(AResource->ResType == TRemoteResourceType::Data) {
      int pId = StrToInt(AResource->Value.AsString);
      CDSProducts->First();
```

```
      while(!CDSProducts->Eof) {
          if(CDSProductsCode->Value == pId) {
              CDSProducts->Edit();
              CDSProductsStock->Value =  CDSProductsStock->Value +
100;
              CDSProducts->Post();
              break;
          }
          CDSProducts->Next();
       }
    }
}
```

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 地址，例如在呼叫 TTetheringManager 元件的 AutoConnect 方法時直接使用一個 IP 地址參數即可：

```
TetherBDTestManager->AutoConnect(2000,"192.168.0.27");
```

11-5 藍牙開發

C++Builder 從 XE7 開始便支持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 C++Builder 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台對於藍牙技術的支援差異，下面的表格列出了 4 個平台對傳統藍牙和低耗電藍牙 BLE 的支援程度：

	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	✔	✔

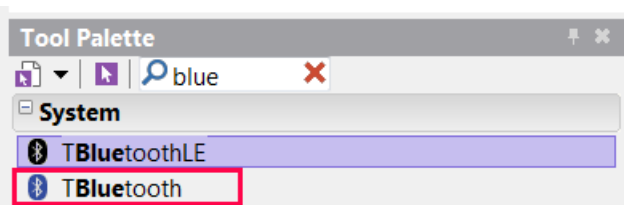
在本小節中我們將說明如何使用 TBlueTooth 元件來開發藍牙 App。

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：

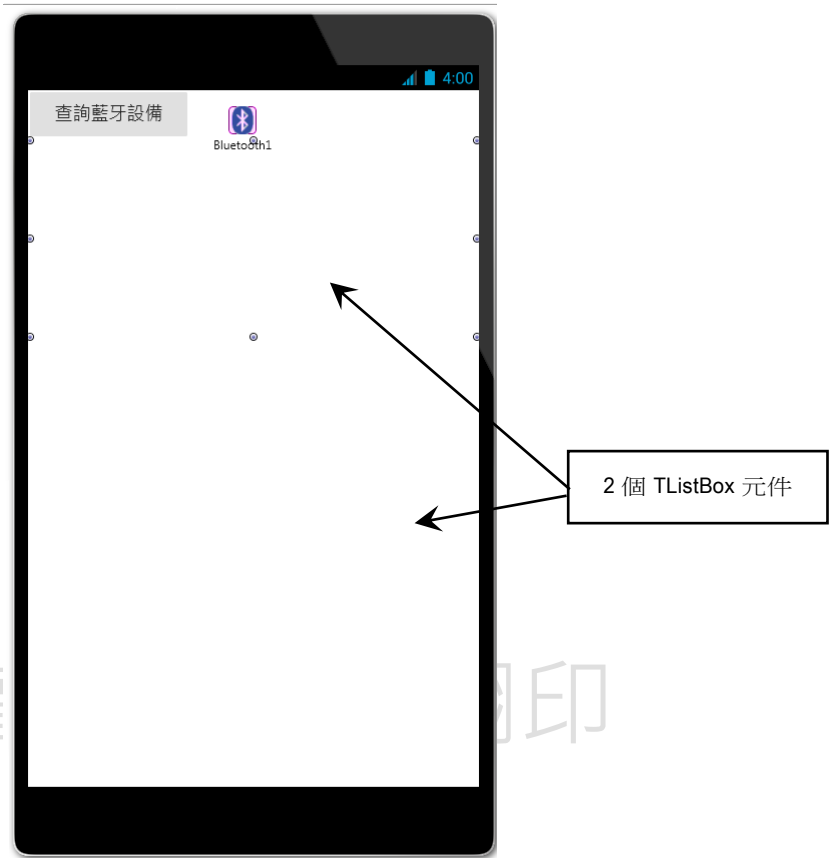
1. 搜尋附近的藍牙設備
2. 搜尋藍牙設備提供的服務
3. 配對藍牙設備
4. 建立配對藍牙設備之間的通訊管道
5. 傳遞資料

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：



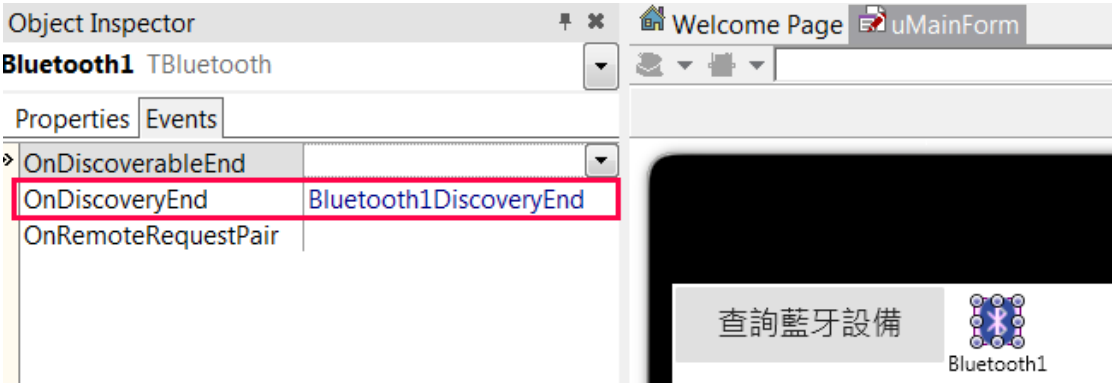
要查詢藍牙設備很簡單，只要呼叫 TBlueTooth 元件的 DiscoverDevices 方法即可，DiscoverDevices 接收一個搜尋藍牙設備時間的參數：

```
void __fastcall DiscoverDevices(int ATimeout);
```

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：

```
void __fastcall TfmMainForm::Button1Click(TObject *Sender)
{
  Bluetooth1->DiscoverDevices(10000);
}
```

在 TBlueTooth 元件查詢完畢之後它會觸發 OnDiscoveryEnd 事件：

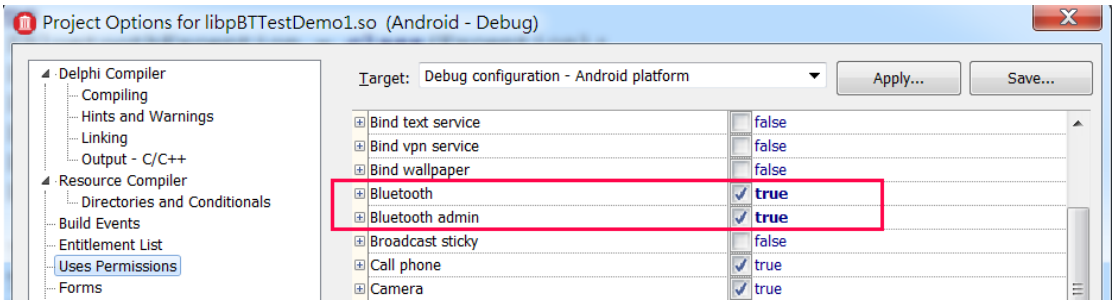


在 OnDiscoveryEnd 事件會傳入所有找到的藍牙設備參數 ADeviceList ，它的型態是 TBluetoothDeviceList。TBluetoothDeviceList 的 Items 特性是 TBluetoothDevice 類別物件，每一個 TBluetoothDevice 類別物件代表一個搜尋到的藍牙設備。TBluetoothDevice 類別物件即可提供藍牙設備的名稱，地址等資訊，而且我們也可以使用它來獲得此藍牙設備提供的服務資訊。

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：


```
void __fastcall TfmMainForm::Bluetooth1DiscoveryEnd(TObject *
const Sender, TBluetoothDeviceList * const ADeviceList)
{
  lbBTDevices->Items->Clear();
  for (int iCount = 0; iCount < ADeviceList->Count; iCount++)
  {
  lbBTDevices->Items->Add(ADeviceList->Items[iCount]->DeviceName
+ ":" + ADeviceList->Items[iCount]->Address);
  }
  FDiscoverDevices = ADeviceList;
  TabControl1->TabIndex = 0;
}
```

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 存取藍牙的權限。請點選 Project|Options...選項並如下圖勾選 Bluetooth 和 Bluetooth admin 權限：



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：

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：

```
void __fastcall TfmMainForm::lbBTDevicesItemClick(TCustomListBox *
const Sender, TListBoxItem * const Item)
{
  DisplayDeviceServices(FDiscoverDevices->Items[Item->Index]);
}
```

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：

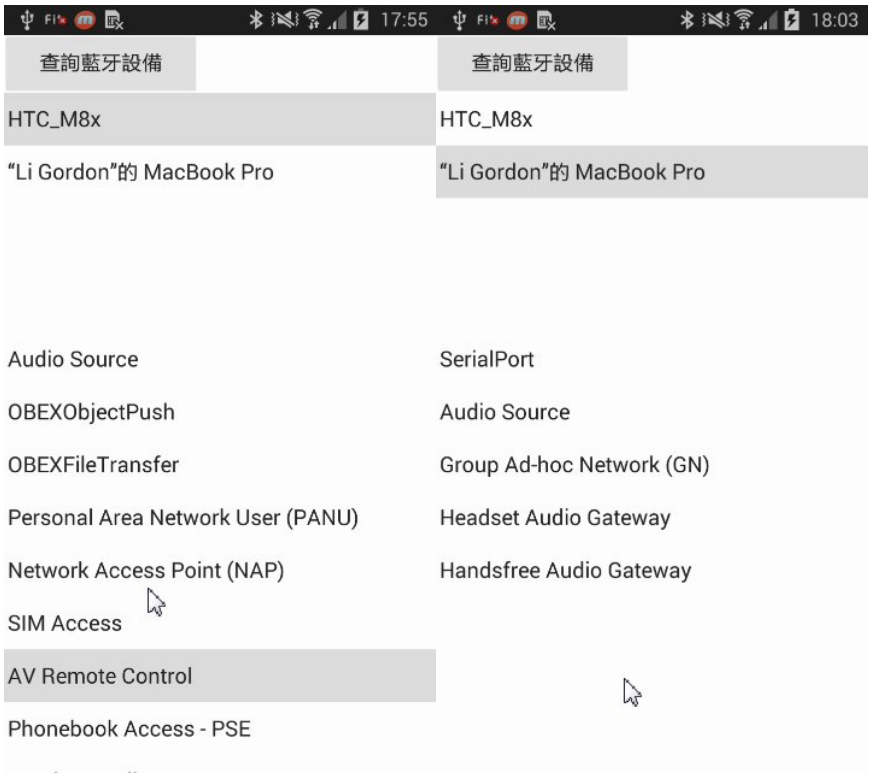
```
System::Bluetooth::TBluetoothServiceList* __fastcall
GetServices(System::Bluetooth::TBluetoothDevice* const ADevice);
```

GetServices()方法會回傳藍牙設備提供的所有服務，在回傳的 TBluetoothServiceList 物件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：

```
void TfmMainForm::DisplayDeviceServices(TBluetoothDevice*
aDevice)
{
  BTDeviceServices->Items->Clear();
  TBluetoothServiceList * sl = Bluetooth1->GetServices(aDevice);
  for (int iCount = 0; iCount < sl->Count; iCount++)
  {
  BTDeviceServices->Items->Add(sl->Items[iCount].Name);
  }
}
```

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 元件中：

```
void __fastcall TfmMainForm::btnPairClick(TObject *Sender)
{
  System::Bluetooth::TBluetoothDevice* pDevice =
FDiscoverDevices->Items[lbBTDevices->ItemIndex];
  if (Bluetooth1->Pair(pDevice))
  {
  cbPairDevices->Items->Add(FDiscoverDevices->Items[lbBTDevices-
>ItemIndex]->DeviceName);
  cbPairDevices->ItemIndex =
cbPairDevices->Items->IndexOf(FDiscoverDevices->Items[lbBTDevice
s->ItemIndex]->DeviceName);
  }
}
```


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：

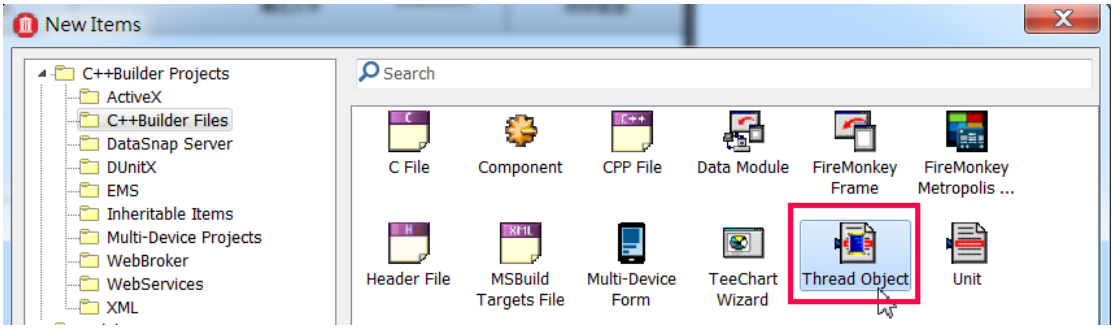
點選配對按鈕之後可以看到 Samsung S4 顯示的配對要求：

按下確定之後可以看到成功和 HTC M8 配對成功了：

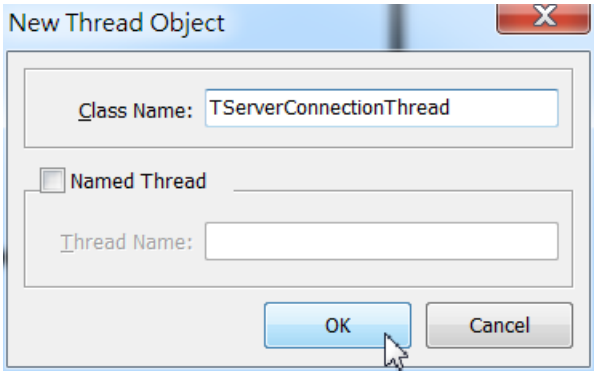
完成了配對步驟之後我們就可以開始下一個步驟”建立配對藍牙設備之間的通訊管道”，一旦通訊管道建立成功之後就可以在藍牙設備之間傳遞資料了。

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客戶端建立通訊管道

3. 伺服端的藍牙設備先建立 TBluetoothServerSocket 物件，並使用它監聽客戶端的連結請求，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客戶端傳送來的資料

4. 客戶端藍牙設備建立 TBluetoothSocket 物件並使用它傳遞資料給伺服端的藍牙設備

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客戶端的連結請求並讀取客戶端傳送來的資料。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：

```
void __fastcall TfmMainForm::btnCreateTextServiceClick(TObject
*Sender)
{
  CreateBTTextService();
}
```

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：

```
TBluetoothServerSocket* __fastcall CreateServerSocket(const
System::UnicodeString AName, const GUID &AUUID, bool Secure);
```

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 String ServiceName = "範例藍牙傳遞資料服務";
const String ServiceGUI =
"{9827B0D2-66FF-4B30-A3C3-59F38ED59B61}";
```

下面就是 CreateBTTextService() 方法的實作程式碼，005 行建立 TServerConnectionThread 物件，006 行建立 TBluetoothServerSocket 物件並指定給 TServerConnectionThread 物件的 ServerSocket 特性以便 TServerConnectionThread 物件使用來接受客戶端的連絡請求，最後在 007 行啟示執行 TServerConnectionThread 物件代表的獨立執行緒：


```
001    void TfmMainForm::CreateBTTextService()
002    {
003      try
004      {
005        scThread = new TServerConnectionThread(true);
006        scThread->ServerSocket =
Bluetooth1->CurrentAdapter->CreateServerSocket(ServiceName,
StringToGUID(ServiceGUI), false);
007        scThread->Start();
008        PostBTMessage(" - 成功建立服務 : '" + ServiceName + "'");
009      }
010      catch(Exception& ex)
011      {
012        PostBTMessage(ex.Message);
013      }
014    }
015
016    void TfmMainForm::PostBTMessage(const String sMessage)
```

```
017    {
018      mmMessages->Lines->Add(sMessage);
019      mmMessages->GoToTextEnd();
020    }
```

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 物件準備接受客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 即是使用來接受客戶端藍牙連結並接收資料的 TBluetoothSocket 物件變數，而 FData 則是 FSocket 從客戶端接收的資料：

```
 #ifndef uTServerConnectionThreadH
#define uTServerConnectionThreadH
//---------------------------------------------------------------------
#include <System.Classes.hpp>
#include <System.Bluetooth.hpp>
#include <System.Bluetooth.Components.hpp>
//---------------------------------------------------------------------
class TServerConnectionThread : public TThread
{
private:
  TBluetoothServerSocket *FServerSocket;
  TBluetoothSocket *FSocket;
  TBytes FData;

  void __fastcall TThreadMethod(void);
  void __fastcall TThreadMethodException(void);
protected:
  void __fastcall Execute();
public:
  __fastcall TServerConnectionThread(bool CreateSuspended);
    __fastcall virtual ~TServerConnectionThread(void);

__property TBluetoothServerSocket* ServerSocket =
{read=FServerSocket, write=FServerSocket, nodefault};
};
```

當 TServerConnectionThread 物件在前面 CreateBTTextService()方法的第 7 行啟動之後就會執行它的 Execute()方法，在 013 行先接受客戶端藍牙連結以取得 TBluetoothSocket 物件，如果連結成功就在 020 行呼叫 ReadData()方法接受客戶端藍牙傳遞來的資料，接著在 023 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   void __fastcall TServerConnectionThread::Execute()
002   {
003     TBluetoothSocket *ASocket;
004     String Msg;
005
006     while (!Terminated)
007     {
008       try
009       {
010         ASocket = NULL;
011         while ( (!Terminated) && (ASocket == NULL) )
012         {
013           ASocket = FServerSocket->Accept(100);
014         }
015         if(ASocket != NULL)
016         {
017           FSocket = ASocket;
018           while (!Terminated)
019           {
020             FData = ASocket->ReadData();
021             if(FData.Length > 0)
022             {
023               Synchronize(TThreadMethod);
024             }
025             Sleep(100);
026           }
027         }
028       }
029       catch (Exception& ex)
```

```
030       {
031         Msg = ex.Message;
032         Synchronize(TThreadMethodException);
033       }
034     }
035   }
036
037   void __fastcall
TServerConnectionThread::TThreadMethod(void)
038   {
039
  fmMainForm->PostBTMessage(TEncoding::UTF8->GetString(FData));
040   }
041
042   void __fastcall
TServerConnectionThread::TThreadMethodException(void)
043   {
044       fmMainForm->PostBTMessage("伺服端連結已關閉: " + Msg);
045   }
```


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：

```
__fastcall
TServerConnectionThread::~TServerConnectionThread(void)
{
  delete FSocket;
  delete FServerSocket;
}
```

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：

```
void __fastcall TfmMainForm::btnRemoveTextServiceClick(TObject
*Sender)
{
  FreeBTTextService();
}
```

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：

```
void TfmMainForm::FreeBTTextService()
{
  if (scThread != NULL)
  {
  scThread->Terminate();
  scThread->WaitFor();
  delete scThread;
  PostBTMessage(" - 連結服務已移除 -");
  }
}
```

現在我們已完成藍牙伺服端的開發工作，再來需要完成客戶端的開發工作，客戶端需要建立一個 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，再建立客戶端的 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()方法的主程式碼部份，SendDataToServer()方法先呼叫 CreateClientSocket()方法建立客戶端 TBluetoothSocket 物件，再呼叫 SendData()方法傳遞資料：

```
 void TfmMainForm::SendDataToServer()
{
  try
  {
  CreateClientSocket();
  SendData();
  }
```

```
  catch (Exception& Ex)
  {
    PostBTMessage(Ex.Message);
    delete FClientSocket;
  }
}
```

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()方法建立客戶端的 TBluetoothSocket 物件，請注意 CreateClientSocket()方法的第 1 個參數是伺服端和客戶端使用來建立連結的相同 GUID 值。

在建立了客戶端 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    void TfmMainForm::CreateClientSocket()
002    {
003      TBluetoothDevice *pairedDevice;
004
005      if (FClientSocket == NULL)
006      {
007        pairedDevice =
GetThePairedDevice(cbPairDevices->Items->Strings[cbPairDevices->
ItemIndex]);
008        PostBTMessage(GetServiceName(pairedDevice,
ServiceGUI));
009        FClientSocket =
pairedDevice->CreateClientSocket(StringToGUID(ServiceGUI),
false);
010        if (FClientSocket != NULL)
011        {
012          FClientSocket->Connect();
013          PostBTMessage(L"藍牙已連結");
014        }
015        else
016          PostBTMessage(L"發生錯誤, 藍牙連結超時!");
017      }
```

```
018    }
```

上面的 FClientSocket 變數當然是宣告為 TBluetoothSocket 型態的物件變數：

```
   TBluetoothSocket *FClientSocket;
```

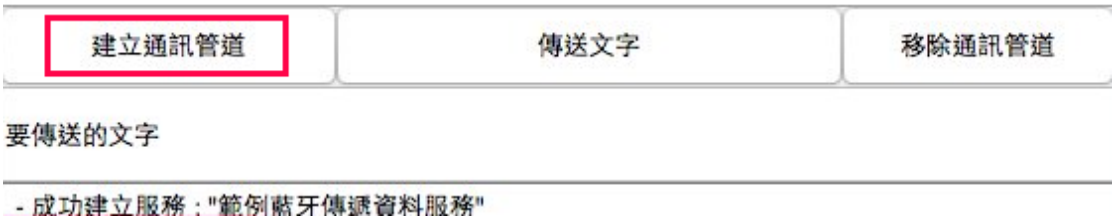
最後的 SendData()方法則非常的簡單，它使用剛才建立的 FClientSocket 物件呼叫它的 SendData()方法傳遞資料，由於 SendData()方法需要傳遞 TBytes 型態的資料，因此在 003 行需要藉由 TEncoding::UTF8->GetBytes() 方法把主表單中 TEdit 元件中的文字字串型態資料先轉換成 TBytes 型態的資料：

```
001    void TfmMainForm::SendData()
002    {
003      TBytes ToSend = TEncoding::UTF8->GetBytes(edtText->Text);
004      FClientSocket->SendData(ToSend);
005      PostBTMessage(L"訊息已傳送!");
006    }
```


到這裡我們也完成了客戶端的開發工作，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 做為藍牙客戶端，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 物件等待稍後客戶端連結：



接著在客戶端 M8 手機中執行範例 App，找到配對的 Mac 機器，再點選”傳送文字”按鈕建立客戶端 TBluetoothSocketh 物件，連結伺服端並傳遞資料給伺服端：

下面的 2 個畫面就是伺服端和客戶端的執行結果，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。

12 呼叫 Android 系統功能

在 Android 平台 C++Builder 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 C++Builder 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 C++Builder 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 C++Builder 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 C++Builder 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	C++Builder 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	C++Builder 使用來封裝 Java API 物件的內容，例如物件方法等
TJavaGenericImport 類別	C++Builder 使用來合成 JObjectClass 和 JObject 介面成為 C++Builder 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，C++Builder 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

由於在 RAD Studio 中 Android 的 SDK 大都是使用 C++Builder 程式語言定義的，因對於 C++Builder 的開發人員來說在瞭解了上面的內容後更需要知道如何藉由 C/C++來呼叫 Android 的 SDK。一般來說 C++Builder 開發人員可以藉由

```
_di_C++Builder 介面名稱
```

和

```
_di_C++Builder 類別名稱
```


來呼叫 Android 的 SDK，也許讓我們使用一個簡單的範例來說明就很容易瞭解了。

例如在下面即將說明的範例中我們需要呼叫 Android SDK 的 JContentResolver 介面中的方法，而 JContext 是由 DC++Builder 程式語言定義如下：

```
  [JavaSignature('android/content/ContentResolver')]
  JContentResolver = interface(JObject)
    ['{774C50C1-66DC-489E-9CAC-5434A5DE7CE0}']
    function acquireContentProviderClient(uri: Jnet_Uri):
JContentProviderClient; cdecl; overload;
    …
```

在 C++Buider 中對於 C++Builder 定義的 JContentResolver 介面，我們使用 typedef 重定義了 C++Builder 的介面變數：

```
__interface JContentResolver;
typedef System::C++BuilderInterface<JContentResolver>
_di_JContentResolver;
```

因此在 C++Builder 中我們只使用使用_di_JContentResolver 就可以呼叫 Android SDK 的 JContentResolver 介面中的方法。

同樣的對於 C++Builder 定義的 JContentResolverClass 類別介面：

```
  JContentResolverClass = interface(JObjectClass)
    ['{29F2ED97-64A0-435B-A79C-7B8F80E6659A}']
    {class} function _GetCURSOR_DIR_BASE_TYPE: JString; cdecl;
…
```

在 C++Builder 中也只需要在 JContentResolverClass 名稱之前加上"_di_"即可：

```
__interface JContentResolverClass;
typedef System::C++BuilderInterface<JContentResolverClass>
_di_JContentResolverClass;
```

如果您不知道_di_的意義，其實它就是代表"**C++Builder I**nterface"。

接著 C++Builder 便用 TJContentResolver 類別封裝這 2 個介面，這也意味 C++Builder 的開發人員可以使用 TJContentResolver 類別來取得這 2 個介面。例如要取得_di_JContentResolverClass，那我們可以使用如下的程式碼：


```
_di_JContentResolverClass  jcrc = TJContentResolver::JavaClass;
```

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存取 Android 手機中的連絡人資訊。

12-1 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 C++Builder 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 C++Builder 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 C++Builder 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);
    }
}
```


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 C/C++直接呼叫 Java 程式碼：

1. 使用 Java2OP 工具把 Java 類別宣告轉成 C++Builder 類別宣告
2. 使用 dccaarm.exe 編譯器和編譯器指令把 C++Builder 類別宣告直接轉成 C/C++的表頭宣告
3. 在 C++Builder 專案中 include 上面步驟產生的 C/C++的表頭宣告，編譯成.O 的檔案並直接連結到最後的 App 中
4. 使用 C++Builder 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 C++Builder 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 C++Builder 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 C++Builder 類別宣告：

```
Java2OP.exe -jar mylib.jar
```

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 C++Builder 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.
```

讓我們把這個 C++Builder 類別宣告儲存為 com.xe5.ClassTesting1.RTClassTesting1.pas。

步驟 2 把 C++Builder 類別宣告直接轉成 C/C++的表頭宣告

有了 Java 類別的 C++Builder 類別宣告之後接下來我們可使用 RAD Studio 中的 dccaarm.exe 這個編譯器直接把 C++Builder 類別宣告直接轉成 C/C++的表頭宣告，非常的簡單。

dccaarm.exe 有一個編譯器令 jphne 就可以幫助我們完成這個工作，因此我們可以簡單的使用下面的指令把上面的 com.xe5.ClassTesting1.RTClassTesting1.pas 直接轉成 C/C++的表頭宣告：

```
dccaarm -jphne  com.xe5.ClassTesting1.RTClassTesting1.pas
```

我們就可以立刻得到 com.xe5.ClassTesting1.RTClassTesting1.hpp 表頭檔：

```
001    // CodeGear C++Builder
002    // Copyright (c) 1995, 2015 by Embarcadero Technologies, Inc.
003    // All rights reserved
004
```

```
005   // (DO NOT EDIT: machine generated header)
'com.xe5.ClassTesting1.RTClassTesting1.pas' rev: 29.00 (Android)
006
007   #ifndef Com_Xe5_Classtesting1_Rtclasstesting1HPP
008   #define Com_Xe5_Classtesting1_Rtclasstesting1HPP
009
010   #pragma C++Builderheader begin
011   #pragma option push
012   #pragma option -w-     // All warnings off
013   #pragma option -Vx     // Zero-length empty class member
014   #pragma pack(push,8)
015   #include <System.hpp>
016   #include <SysInit.hpp>
017   #include <Androidapi.JNIBridge.hpp>
018   #include <Androidapi.JNI.JavaTypes.hpp>
019
020   //-- user supplied
-----------------------------------------------------------
021
022   namespace Com
023   {
024   namespace Xe5
025   {
026   namespace Classtesting1
027   {
028   namespace Rtclasstesting1
029   {
030   //-- forward type declarations
-----------------------------------------------
031   __interface JRTClassTesting1Class;
032   typedef System::C++BuilderInterface<JRTClassTesting1Class>
_di_JRTClassTesting1Class;
033   __interface JRTClassTesting1;
034   typedef System::C++BuilderInterface<JRTClassTesting1>
_di_JRTClassTesting1;
035   class C++BUILDERCLASS TJRTClassTesting1;
036   //-- type declarations
-------------------------------------------------------
```

```
037    __interface
INTERFACE_UUID("{8EF97555-32BC-4262-B17E-7A5157944E97}")
JRTClassTesting1Class  : public
Androidapi::Jni::Javatypes::JObjectClass
038    {
039        virtual int __cdecl getIntValue(void) = 0 ;
040        HIDESBASE virtual _di_JRTClassTesting1 __cdecl init(void)
= 0 ;
041        virtual void __cdecl setIntValue(int newvalue) = 0 ;
042        virtual void __cdecl
stunt(Androidapi::Jni::Javatypes::_di_JString s, int n) = 0 ;
043    };
044
045    __interface
INTERFACE_UUID("{983668BC-BD74-4142-8A1F-3DDA43F3E29F}")
JRTClassTesting1  : public Androidapi::Jni::Javatypes::JObject
046    {
047        virtual int __cdecl getIntValue(void) = 0 ;
048        virtual void __cdecl setIntValue(int newvalue) = 0 ;
049        virtual void __cdecl
stunt(Androidapi::Jni::Javatypes::_di_JString s, int n) = 0 ;
050    };
051
052    #pragma pack(push,4)
053    class PASCALIMPLEMENTATION TJRTClassTesting1 : public
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JRTClassTesting
1Class,_di_JRTClassTesting1>
054    {
055        typedef
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JRTClassTesting
1Class,_di_JRTClassTesting1>  inherited;
056
057    public:
058        /* TObject.Create */ inline __fastcall
TJRTClassTesting1(void) :
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JRTClassTesting
1Class,_di_JRTClassTesting1> () { }
059        /* TObject.Destroy */ inline __fastcall virtual
```

```
~TJRTClassTesting1(void) { }
060
061    };
062
063    #pragma pack(pop)
064
065    //-- var, const, procedure
-------------------------------------------------
066    }  /* namespace Rtclasstesting1 */
067    }  /* namespace Classtesting1 */
068    }  /* namespace Xe5 */
069    }  /* namespace Com */
070    #if !defined(C++BUILDERHEADER_NO_IMPLICIT_NAMESPACE_USE)
&& !defined(NO_USING_NAMESPACE_COM_XE5_CLASSTESTING1_RTCLASSTEST
ING1)
071    using namespace Com::Xe5::Classtesting1::Rtclasstesting1;
072    #endif
073    #if !defined(C++BUILDERHEADER_NO_IMPLICIT_NAMESPACE_USE)
&& !defined(NO_USING_NAMESPACE_COM_XE5_CLASSTESTING1)
074    using namespace Com::Xe5::Classtesting1;
075    #endif
076    #if !defined(C++BUILDERHEADER_NO_IMPLICIT_NAMESPACE_USE)
&& !defined(NO_USING_NAMESPACE_COM_XE5)
077    using namespace Com::Xe5;
078    #endif
079    #if !defined(C++BUILDERHEADER_NO_IMPLICIT_NAMESPACE_USE)
&& !defined(NO_USING_NAMESPACE_COM)
080    using namespace Com;
081    #endif
082    #pragma pack(pop)
083    #pragma option pop
084
085    #pragma C++Builderheader end.
086    //-- end unit
-----------------------------------------------------------------
087    #endif // Com_Xe5_Classtesting1_Rtclasstesting1
```

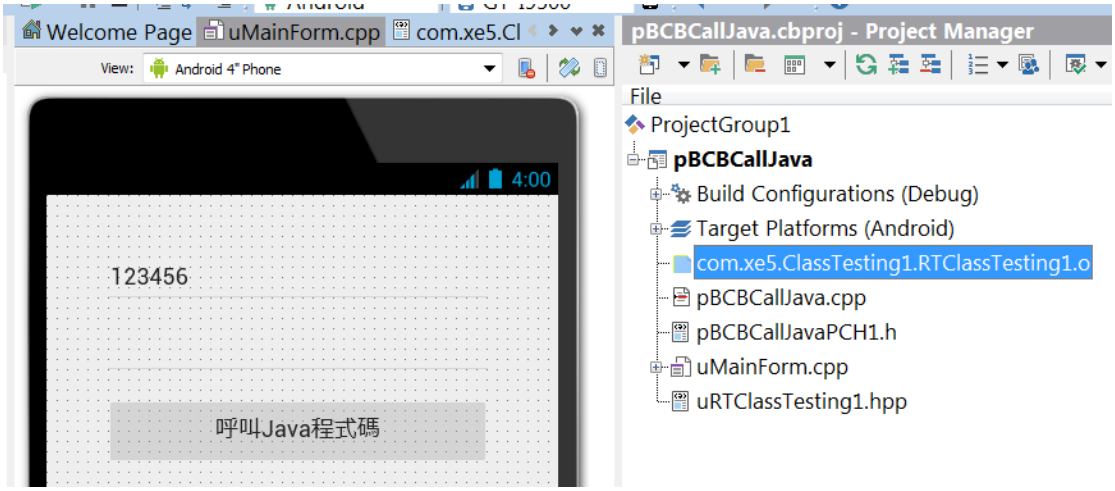

請花些時間仔細觀察一下上面表頭檔的內容，並且對照前面本書說明 C++Builder 和 C++Builder 如何封裝 Java 類別和 Android SDK 的規則。例

如在 032 行有一個封裝 Java 類別的_di_JRTClassTesting1Class，034 行有一個封裝 Java 物件的_di_JRTClassTesting1，最後在 058 行使用了 Androidapi::Jnibridge::TJavaGenericImport__2 封裝 JRTClassTesting1Class 和 JRTClassTesting1。整個表頭檔程式碼都符合本書前面說明的規則。

讓我們把這個 C/C++ 表頭宣告儲存為 com.xe5.ClassTesting1.RTClassTesting1.hpp。

步驟 3 編譯成.O 的檔案並直接連結到最後的 App 中

現在在 C++Builder IDE 中建立一個 Multi-Device Application 專案如下：



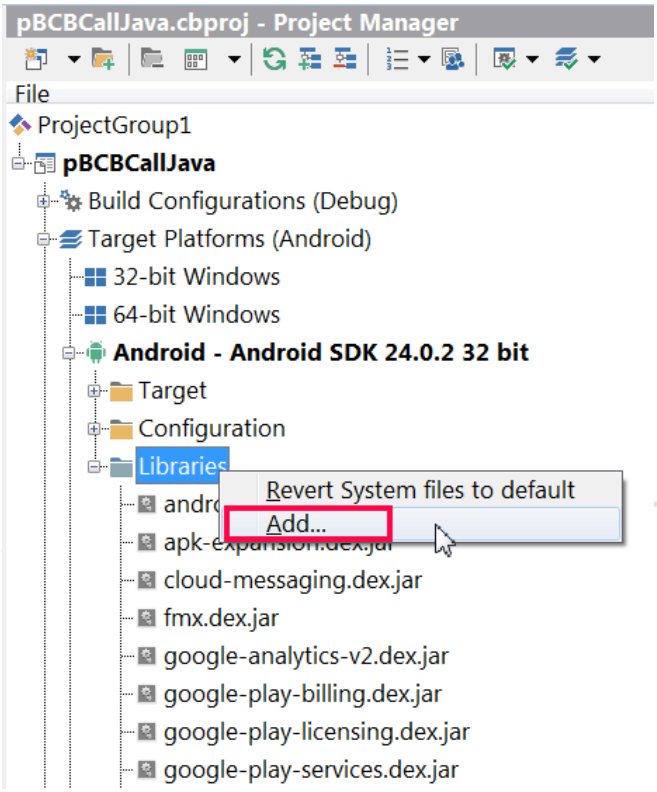
接著在主表單程式碼中只需要 include 在步驟 2 產生的 C/C++表頭宣告檔：

```
#include <fmx.h>
#pragma hdrstop

#include "uMainForm.h"
#include "com.xe5.ClassTesting1.RTClassTesting1.hpp"
…
```

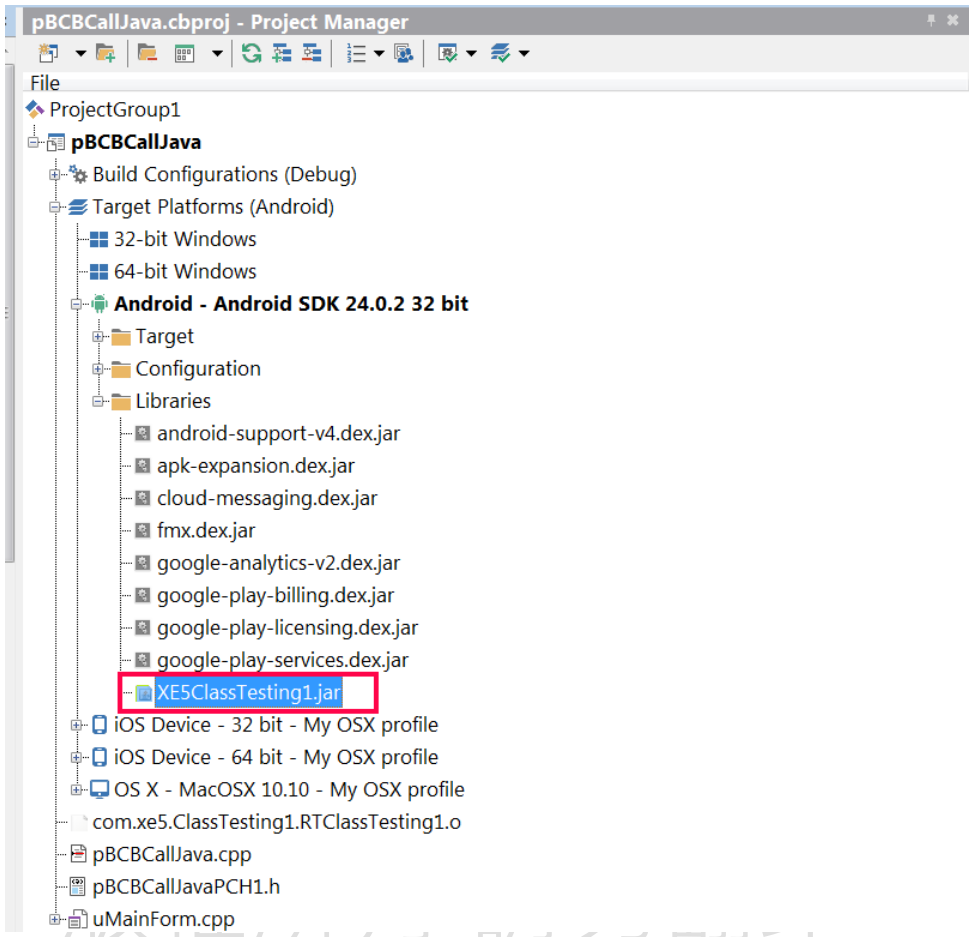
就可以得到.O 的檔案了。

步驟 4 加入 Java 的.jar 檔並部署

提供了可讓程式師在 C++Builder 使用的在 IDE 中 Java 及函式館中加入客製化的 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 平台，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：



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 平台中。

完成了上面的步驟後要讓 C/C++呼叫 Java 就簡單了，對於 C++Builder 來說就像是呼叫一般的 C/C++程式碼一樣。我們只需要使用如下的程式碼即可：

```
001    void __fastcall TForm5::Button1Click(TObject *Sender)
002    {
003      _di_JRTClassTesting1 ijt = TJRTClassTesting1::Create();
004      ijt->setIntValue(StrToInt(Edit1->Text));
005      Edit2->Text = IntToStr(ijt->getIntValue());
006      ijt = NULL;
007    }
```

在 003 行我們藉由呼叫 TJRTClassTesting1::Create() 建立 _di_JRTClassTesting1 型態的物件，而這也等於建立了 Java 的 RTWrapClassTest1 類別物件。接著在 004 和 005 行就可以藉由 ijt 直接呼叫

RTWrapClassTest1 類別物件的 setIntValue()和 getIntValue()方法了，就像直接呼叫由 C/C++語言寫的程式碼一樣。

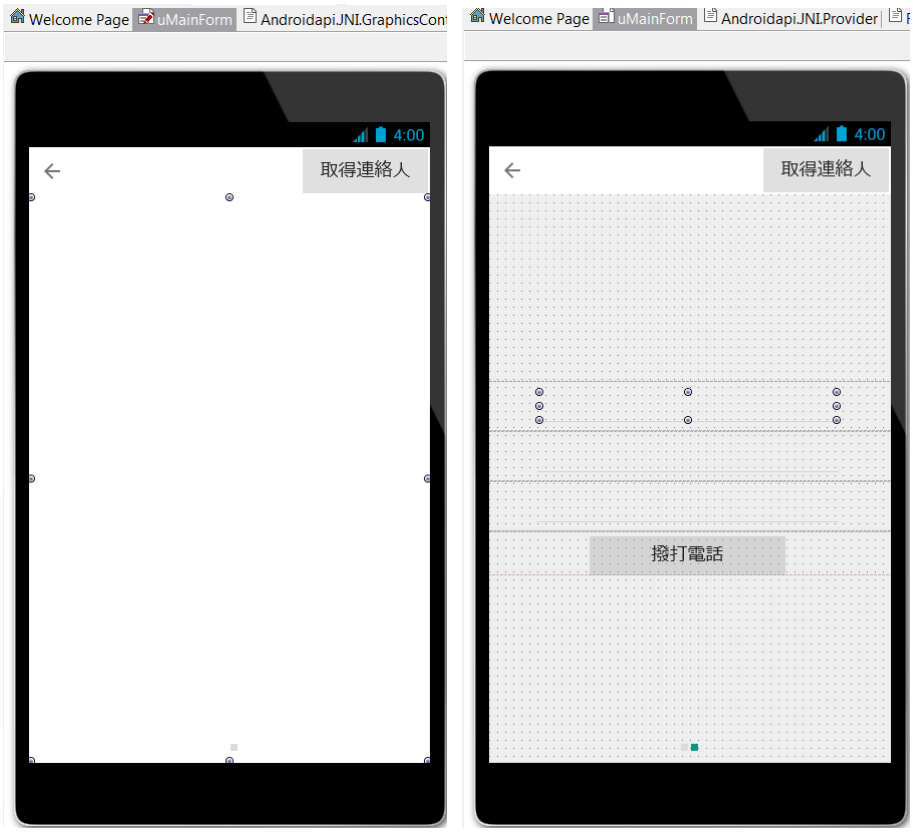
最後編譯並部署到 Android 手機中執行這個範例 App，從下面的執行畫面可以看到我們成功的使用 C++Builder 呼叫了由 Java 撰寫的程式碼。

12-2 呼叫 Java API 存取 Android 連絡人資訊

上面的範例是說明如何使用 C++Builder 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 C++Builder 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 C++Builder 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 C++Builder 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：



因此我們需要使用 C++Builder 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：

```
class TContactPerson
{
private:
  String FID;
  String FName;
  String FPhone;
  String FEMail;
public:
  __property String ID = {read=FID, write=FID};
  __property String Name = {read=FName, write=FName};
  __property String Phone = {read=FPhone, write=FPhone};
  __property String EMail = {read=FEMail, write=FEMail};
};
```

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：

```
void __fastcall TfmMainForm::btnGetContactsClick(TObject *Sender)
{
  GetContactsInfo();
}
```

GetContactsInfo()方法會呼叫GetContentResolver()，QueryContactInfo()和DisplayContactInfo()等3個方法，它們分別取得查詢連絡人的_di_JCursor介面，實際進行查詢資訊的工作以及顯示查詢的結果：

```
void TfmMainForm::GetContactsInfo()
{
  GetContentResolver();
  QueryContactInfo();
  DisplayContactInfo();
}
```


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：

```
  virtual _di_JCursor __cdecl
query(Androidapi::Jni::Net::_di_Jnet_Uri uri,
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> * projection,
Androidapi::Jni::Javatypes::_di_JString selection,
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> * selectionArgs,
Androidapi::Jni::Javatypes::_di_JString sortOrder) = 0 /* overload
*/;
  virtual _di_JCursor __cdecl
query(Androidapi::Jni::Net::_di_Jnet_Uri uri,
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> * projection,
Androidapi::Jni::Javatypes::_di_JString selection,
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> * selectionArgs,
```

```
Androidapi::Jni::Javatypes::_di_JString sortOrder,
Androidapi::Jni::Os::_di_JCancellationSignal cancellationSignal)
= 0 /* 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 C++Builder 定義的 SharedActivityContext() 公共方法取得 _di_JCursor 介面再呼叫 _di_JCursor 介面中的 getContentResolver() 方法即可取得代表 android.content.ContentResolver 的介面_di_JContentResolver。

```
void TfmMainForm::GetContentResolver()
{
  jcr = SharedActivityContext()->getContentResolver();
}
```

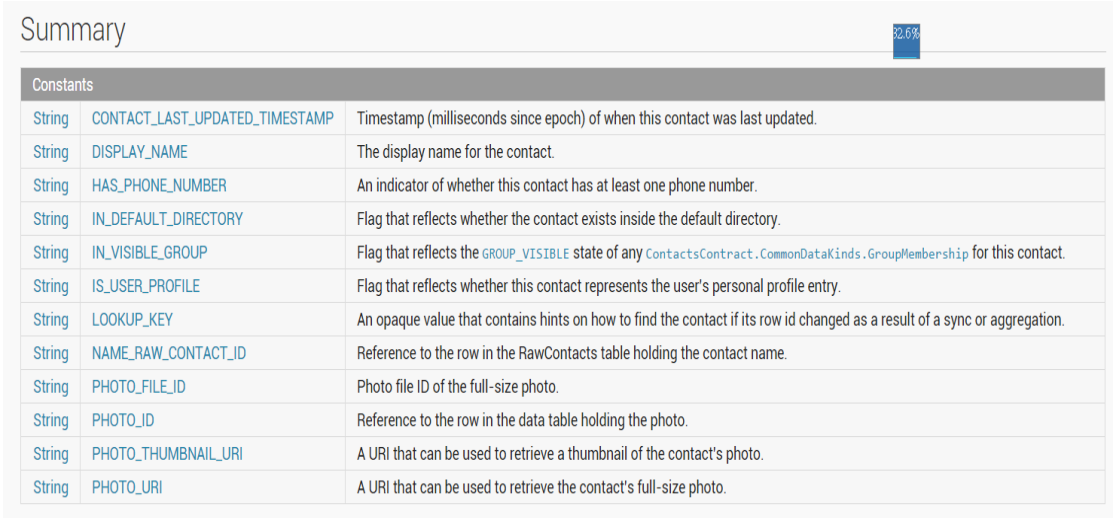

上面的 jcr 是宣告在表單類別 private 部份的_di_JContentResolver 介面變數：

```
private:  // User declarations
  _di_JContentResolver jcr;
  _di_JCursor jc;
  TList *contacts;
```

取得了_di_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

```
http://developer.android.com/reference/android/provider/Contacts
Contract.ContactsColumns.html
```

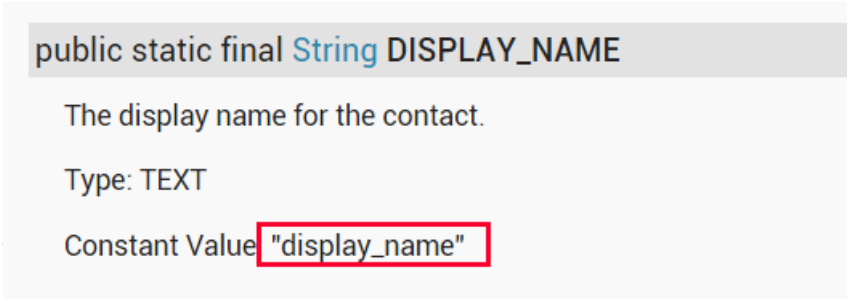
我們可以看到下面的資訊，其中的 DISPLAY_NAME 正是我們要查詢的連絡人名稱，而 LOOKUP_KEY 正是查詢鍵值：

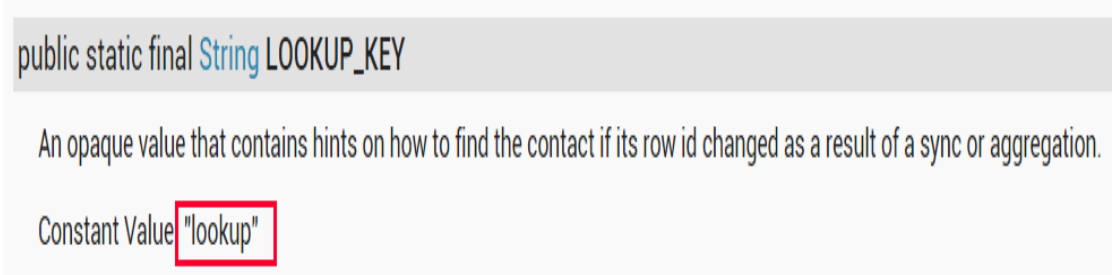


Summary

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”：





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。

瞭解了這些基本的觀念後就可以準備呼叫_di_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 文件連絡人是定義在 ContactsContract.Contacts 中，而 C++Builder 使用 TJContactsContract_Contacts 定義了這個類別(C++Builder 原始定義)：

```
  [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 C++Builder 的 Androidapi.Jni.Provider.hpp 表頭檔中定義如下：

```
__interface
INTERFACE_UUID("{3BA68A03-57B0-426D-B452-93635322BBDE}")
JContactsContract_ContactsClass  : public
Androidapi::Jni::Javatypes::JObjectClass
{
…
```


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：

```
__property Androidapi::Jni::Net::_di_Jnet_Uri CONTENT_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 NULL。Query()方法最後一個參數我們藉由呼叫 StringToJString()把 C++Builder 的字串轉成 Java 字串要求所有連絡人資訊以名稱排序：

```
void TfmMainForm::QueryContactInfo()
{
  TContactPerson *aContact;

  jc =
jcr->query(TJContactsContract_Contacts::JavaClass->CONTENT_URI,
NULL, NULL, NULL, StringToJString("display_name ASC"));
  jc->moveToFirst();
```

```
  while (jc->moveToNext())
  {
  aContact = new TContactPerson();
  aContact->ID = GetContactID();
  aContact->Name = GetContactName();
  aContact->Phone = GetContactPhone(aContact);
  aContact->EMail = GetContactEMail(aContact);
  contacts->Add(aContact);
  }
}
```

query()方法成功執行之後即會回傳_di_JCursor 介面(C++Builder 原始定義)：

```
  [JavaSignature('android/database/Cursor')]
  JCursor = interface(JCloseable)
```

在 C++Builder 中重新定義如下：

```
typedef System::C++BuilderInterface<JCursor> _di_JCursor;
class C++BUILDERCLASS TJCursor;
```

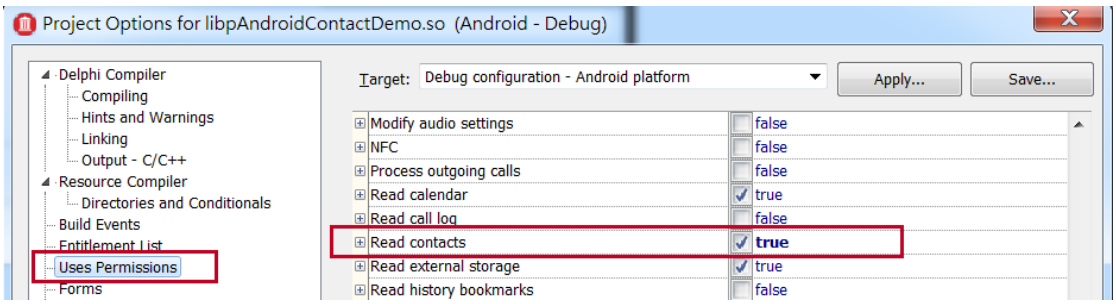

我們就可以使用_di_JCursor 介面中的方法再存取我們需要的資訊值，由於 query()方法回傳的類似一個資料表，因此我們先呼叫_di_JCursor 介面中的 moveToFirst()方法定位到第 1 筆資料，再進入迴圈藉由呼叫 moveToNext()方法依序的存取每一筆資料。

GetContactID()，GetContactName()方法就是使用回傳的_di_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 C++Builder 字串：

```
String TfmMainForm::GetContactID()
{
  return
UTF8String( JStringToString( jc->getString(jc->getColumnIndex(St
ringToJString("lookup"))) ) );
}
```

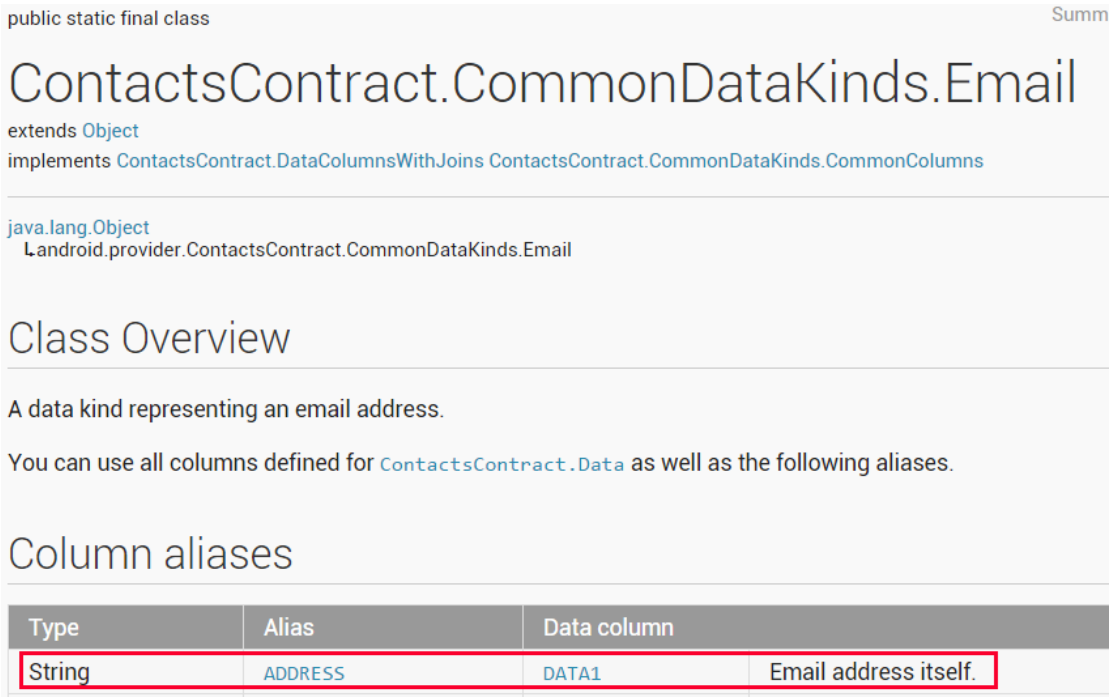
```
String TfmMainForm::GetContactName()
{
  return
UTF8String( JStringToString( jc->getString(jc->getColumnIndex(St
ringToJString("display_name"))) ) );
}
```

在試著執行此範例 App 之前您必須開啟 App 讀取連絡資訊的權限，請點選 Tools | Options...選單，在 User Permissions 項目中如下勾選設定 Read contacts 為 true：



完成上面的程式碼和權限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：



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 C++Builder 中 ContactsContract.CommonDataKinds.Email 類別已經由 TJCommonDataKinds_Email 定義了(C++Builder 原始定義)：

```
[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;
```

在 C++Builder 中重新定義如下：

```
__interface
INTERFACE_UUID("{E1AC9554-07BA-4399-8907-B92FA714B486}")
JCommonDataKinds_Email  : public
Androidapi::Jni::Javatypes::JObject
{

};

#pragma pack(push,4)
class PASCALIMPLEMENTATION TJCommonDataKinds_Email : public
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JCommonDataKind
s_EmailClass,_di_JCommonDataKinds_Email>
{
  typedef
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JCommonDataKind
s_EmailClass,_di_JCommonDataKinds_Email> inherited;

public:
  /* TObject.Create */ inline __fastcall
TJCommonDataKinds_Email(void) :
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JCommonDataKind
s_EmailClass,_di_JCommonDataKinds_Email> () { }
  /* TObject.Destroy */ inline __fastcall virtual
~TJCommonDataKinds_Email(void) { }

};

#pragma pack(pop)
```

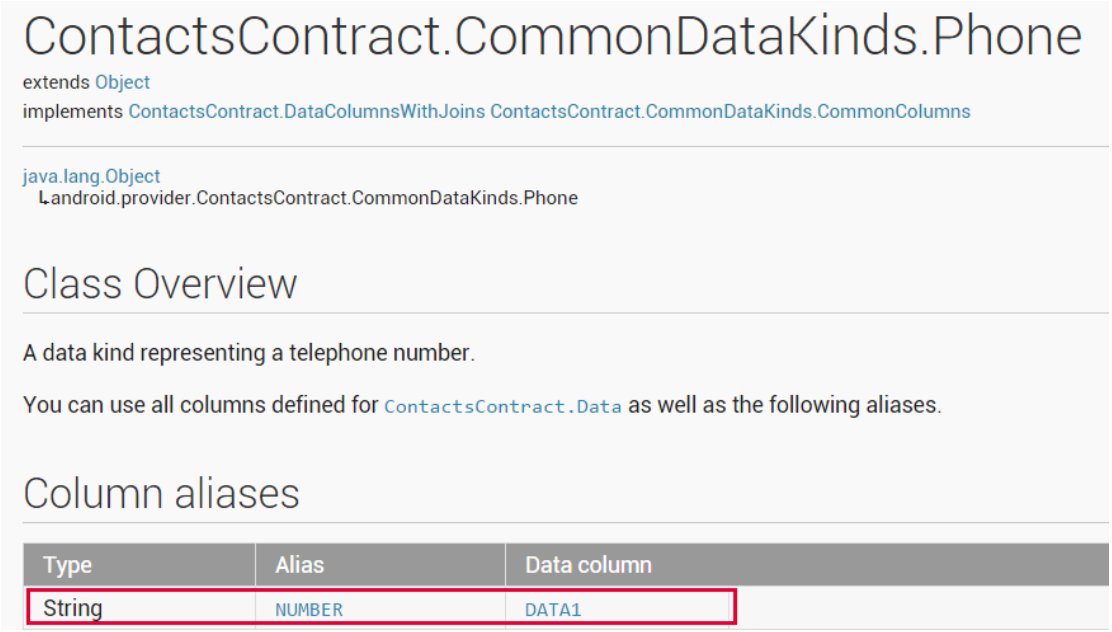

因此在下面的 GetContactEMail() 方法中 005~007 我們使用 Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Javatypes::_di_JString>泛型類別建立要查詢的 Email 地址欄位，008~010 建立查詢條件，013 代表要查詢 Email 資訊內容，010 則帶入先前已取得的此連絡人的查詢鍵值。012 行同樣取得代表查詢結果的 _di_JCursor 介面變數

emailCursor。015 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 地址：

```
001    String TfmMainForm::GetContactEMail(TContactPerson
*aContact)
002    {
003      String sResult = "";
004
005
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> *sProjection = new
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString>(1);
006      sProjection->Items[0] =
TJCommonDataKinds_Email::JavaClass->ADDRESS;
007
008
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> *sWhere = new
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString>(2);
009      sWhere->Items[0] =
TJCommonDataKinds_Email::JavaClass->CONTENT_ITEM_TYPE;
010      sWhere->Items[1] = StringToJString(aContact->ID);
011
012      _di_JCursor emailCursor =
jcr->query(TJContactsContract_Data::JavaClass->CONTENT_URI,sProj
ection, StringToJString("mimetype = ? AND lookup = ?"), sWhere,
NULL);
013      try
014      {
015        if (emailCursor->getCount() > 0)
016        {
017          emailCursor->moveToNext();
018          try
019          {
020            sResult =
UTF8String( JStringToString(emailCursor->getString(0)) );
```

```
021         }
022         catch(...)
023         {
024             ;
025         } }
026     }
027     __finally
028     {
029       emailCursor->close();
030       emailCursor = NULL;
031     }
032
033     return sResult;
034   }
```

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 C++Builder 的 TJCommonDataKinds_Phone 類別中(C++Builder 原始定義)：

```
[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;
```

在 C++Builder 中重新定義如下：

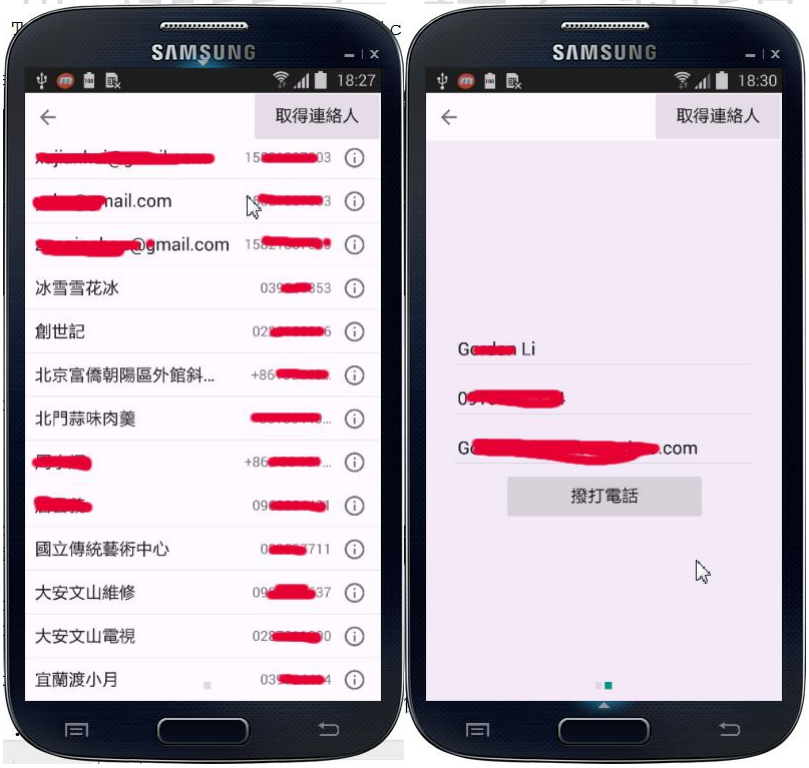
```
__interface
INTERFACE_UUID("{26C70162-6E79-45BB-9BE5-6C1EF293FD07}")
JCommonDataKinds_PhoneClass  : public
Androidapi::Jni::Javatypes::JObjectClass
{
…
}

#pragma pack(push,4)
class PASCALIMPLEMENTATION TJCommonDataKinds_Phone : public
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JCommonDataKind
s_PhoneClass,_di_JCommonDataKinds_Phone>
{
  typedef
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JCommonDataKind
s_PhoneClass,_di_JCommonDataKinds_Phone>  inherited;

public:
  /* TObject.Create */ inline __fastcall
TJCommonDataKinds_Phone(void) :
Androidapi::Jnibridge::TJavaGenericImport__2<_di_JCommonDataKind
s_PhoneClass,_di_JCommonDataKinds_Phone> () { }
  /* TObject.Destroy */ inline __fastcall virtual
~TJCommonDataKinds_Phone(void) { }

};

#pragma pack(pop)
```

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    String TfmMainForm::GetContactPhone(TContactPerson
*aContact)
002    {
003      String sResult = "";
004
005
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> *sProjection = new
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString>(1);
006      sProjection->Items[0] =
TJCommonDataKinds_Phone::JavaClass->NUMBER;
007
008
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString> *sWhere = new
Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Java
types::_di_JString>(2);
009      sWhere->Items[0] =
TJCommonDataKinds_Phone::JavaClass->CONTENT_ITEM_TYPE;
010      sWhere->Items[1] = StringToJString(aContact->ID);
011
012      _di_JCursor PhoneCursor =
jcr->query(TJContactsContract_Data::JavaClass->CONTENT_URI,sProj
ection, StringToJString("mimetype = ? AND lookup = ?"), sWhere,
NULL);
013      try
014      {
015        if (PhoneCursor->getColumnCount() > 0)
016        {
017          PhoneCursor->moveToNext();
018          try
019          {
020            sResult =
UTF8String( JStringToString(PhoneCursor->getString(0)) );
```

```
021         }
022         catch(...)
023         {
024             ;
025         }
026       }
027     }
028     __finally
029     {
030       PhoneCursor->close();
031       PhoneCursor = NULL;
032     }
033
034     return sResult;
035   }
```

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。

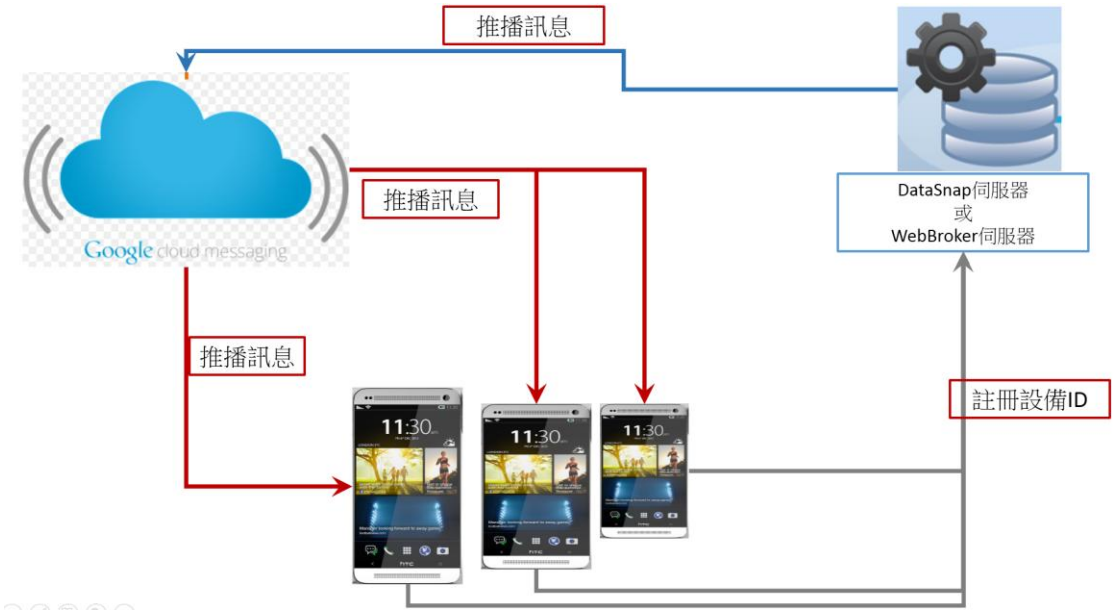


12-3　使用 Google GCM 實作推播功能

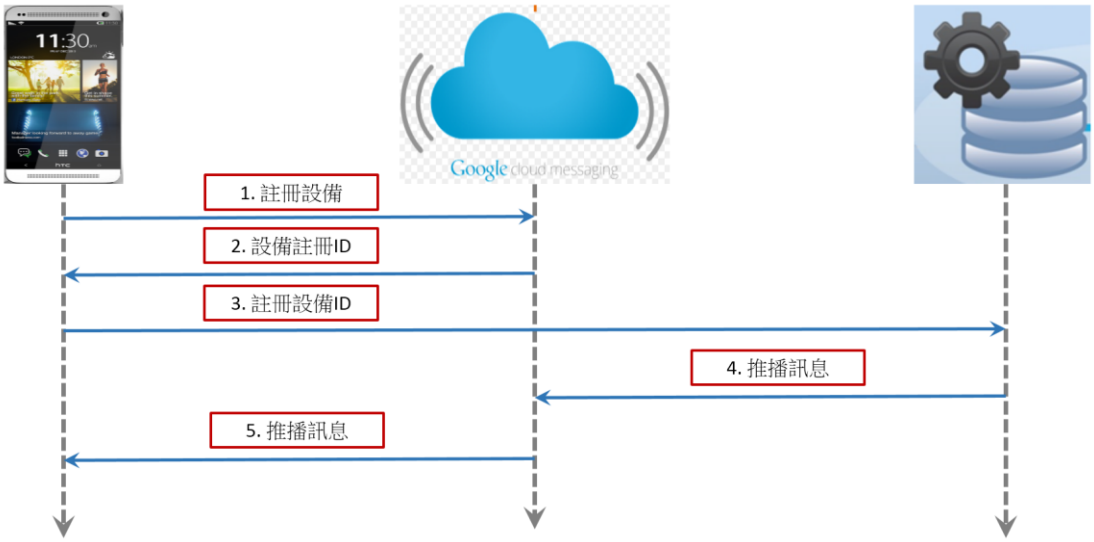
在最後一個小節中讓我們使用一個更複雜範例，那就是如何使用 C++Builder 實作推播的功能，這個範例不但需要呼叫 Java 程式碼，也需要從 Java 接收回傳通知。但在說明如何開發之前讓我們簡單的介紹一下 GCM 的架構。

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 個對象組成的，第 1 即是 GCM 本身，第 2 是客戶端的 Android 設備以及一個伺機器。在 C++Builder 技術領域中我們可以使用下圖來說明 GCM 的架構連作。

在 C++Builder 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 客戶端。至於下圖中的 Android 客戶端就是使用 C++Builder 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 客戶端。



下圖是這 3 者之間基本的互動流程圖，一開始 Android 客戶端 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 客戶端，那一群 Android 客戶端或是所有的 Android 客戶端。



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 客戶端 App
3. 開發 DataSnap 伺服器

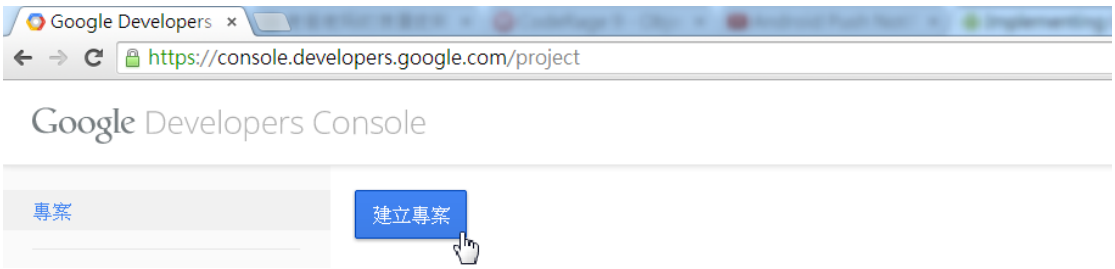
在下面的小節中將分別說明如何完成每一個步驟。

12-3-1 啟動使用 GCM 功能

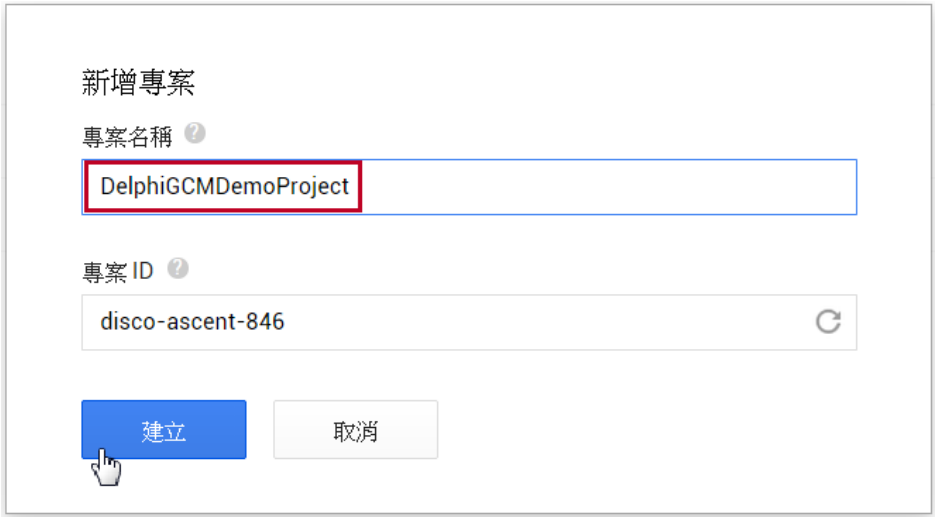
在使用 GCMi 之前您必須先到

```
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：



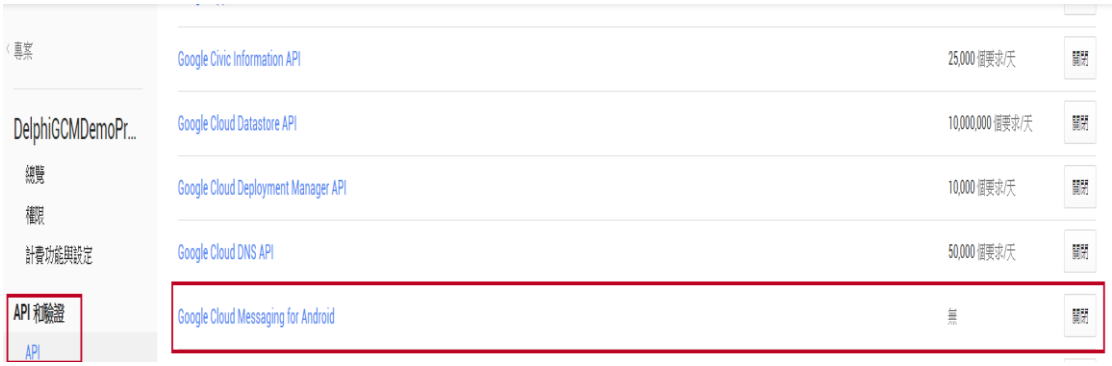
取一個專案名稱，例如”C++BuilderGCMDemoProject”，並點選”建立”按鈕：



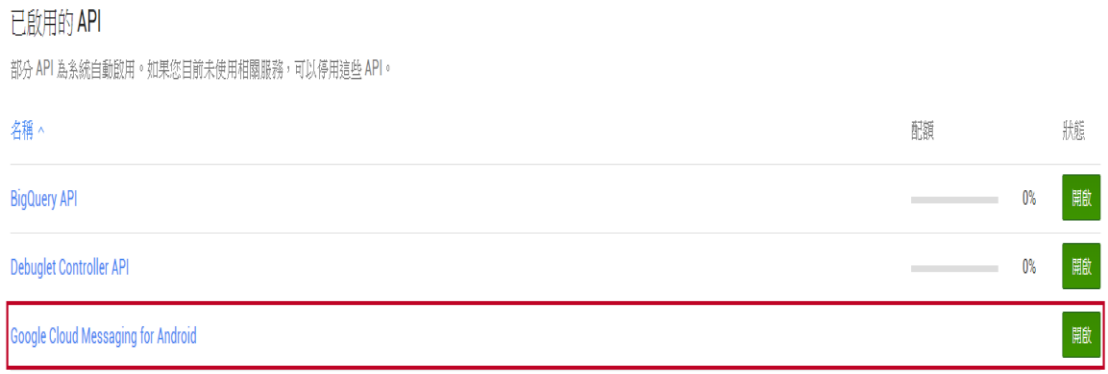
成功建立專案之後在”總覽”頁面可看到一個專案編號，此編號在稍後的開發中非常的重要請保存下來：

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 的客戶端設備可以使用，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：

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

12-3-2　開發 GCM 客戶端 App

在開發 GCM 架構中 GCM 客戶端是比較複雜的，因為 GCM 客戶端需要完成下列的數項工作：

1. 向 GCM 註冊客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

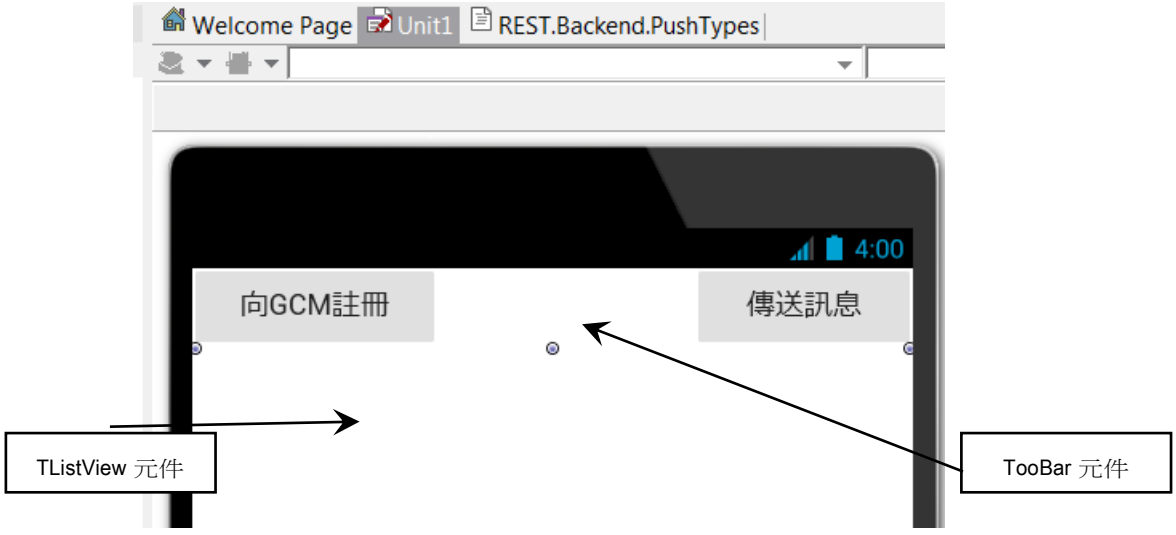
其中第 1 項工作需要呼叫 GCM API，這在前面的章節中已經說明如何呼叫 Java 程式碼所以不困難。第 1 項工作需要實作一個 DataSnap 伺服器並呼叫它的服務方法，這也不困難。最困難的是第 3 項工作，因為 GCM 會推播訊息通知給 Java 的回叫函式，我們必須把這個回叫的執行導引到我們的 C++Builder 程式碼中。但是記得 C++Builder 是一個 RAD 工具，因此只要我們瞭解了 GCM 的工作原理之後再想通如何在 C++Builder 中根據這些原理來開發，那麼就很簡單了，而且簡單到您會很吃驚。

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。

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 伺服器提供所有客戶端設備的功能，最後再開發一個 VCL 客戶端能夠推播訊息到客戶端設備。

在 C++Builder 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 註冊客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：

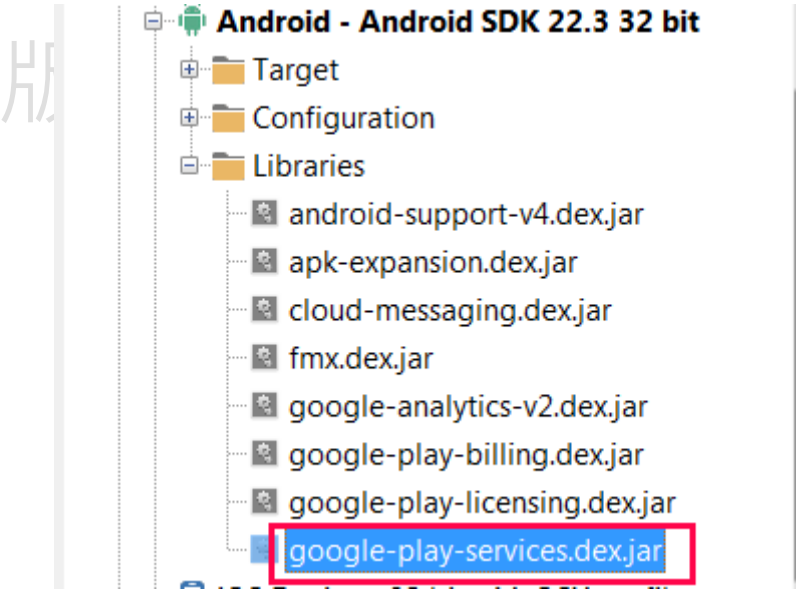
1. 查詢是否有 Google Play Service

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 C++Builder 中的 TJGooglePlayServicesUtil 類別，它的 JGooglePlayServicesUtilClass 介面中的 isGooglePlayServicesAvailable 類別方法可提供查詢：

```
virtual int __cdecl
isGooglePlayServicesAvailable(Androidapi::Jni::Graphicscontentvi
ewtext::_di_JContext context) = 0 ;
```

如果 isGooglePlayServicesAvailable 回傳 true，那應就可以開始註冊的工作，在這應該一定回傳 true，因為專案的 Libraries 中已經包含了提供 Google Play Service 的 jar 檔案了：

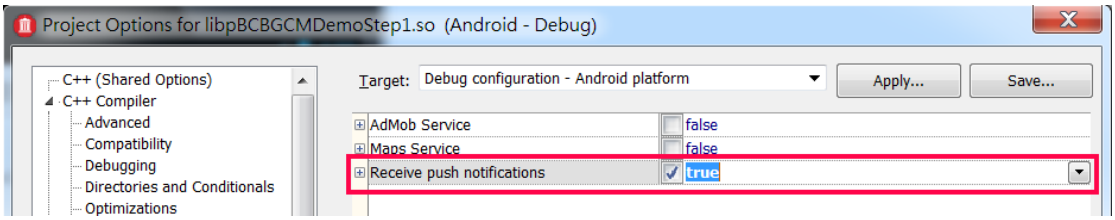


要向 GCM 註冊我們可以使用 TJGoogleCloudMessaging 類別中的 register 方法，它接受一個 Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::Jni::Javatypes::_di_JString>型態的參數：

```
__interface
INTERFACE_UUID("{A4F0B5B0-FCB2-45F8-A3CB-D37481F2FBD8}")
```

```
JGoogleCloudMessaging  : public
Androidapi::Jni::Javatypes::JObject
{
  virtual void __cdecl close(void) = 0 ;
  virtual Androidapi::Jni::Javatypes::_di_JString __cdecl
getMessageType(Androidapi::Jni::Graphicscontentviewtext::_di_JIn
tent intent) = 0 ;
  virtual Androidapi::Jni::Javatypes::_di_JString __cdecl
Register(Androidapi::Jnibridge::TJavaObjectArray__1<Androidapi::
Jni::Javatypes::_di_JString> * senderIds) = 0 ;
  virtual void __cdecl
send(Androidapi::Jni::Javatypes::_di_JString to_,
Androidapi::Jni::Javatypes::_di_JString msgId,
Androidapi::Jni::Os::_di_JBundle data) = 0 /* overload */;
  virtual void __cdecl
send(Androidapi::Jni::Javatypes::_di_JString to_,
Androidapi::Jni::Javatypes::_di_JString msgId, __int64 timeToLive,
Androidapi::Jni::Os::_di_JBundle data) = 0 /* overload */;
  virtual void __cdecl unregister(void) = 0 ;
};
```


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：

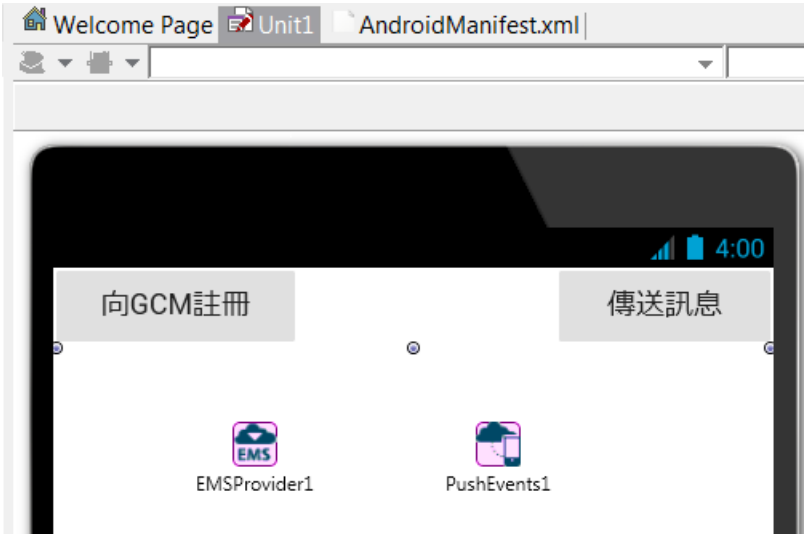


如此一來 C++Builder 就會在專案的 AndroidManifest.xml 檔中加入正確的權限和接收 GCM 訊息的接收器了。

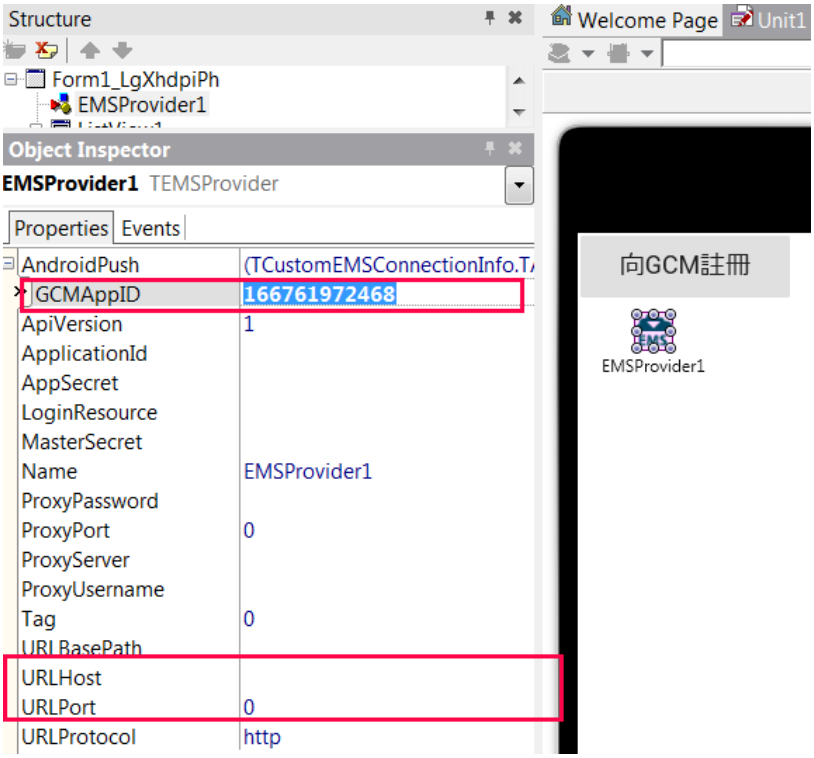
好了，那麼要如何前面的工作呢？非常簡單我們只需要轉個腦筋藉由使用中的 BAAS Client 元件就可以了。在 BAAS Client 元件組中有 TEMSProvider 和 TPushEvents 這 2 個元件，雖然 C++Builder 的英文手冊中說明 TEMSProvider 元件是連結到 EMS 伺服器使用的，而 TPushEvents 則是需要使用 Parse 或 Kinvey 等第 3 方供應商才能使用的，但不用管手冊說什麼，因為

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：

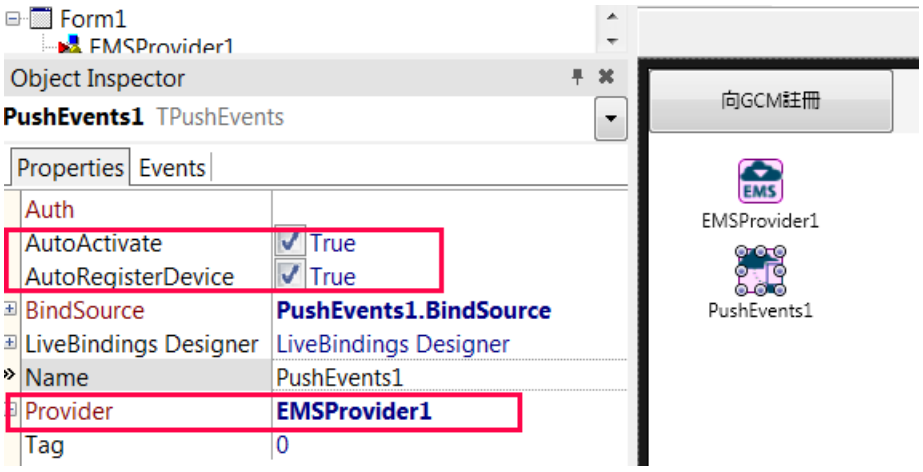


接下來要設定這 2 個元件，首先設定 TEMSProvider 元件。設定 TEMSProvider 元件的關鍵點是完全忽略 TEMSProvider 元件連結 EMS 伺服器的相關設定，我們只需要設定它的 AndroidPush 特性下的 GCMAppID 子特性，GCMAppID 的設定值就是我們前面說明的 Google 專案 ID 值，至於其他的特性，例如 MasterScret，AppSecret 等特性請都留成空白。這樣的設定等於讓 TEMSProvider 元件連結連結到 Google 的 GCM 服務伺服器而不是 EMS 伺服器，如下圖所示：



接著對 TPushEvents 元件進行如下的設定：

特性	設定值
AutoActivate	True
AutoRegisterDevice	True
Provider	EMSProvider1



TPushEvents 元件的 AutoRegisterDevice 特性就可以向 Google GCM 自動註冊客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 System::Pushnotification::TPushServiceConnection*
PushConnection = {read=GetPushServiceConnection};
```

另外還有下面 3 個關鍵特性：

```
__property bool DeviceRegistered = {read=GetDeviceRegistered,
nodefault};
__property System::UnicodeString DeviceToken =
{read=GetDeviceToken};
__property System::UnicodeString DeviceID = {read=GetDeviceID};
```

上面的 DeviceToken 特性值即是客戶端移動設備的 GCM 註冊 ID，而 DeviceID 特性值則是客戶端移動設備 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客戶端移動設備的 GCM 註冊 ID：

```
if (PushEvents->DeviceRegistered)
  sGCMID = PushEvents->DeviceToken;
```


在 PushConnection 中有一 Service 特性它是 TPushService 類別物件：

```
__property TPushService* Service = {read=FService};
```

當我們用 TEMSProvider 連結 Google GCM 時，這個 Service 其實會是 TGCMPushService 物件：

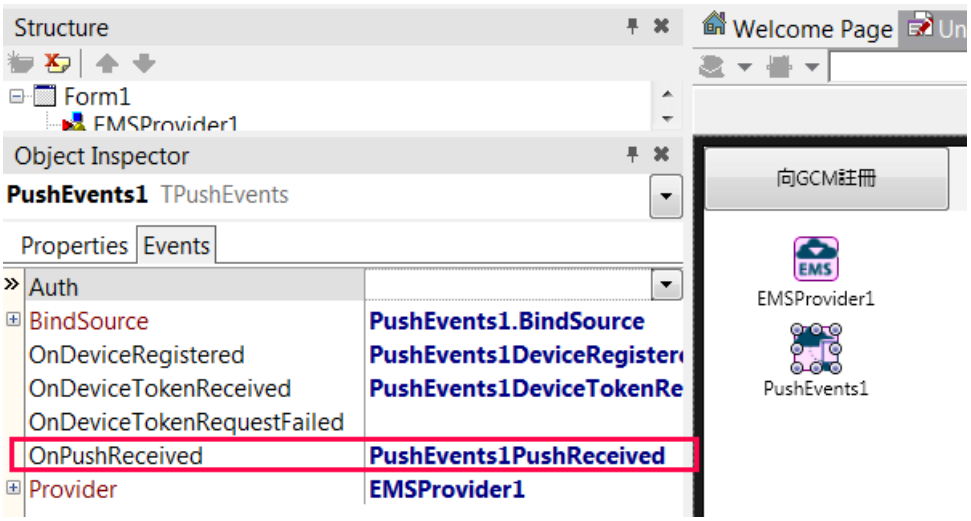
```
  TGCMPushService = class(TPushService)
```

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(以下為 C++Builder 的實作碼)：

```
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;
      GCMPushService.DoChange([TPushService.TChange.Status]);
    end;
  end;
end;
```


如何？很棒吧，只要腦筋轉一下就可以叫 C++Builder 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：



之後 C++Builder 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 一下你的專案即會自動產生)，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：

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：

```
void __fastcall TForm3::PushEvents1PushReceived(TObject *Sender,
TPushData * const AData)

{
  ListView1->Items->Add()->Text = AData->GCM->Title;
  ListView1->Items->Add()->Text = AData->GCM->Msg;
  ListView1->Items->Add()->Text = AData->GCM->Message;
}
```

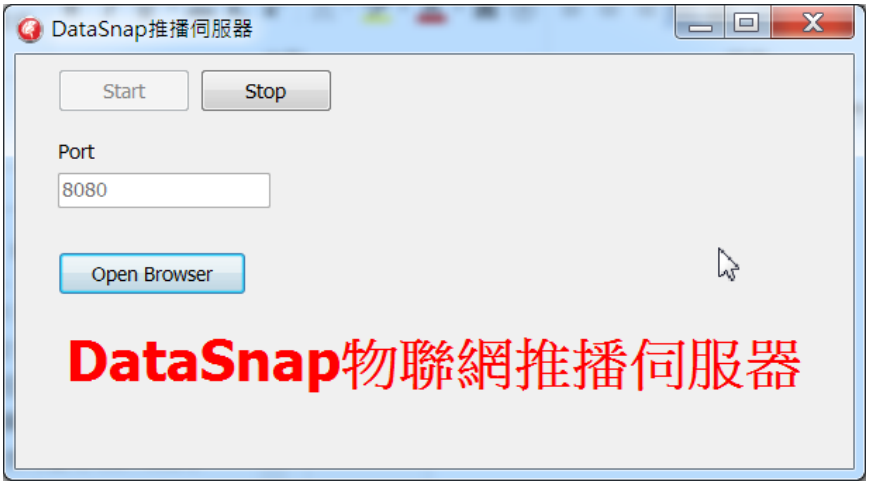
接著我們在主表單的”向 GCM 註冊”按鈕中存取設備 ID 和 GCM 註冊 ID：

```
void __fastcall TForm3::Button1Click(TObject *Sender)
{
  ListView1->Items->Add()->Text = PushEvents1->DeviceID;
  ListView1->Items->Add()->Text = PushEvents1->DeviceToken;
```

```
}
```

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，在這裡讓我們使用一個已經寫好的 DataSnap 推播伺服器：

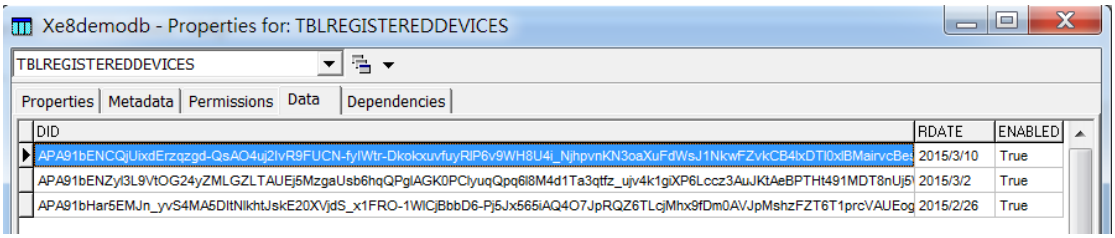


在這個 DataSnap 推播伺服器中提供了一個方法可以讓客戶端的 App 向它註冊客戶端手機的 ID，

```
bool __fastcall RegisterdeviceID(System::UnicodeString sID, const
String& ARequestFilter = String());
```

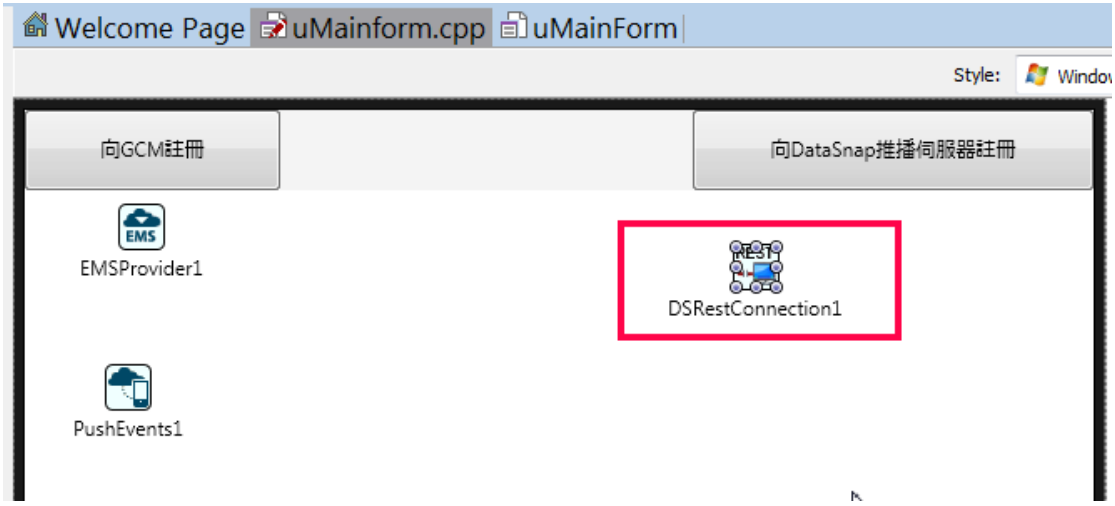
一旦客戶端的 App 呼叫 RegisterdeviceID 方法之後，DataSnap 推播伺服器就會在一個名為 TBLREGISTEREDDEVICES 資料表中註冊此客戶端手機的 ID，之後 DataSnap 推播伺服器就可以藉由這個 ID 向客戶端手機的 ID 推播資訊了。例如在下面的 TBLREGISTEREDDEVICES 資料表中目前註冊了 3

個手機 ID，稍後我們實作完成並呼叫 RegisterdeviceID 方法後在 TBLREGISTEREDDEVICES 資料表就會出現新的 BCB 客戶端 App 的 ID：

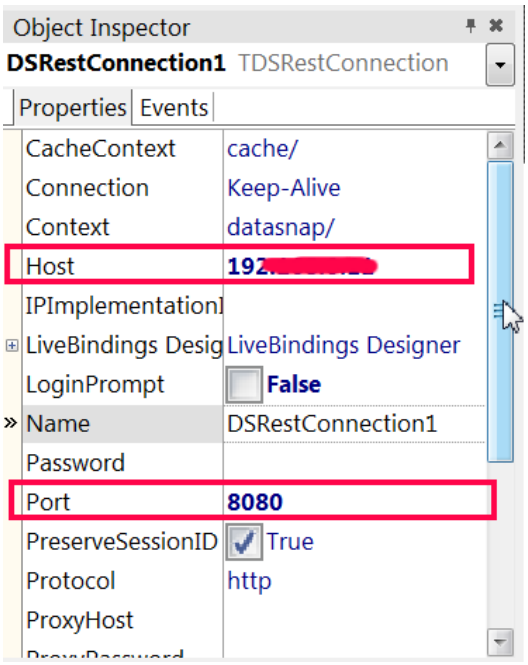


12-3-4 向 DataSnap 伺服器註冊設備 ID

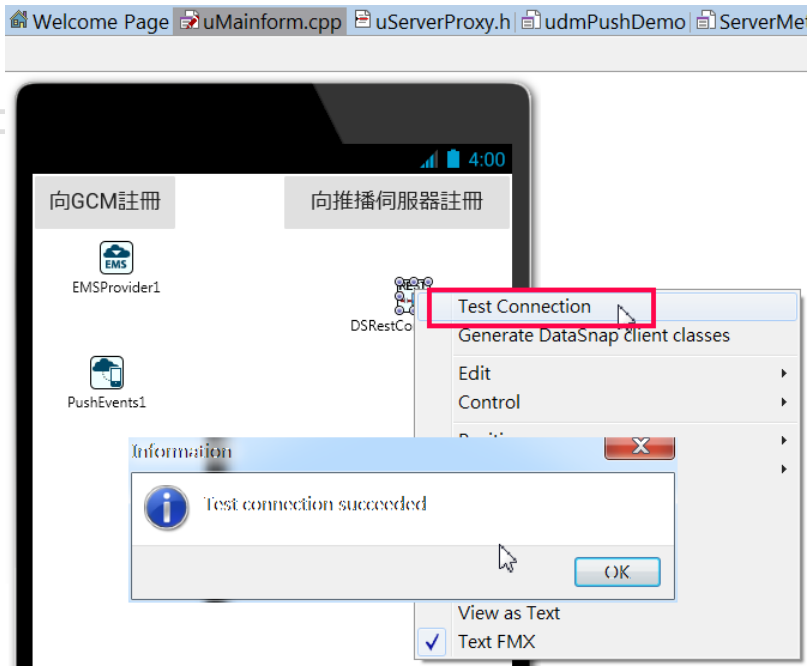
回到前面的範例 BCB App 準備加入向 DataSnap REST 推播伺服器註冊 GCM ID 的功能。首先要讓範例 App 連結範例 DataSnap REST 伺服器，請在主表單中加入 TDSRestConnection 元件：



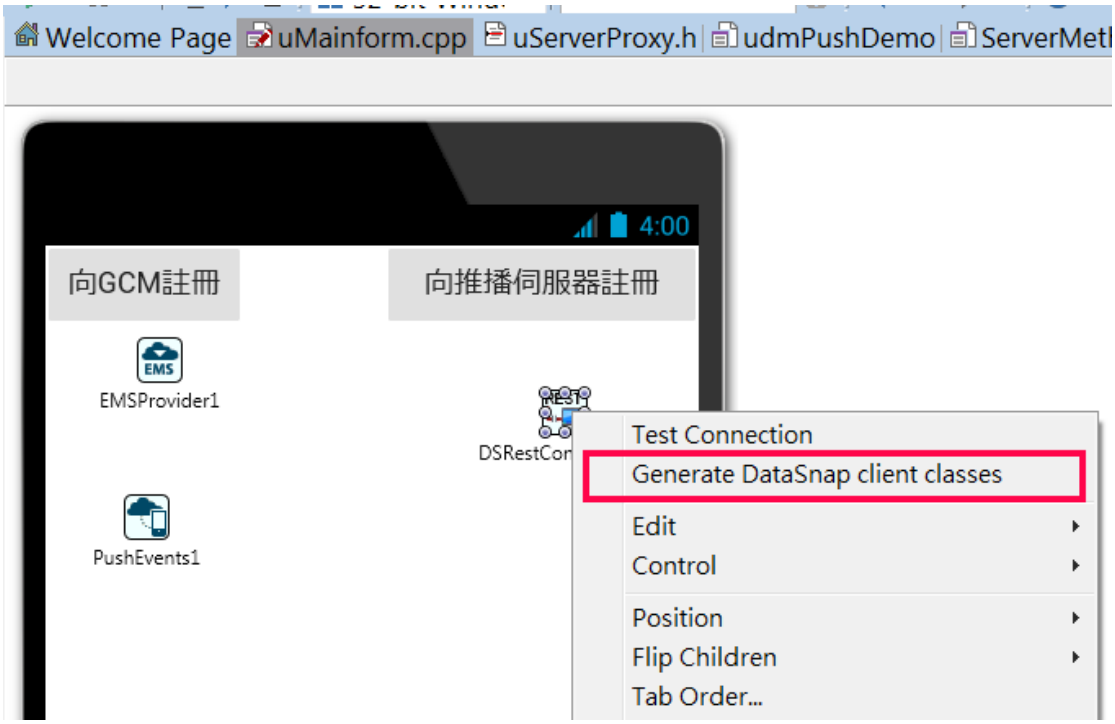
設定 DataSnap REST 伺服器的 IP 地址以及使用的 Port 特性值 8080：



再使用滑鼠右鍵測試是否成功連結到 DataSnap REST 推播伺服器：



再使用滑鼠右鍵產生客戶端連結程式碼：

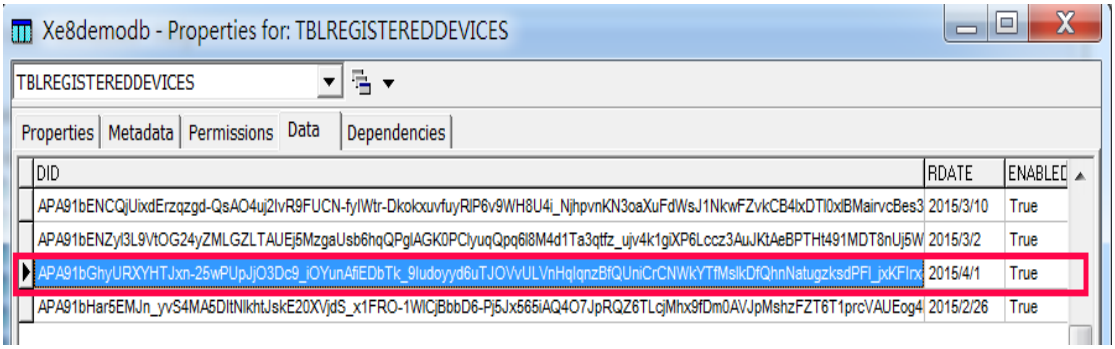


接著在"向推播伺服器註冊"按鈕中使用客戶端連結程式碼呼叫 DataSnap REST 推播伺服器提供的 RegisterdeviceID()方法註冊：

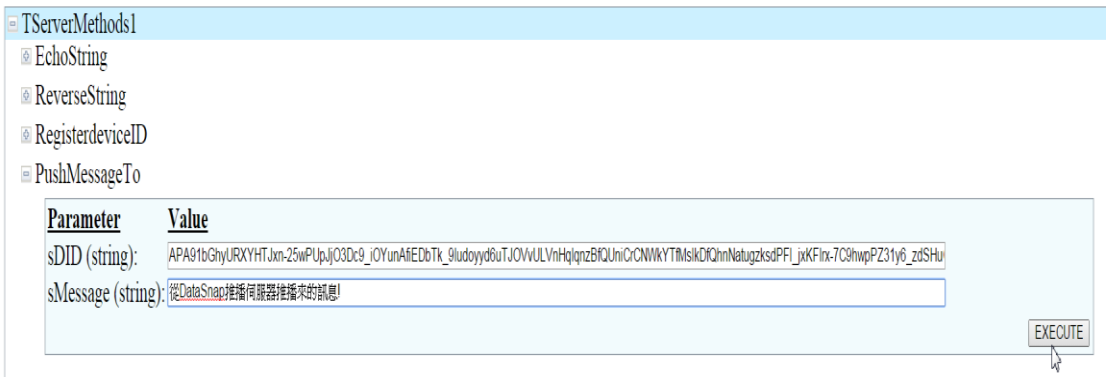

```
void __fastcall TForm3::Button2Click(TObject *Sender)
{
  TServerMethods1Client *pDSServer = new
TServerMethods1Client(DSRestConnection1);
  try
  {
  if (pDSServer->RegisterdeviceID(PushEvents1->DeviceToken))
    ListView1->Items->Add()->Text = L"向伺服器註冊成功";
  else
    ListView1->Items->Add()->Text = L"向伺服器註冊失敗";
  }
  __finally
  {
  delete pDSServer;
  }
}
```

執行範例 App 點選" 向推播伺服器註冊"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客戶端設備的 GCM ID 已經成功寫入了：

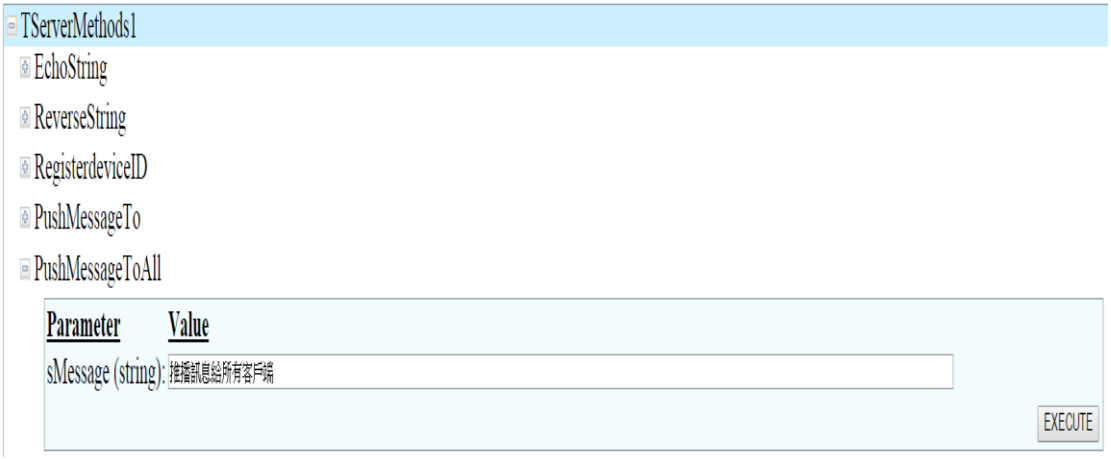


此時如果使用瀏覽器執行 DataSnap REST 推播伺服器的 PushMessageTo 方法並使用範例 BCB App 的客戶端 ID 值如下所示：



那就應該會在註冊的客戶端設備收到推播訊息，例如下面的畫面就是筆者的 Samsung S4 收到範例 DataSnap REST 推播伺服器藉由 GCM 推播來的訊息：

此時如果使用瀏覽器執行 PushMessageToAll 方法如下所示：



那就應該會在所有註冊的客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，一個 App 是使用 C++Builder 開發的，另一個是使用 C++Builder 開發的，但都能夠藉由 GCM 收到 DataSnap REST 伺服器推播來的訊息：


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

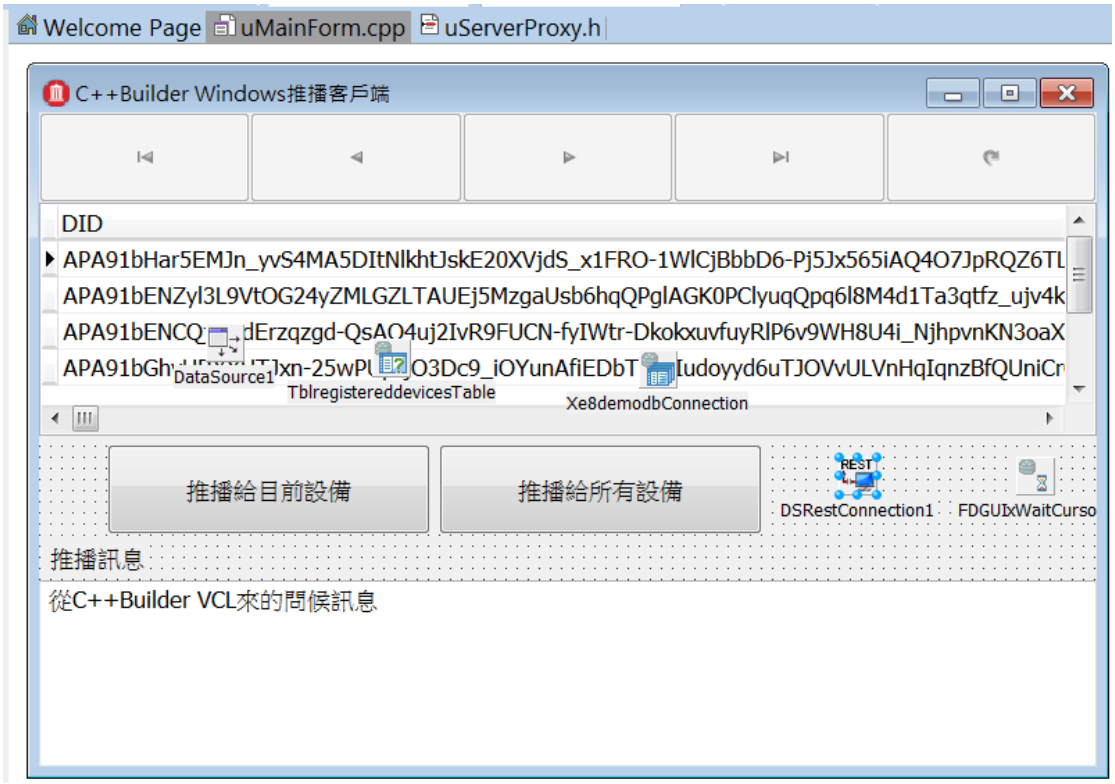

上午11:25
向GCM註冊
向伺服器註冊
傳送訊息
12EBDB10677ABD41CF9D2427AE3BA168
APA91bENZyl3L9VtOG24yZMLGZLTAUEj5MzgaUsb6hqQPglAG...
註冊成功
推播訊息給所有客戶端 -傳遞時間 : 15-03-02 11:23:01

12-3-5 開發 Windows 客戶端

要開發一個可推播訊息的 Windows 客戶端也很簡單，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 推播伺服器的服務方法即可完成。在 DataSnap 推播伺服器中的 PushMessageTo 方法可把訊息推播給特定的客戶端，而 PushMessageToAll 方法則可把訊息推播給所有的客戶端：

```
bool __fastcall PushMessageTo(System::UnicodeString sDID,
System::UnicodeString sMessage, const String& ARequestFilter =
String());
bool __fastcall PushMessageToAll(System::UnicodeString sMessage,
const String& ARequestFilter = String());
```

例如下面的 2 個畫面就是範例 Windows 客戶端推播訊息給手機的結果：



在”推播給目前設備”按鈕的 OnClick 事件中呼叫 PushMessageTo 方法推播訊息給特定的客戶端手機：

```
void __fastcall TfmMainForm::btnPushMessageClick(TObject *Sender)
{
  TServerMethods1Client *pDSServer = new
```

```
TServerMethods1Client(DSRestConnection1);
  try
  {
  pDSServer->PushMessageTo(GetCurrentID(),
mmMessages->Lines->Text);
  }
  __finally
  {
  delete pDSServer;
  }
}
//---------------------------------------------------------------

String TfmMainForm::GetCurrentID()
{
  return TblregistereddevicesTable->FieldByName("DID")->AsString;
}
```

在”推播給目前設備”按鈕的 OnClick 事件中呼叫 PushMessageToAll 方法推播訊息給所有的客戶端手機：


```
void __fastcall TfmMainForm::btnPushMessageToAllClick(TObject
*Sender)
{
  TServerMethods1Client *pDSServer = new
TServerMethods1Client(DSRestConnection1);
  try
  {
  pDSServer->PushMessageToAll(mmMessages->Lines->Text);
  }
  __finally
  {
  delete pDSServer;
  }
}
```

從下面的執行結果畫面可看到 C++Builder 的 VCL Windows 應用程式果然可藉由 DataSnap 推播伺服器推播訊息給 Android 的手機：

13 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客戶端設備之後，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 C++Builder 從版就開始提供強大的物聯網開發功能。

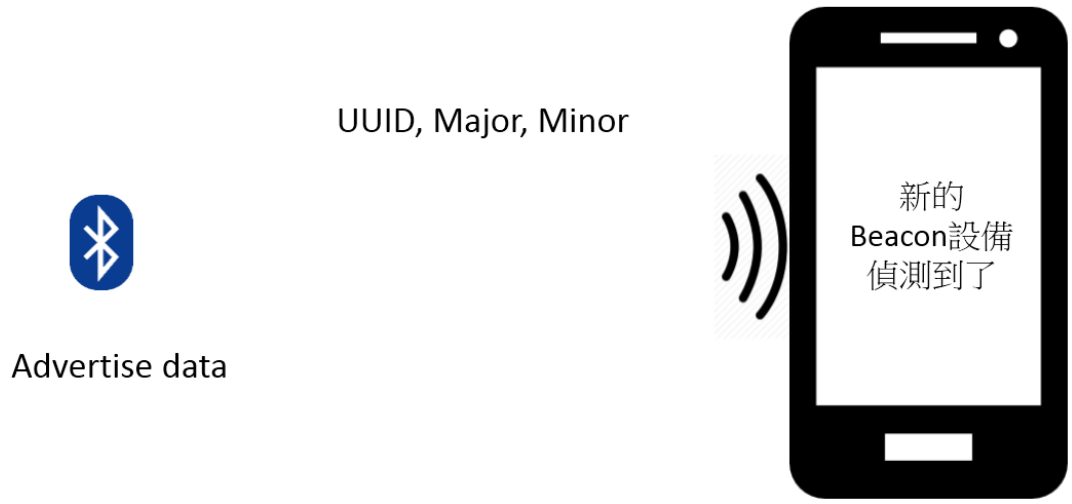
由於物聯網包含的設備眾多，在本小節中我們將討論如何使用 C++Builder 開發可使用 Beacon 設備的物聯網架構。

13-1 什麼是 Beancon 技術

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 設備的範圍內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 則可將定位範圍精準到 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。

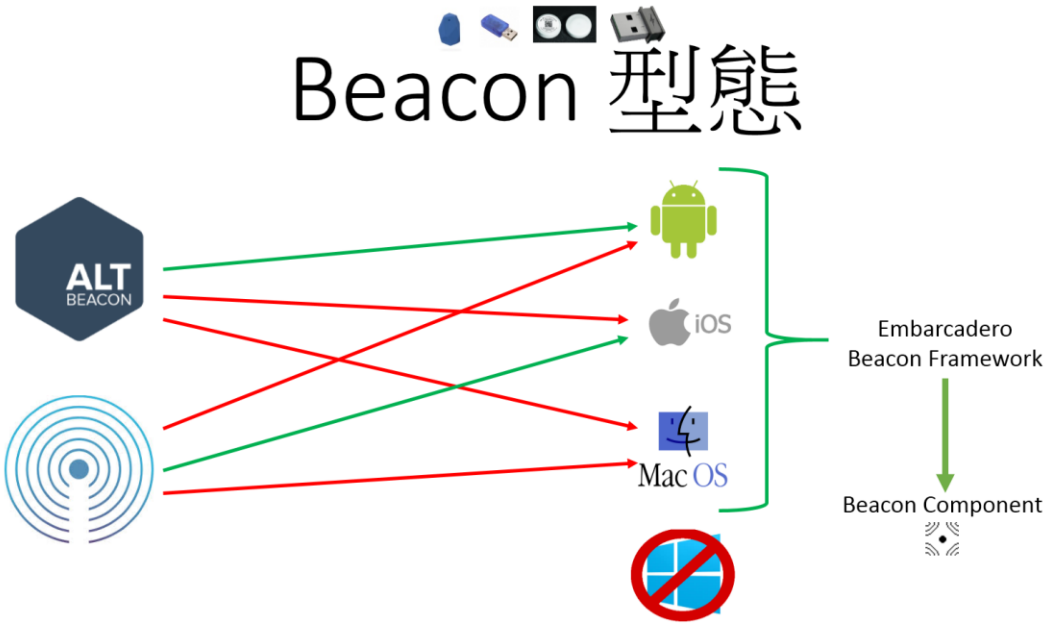
13-2 Beancon 種類

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/

C++Builder 在版開始同時支援 iBeacon 和 AltBeacon。

由於 Beacon 技術是根基於低功耗藍牙技術，因此不是每一個平台都可支援 iBeacon/AltBeacon，下面的圖形說明了目前在中每一個平台對於支援 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 平台尚不能開發 Beacon 的 App。

此外 C++Builder 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，因此提供了新的 Beacon 框架和 Beacon 元件來幫助 C++Builder 開發人員開發 iBeacon/AltBeacon 的 App。

13-3 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

C++Builder 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。

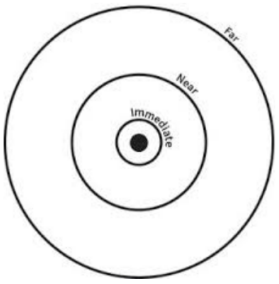
例如下面是 Beacon 設備可提供資料的範例，Beacon 設使用一個唯一的 UUID 來識別特定公司，用 Major ID 做為地區的識別，最後用 Minor ID 做為不同服裝部門的識別：

百貨地點		台北	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 C++Builder 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	資料長度	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

13-4 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)

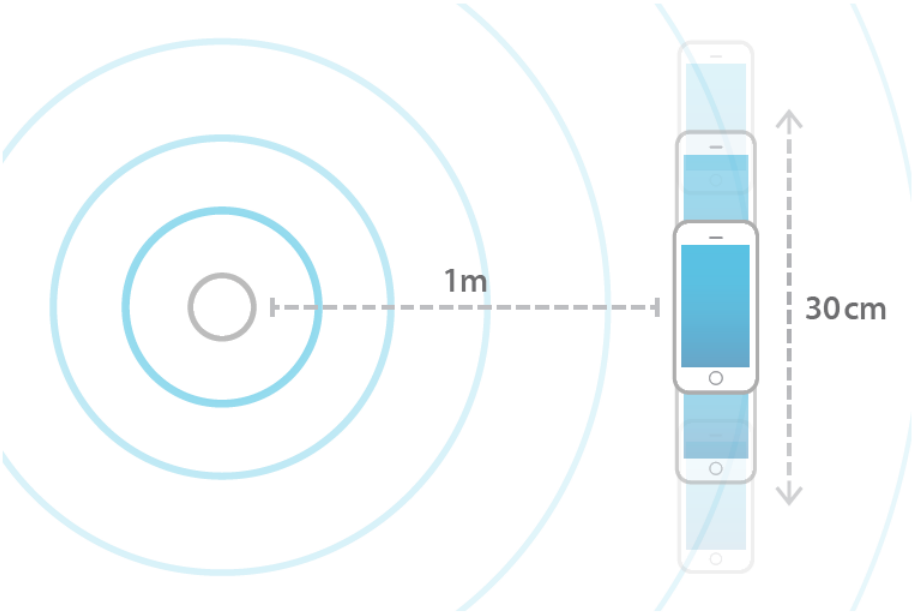
因此 App 可根據 Proximity 來決定要採取什麼行動。

13-5 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



13-6 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 C++Builder 來開發物聯網型態的 App 了。

在使用 C++Builder 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。

13-6-1 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。

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：

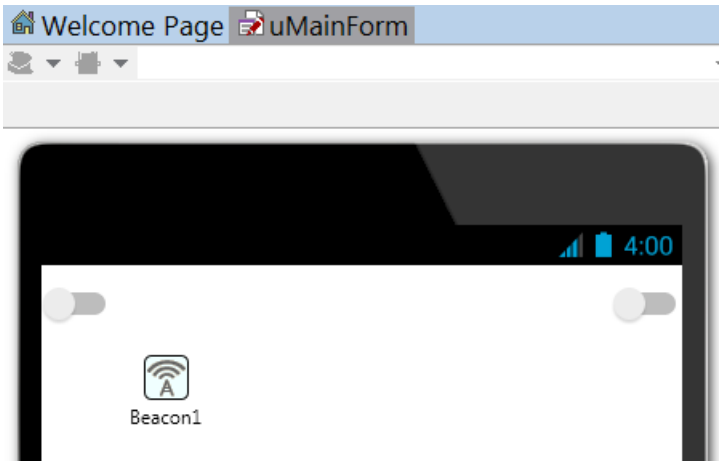
```
__property TDiscoveryLEEndEvent OnDiscoveryEnd =
{read=FOnDiscoveryLEEnd, write=FOnDiscoveryLEEnd};
__property TDiscoverLEDeviceEvent OnDiscoverLEDevice =
{read=FOnDiscoverLEDevice, write=FOnDiscoverLEDevice};
```


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：

```
bool __fastcall StartDiscovery(unsigned Timeout,
TBluetoothUUIDsList* const AFilterUUIDList =
(TBluetoothUUIDsList*)(0x0));
```

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 元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：

```
void __fastcall TfmMainForm::swtScanBeaconsSwitch(TObject *Sender)
{
  DoAutoScanBeacons();
}
```


接著在主表單表頭檔中宣告一個 TBluetoothLEManager 類別物件變數，以便使用它進行自動偵測 Beacon 設備：

```
    TBluetoothLEManager *FManager;
```

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：

```
void TfmMainForm::DoAutoScanBeacons()
{
  if (FManager == NULL)
  {
  FManager =  TBluetoothLEManager::Current;
  FManager->OnDiscoverLEDevice = DiscoverLEDevice;
```

```
  }
  FManager->StartDiscovery(10000);
}
```

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資料（$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）。在 System.Bluetooth.HPP 表頭檔中定義了 TScanResponseKey 資料型態，其中的 ManufacturerSpecificData 定義值是 255，也就是 16 進位的 FF：

```
enum class DECLSPEC_DENUM TScanResponseKey : unsigned short { Flags
= 1, IncompleteList16SCUUID,… ManufacturerSpecificData = 255 };
```

因此 DiscoverLEDevice()方法先檢查掃瞄到的低功耗藍牙設備是否包含 FF，如果是的話就建立一個 MyThreadProcedure 物件再呼叫 Synchronize() 方法來顯示資料：

```
void __fastcall TfmMainForm::DiscoverLEDevice(System::TObject*
const Sender, TBluetoothLEDevice* const ADevice, int Rssi,
  TScanResponse* const ScanResponse)
{
  if
(ScanResponse->ContainsKey(TScanResponseKey::ManufacturerSpecifi
cData))
  {
      _di_TThreadProcedure mtp = new MyThreadProcedure(ADevice,
Rssi, ScanResponse);
      System::Classes::TThread::Synchronize(NULL, mtp);
  }
}
```


但為什麼要建立一個 MyThreadProcedure 物件？為什麼要呼叫 Synchronize()方法來顯示資料呢？

這是因為在 C++Builder 中掃瞄低功耗藍牙設備要使用額外獨立的執行緒來進行，由於此獨立的執行緒不是主執行緒，因此在立的執行緒掃瞄完成要顯示資料時就需要呼叫 Synchronize()方法來處理同步的問題。

為了維護範例 App 中掃瞄到的低功耗藍牙設備，讓我們先定義一個 struct 資料結構來儲存掃瞄到的低功耗藍牙設備：

```
struct TBeaconDevice{
  TBluetoothLEDevice* ADevice;
TGUID GUID;
Word Major;
Word Minor;
Integer TxPower;
Integer Rssi;
Double Distance;
System::Boolean Alt;
};
typedef std::list<TBeaconDevice> TBeaconDeviceList;
```

接著定義 MyThreadProcedure 類別，請注意 MyThreadProcedure 類別是從 TCppInterfacedObject 類別繼承下來：

```
class MyThreadProcedure : public
TCppInterfacedObject<TThreadProcedure>
{
public:
  MyThreadProcedure(TBluetoothLEDevice* const _ADevice, int _Rssi,
TScanResponse* const _ScanResponse);
  void __fastcall Invoke(void);
private:
  TBluetoothLEDevice* const ADevice;
  int Rssi;
  TScanResponse* const ScanResponse;
};
```


因此我們需要一個介面讓 TCppInterfacedObject 封裝，這個介面就是 TThreadProcedure，它定義了一個介面方法 Invoke()：

```
__interface TThreadProcedure  : public System::IInterface
{
  virtual void __fastcall Invoke(void) = 0 ;
};
```

接下來我們需要做的就是實作 Invoke()方法的工作就是解析掃瞄到的 Becoan 設定資料：

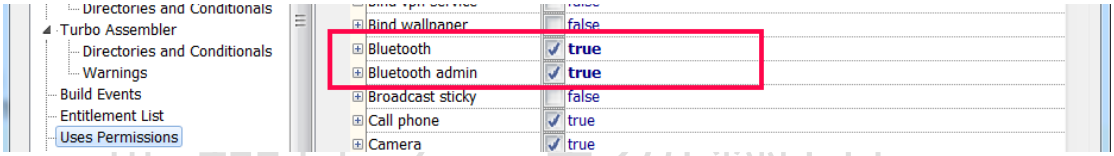
```
void __fastcall MyThreadProcedure::Invoke(void)
{
  TBeaconDevice LBeaconDevice =
fmMainForm->DecodeScanResponse(ScanResponse);
  LBeaconDevice.Rssi = Rssi;
  LBeaconDevice.Distance =
fmMainForm->FManager->RssiToDistance(Rssi, LBeaconDevice.TxPower,
0.5);
  LBeaconDevice.ADevice = ADevice;
  int NewBeacon = 0;
  bool BeaconFound = False;
  if (fmMainForm->BeaconDeviceList.size() > 0)
  {
      for (std::list<TBeaconDevice>::iterator
BD=fmMainForm->BeaconDeviceList.begin(); BD !=
fmMainForm->BeaconDeviceList.end(); ++BD)
      {
          NewBeacon++;
          if (IsEqualGUID(BD->GUID, LBeaconDevice.GUID) &&
(BD->Major == LBeaconDevice.Major) && (BD->Minor ==
LBeaconDevice.Minor))
          {
              *BD = LBeaconDevice;
              BeaconFound = True;
              break;
          }
       }
  }
  TVarRec v[] = { LBeaconDevice.Distance };
  if (!BeaconFound)
  {
      fmMainForm->BeaconDeviceList.push_back(LBeaconDevice);

  fmMainForm->lbBeacons->Items->Add("----------------------------
--");
```

```
      String BeaconName("Beacon Found: "+ ADevice->DeviceName);
      if (LBeaconDevice.Alt)
        BeaconName = BeaconName + "; AltB";
      else
        BeaconName = BeaconName + "; iB";
      fmMainForm->lbBeacons->Items->Add(BeaconName);

      BeaconName = "Device Complete name: ";
      if
(ScanResponse->ContainsKey(TScanResponseKey::CompleteLocalName))
      {
          TByteDynArray value;

  ScanResponse->TryGetValue(TScanResponseKey::CompleteLocalName,
value);
          BeaconName = BeaconName +
(TEncoding::UTF8->GetString(value));
      }
      else
        BeaconName = BeaconName + "No Name";

      fmMainForm->lbBeacons->Items->Add(BeaconName);
      fmMainForm->lbBeacons->Items->Add( " UUID: " +
GUIDToString(LBeaconDevice.GUID) );
      fmMainForm->lbBeacons->Items->Add( " Major:" +
IntToStr(LBeaconDevice.Major) +
                                    ",  Minor:" +
IntToStr(LBeaconDevice.Minor) +
                                    ",  txPower: " +
IntToStr(LBeaconDevice.TxPower) );
      fmMainForm->lbBeacons->Items->Add(" Rssi: " +
IntToStr(LBeaconDevice.Rssi) + Format(" Distance: %f m", v, 0) );
  }
  else
  {
      String BeaconName("Beacon Found: "+ ADevice->DeviceName);
      if (LBeaconDevice.Alt)
```

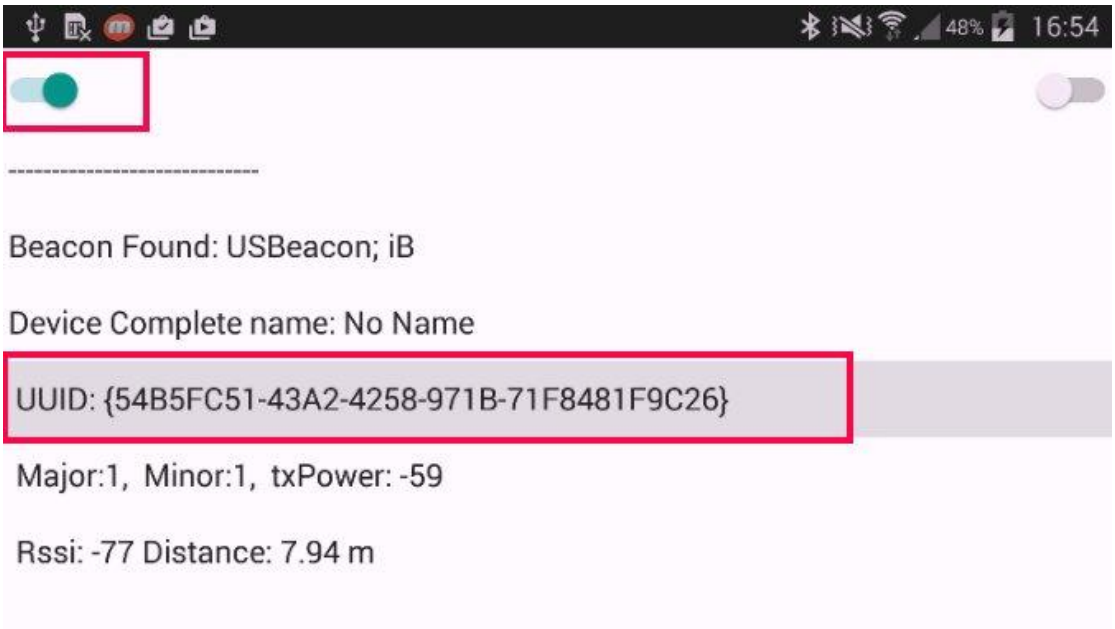
```
        BeaconName = BeaconName + "; AltB";
      else
        BeaconName = BeaconName + "; iB";
      fmMainForm->lbBeacons->Items->Strings[(NewBeacon-1)*6+1] =
(BeaconName);
      fmMainForm->lbBeacons->Items->Strings[(NewBeacon-1)*6+5] =
(" Rssi: " +  IntToStr(LBeaconDevice.Rssi) + Format(" Distance: %f
m", v, 0));
  }
}
```

由於 MyThreadProcedure 物件被封裝在 Synchronize()方法中，因此它可以放心的更新主表單中的 UI。

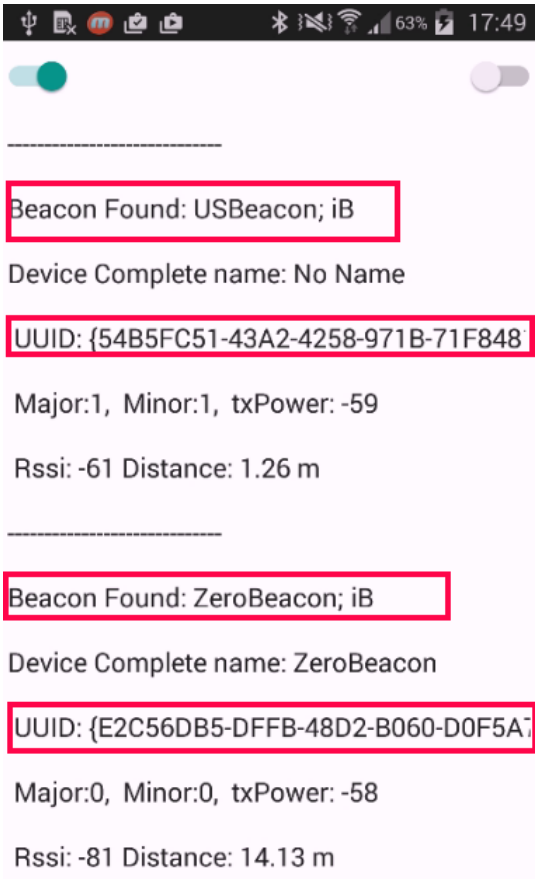
在執行範例 App 之前當然不要忘記開啟藍牙的存取權：



現在如果編譯和執行範例 App 就可以看到如下的畫面：



下圖是此範例 App 掃瞄到多個 Beacon 設備的畫面：



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。

13-6-2 開發已知 Beacon 設備 App

對於企業應用而言，在佈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，在 System.Beacon.Hpp 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(此程式碼是以 C++Builder 實作的)：

```
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 物件
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：

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)

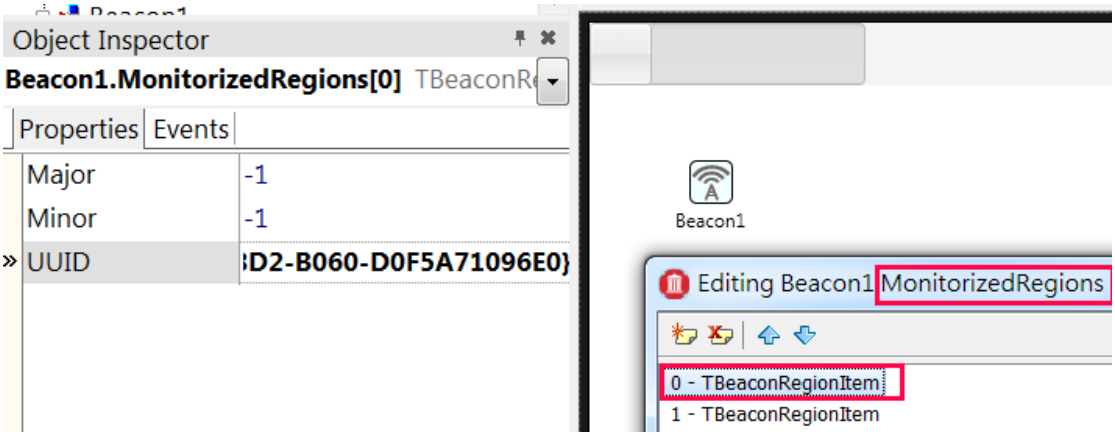
為了進一步簡化開發人員的工作，封裝了 TBeacon 元件,開發人員只需要使用 TBeacon 元件就可以更簡單的完成上面的工作。TBeacon 元件提供了下面的特性設定：

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 設備的製造商資料時觸發

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：



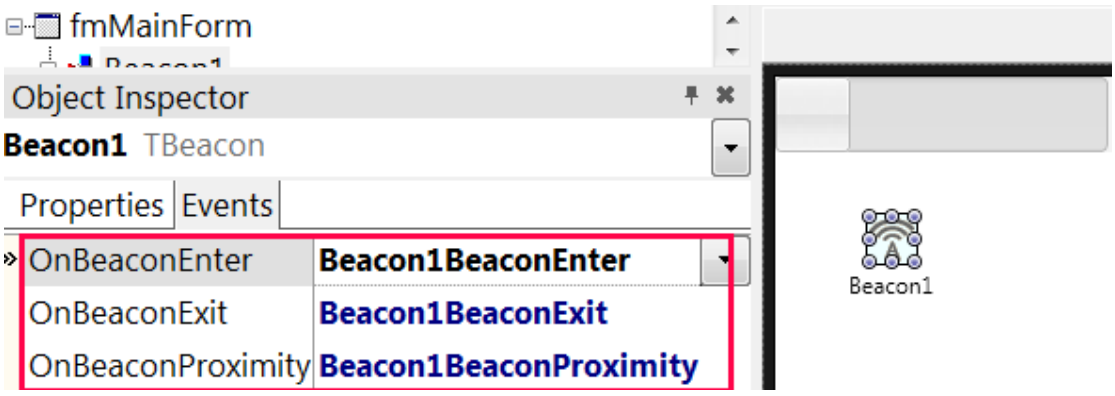
在上面的 TBeaconRegionItem 物件中其 Major 和 Minor 特性值都是-1，-1 代表要監督掃瞄具有相同 UUID 值的 Beacon 設備，不管其 Major 和 Minor 特性值是什麼。

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：

```
void __fastcall TfmMainForm::Switch1Switch(TObject *Sender)
{
  if (Switch1->IsChecked)
  StartScanBeacons();
  else
  StopScanBeacons();

}

void TfmMainForm::StartScanBeacons()
{
  Beacon1->StartScan();
}

void TfmMainForm::StopScanBeacons()
{
  Beacon1->StopScan();
}
```


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 OnBeaconEnter 和 OnBeaconExit 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：

```
void __fastcall TfmMainForm::Beacon1BeaconEnter(TObject * const
Sender, IBeacon * const ABeacon,
       const TBeaconList CurrentBeaconList)
{
  DisplayBeaconInfo(ABeacon, L"進入 : ");
}
//---------------------------------------------------------------
void __fastcall TfmMainForm::Beacon1BeaconExit(TObject * const
Sender, IBeacon * const ABeacon,
       const TBeaconList CurrentBeaconList)
{
  DisplayBeaconInfo(ABeacon, L"離開 : ");
}
//---------------------------------------------------------------
void TfmMainForm::DisplayBeaconInfo(IBeacon * const ABeacon, const
String sStatus)
{
  TMonitor::Enter(FLock);
  try
  {
  TListViewItem *alvi = lvBeacons->Items->Add();
  alvi->Text = sStatus + GUIDToString(ABeacon->GUID);
  alvi->Detail = "Major: " + IntToStr(ABeacon->Major) + " Minor: "
+ IntToStr(ABeacon->Minor)  + " Time :" + TimeToStr(Now());
  }
  __finally
  {
  TMonitor::Exit(FLock);
  }
}
```


上面的 Flock 物件變數是 TObject 型態的物件：

```
private:  // User declarations
  TObject *FLock;
```

它在主表單的 OnCreate 事件中建立並在 OnDestroy 事件中建立釋放：

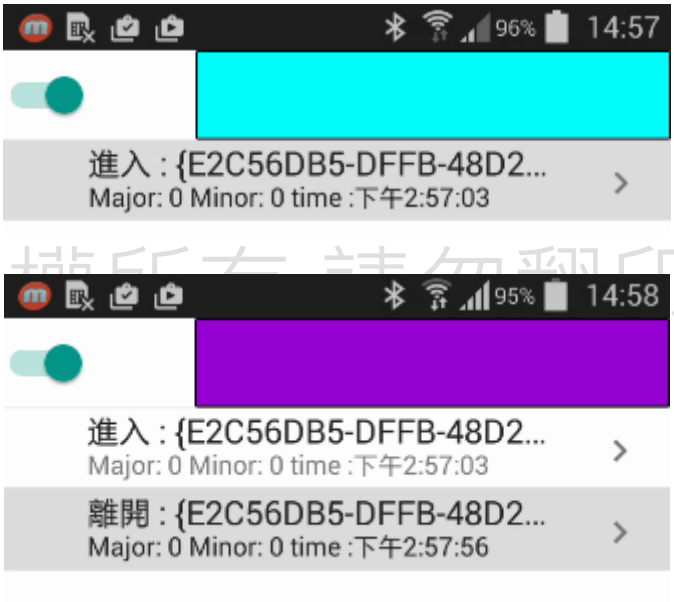
```
void __fastcall TfmMainForm::FormCreate(TObject *Sender)
{
  FLock = new TObject();
}
//---------------------------------------------------------------
void __fastcall TfmMainForm::FormDestroy(TObject *Sender)
{
  Beacon1->Enabled = false;
  delete FLock;
}
```

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：

```
void __fastcall TfmMainForm::Beacon1BeaconProximity(TObject *
const Sender, IBeacon * const ABeacon,
       TBeaconProximity Proximity)
{
  TMonitor::Enter(FLock);
  try
  {
  switch (ABeacon->Proximity)
  {
      case TBeaconProximity::Immediate :
        this->rtProximity->Fill->Color =
TAlphaColorRec::Springgreen;
        break;
      case TBeaconProximity::Near :
        this->rtProximity->Fill->Color = TAlphaColorRec::Aqua;
        break;
      case TBeaconProximity::Far :
        this->rtProximity->Fill->Color =
TAlphaColorRec::Slateblue;
        break;
      case TBeaconProximity::Away :
        this->rtProximity->Fill->Color =
TAlphaColorRec::Darkviolet;
```

```
        break;
    }
    }
    __finally
    {
    TMonitor::Exit(FLock);
    }
}
```

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。

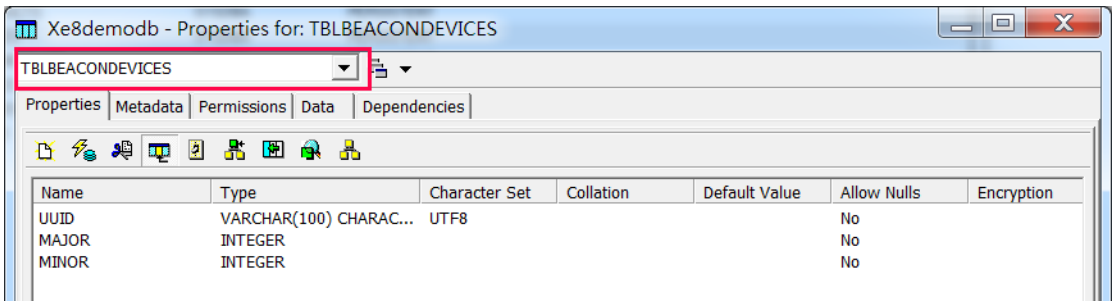
14 開發有趣的物聯網應用架構

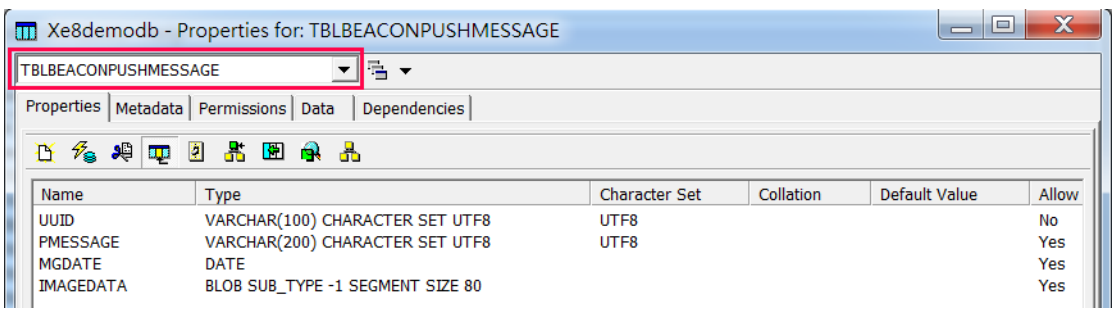
在前面的下節中已經說明了如何藉由 TBeacon 元件開發物聯網相關的 App，但只是偵測和掃瞄/監督 Beacon 設備並沒有什麼實際的做用。開發人員應該再結合其他的技術來提供有意義的服務，例如在本書前面說明的推播技術，開發人員可以結合 Beacon 設備和推播技術以及後端的服務伺服器來提供類似下圖的架構：

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 客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。

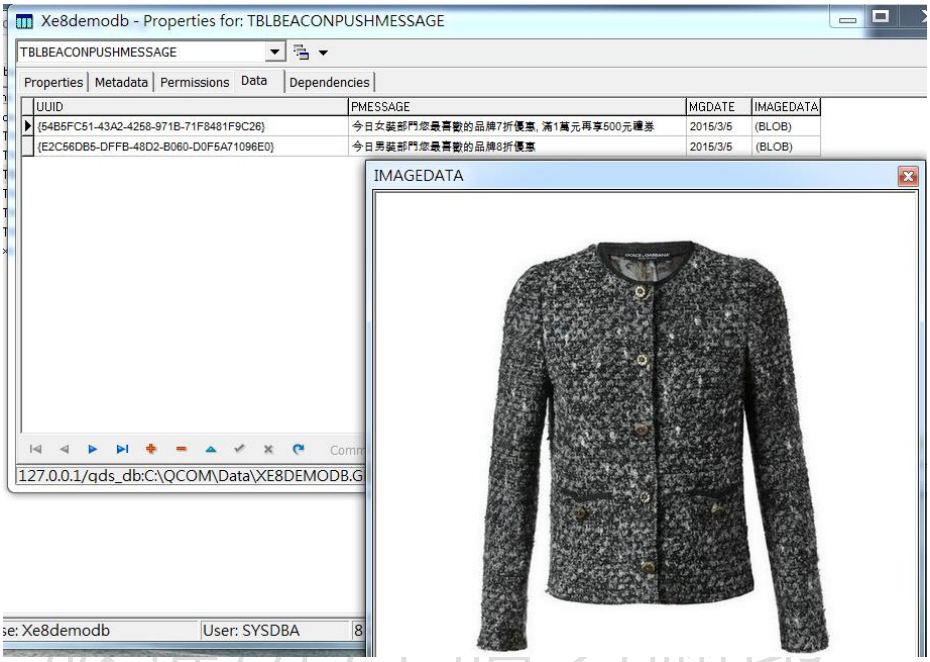
14-1 範例資料表

首先在 DEMODB.GDB 中讓我們加入 2 個資料表：TBLBEACONDEVICES 和 TBLBEACONPUSHMESSAGE：



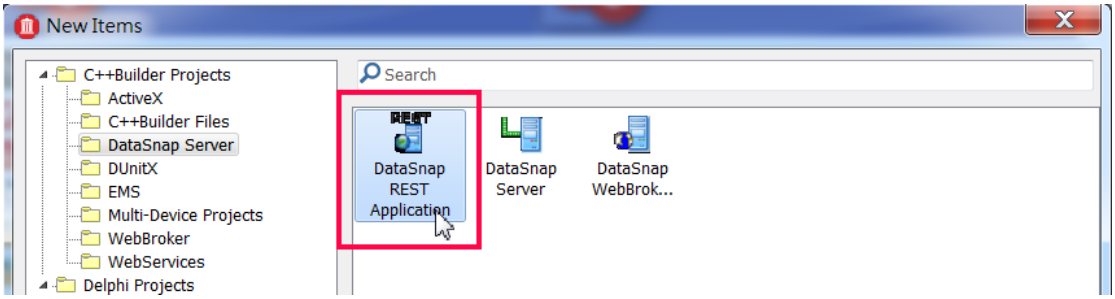


其中的TBLBEACONDEVICES資料表是使用來註冊客戶端App偵測到的Beacon設備而TBLBEACONPUSHMESSAGE資料表則儲存了要推播到特定Beacon設備附近App的推播訊息。如下所示在範例TBLBEACONPUSHMESSAGE資料表中尚包含了圖形的資料在使用者收到推播訊息之後還可以進一步看到詳細的資訊：

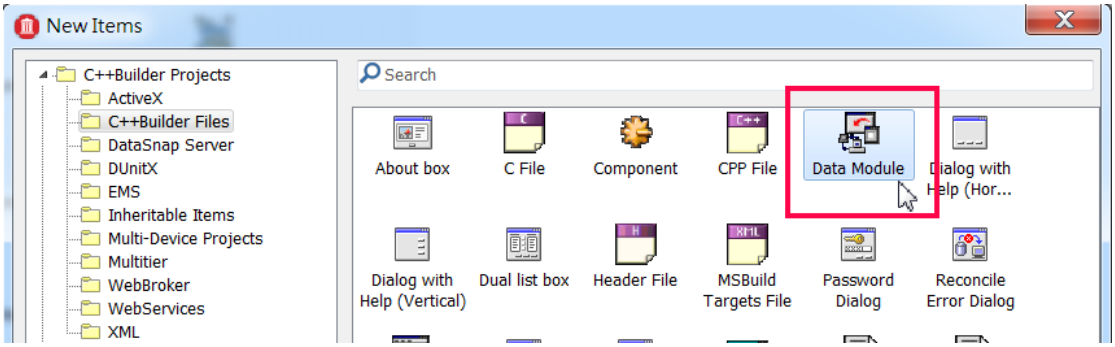


14-2 中介 DataSnap 伺服器

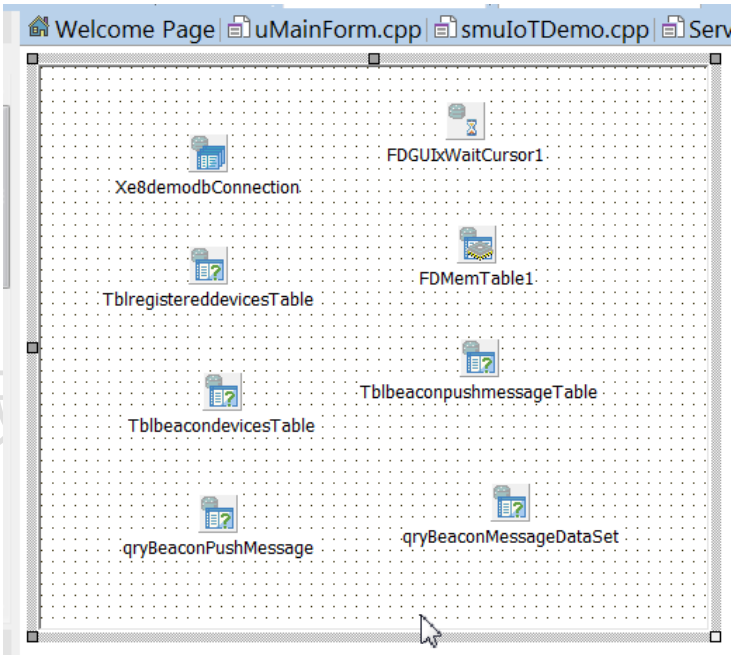
接著建立一個 DataSnap REST Application 專案：



在專案中建立一個資料模組：



在資料模組中加入如下的 FireDAC 元件準備存取前面的範例資料表：



接著在專案的 ServerMethodUnit 中加入如下的 2 個服務方法：

```
//For Beacon Services
  bool RegisterBeacon(const String sID, const String  sUUID, const
int iMajorID, const int iMinorID);
  TDataSet* GetBeaconMessage(const String sMessage);
```

RegisterBeacon()方法是在客戶端偵測到 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：

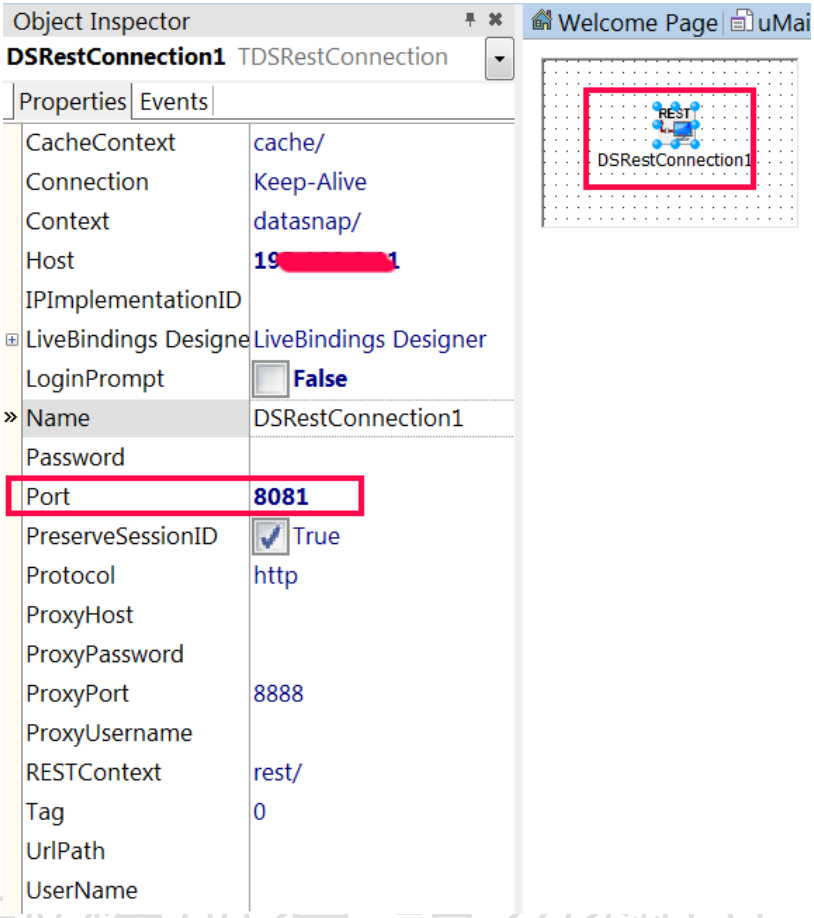
```
bool TsmBCBIoTDemo::RegisterBeacon(const String sID, const String
sUUID, const int iMajorID, const int iMinorID)
{
  bool bResult;

  System::TMonitor::Enter(dmIoTPushDemo);
  try
  {
  bResult = dmIoTPushDemo->RegisterBeacon(sID, sUUID, iMajorID,
iMinorID);
  }
  __finally
  {
  System::TMonitor::Exit(dmIoTPushDemo);
  }
  PushBeaconMessage(sID, sUUID);

  return bResult;
}
```


一旦客戶端 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() 方法則是呼叫 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客戶端的 App 中：

```
bool TsmBCBIoTDemo::PushBeaconMessage(const String sDID, const
String sUUID)
{
  TFDQuery *aDataSet = NULL;
  bool bResult = true;
  System::TMonitor::Enter(dmIoTPushDemo);
  try
  {
  aDataSet = dmIoTPushDemo->GetBeaconMessage(sUUID);
  if (aDataSet != NULL)
  {
      while (!aDataSet->Eof)
```

```
      {
        bResult = PushMessageTo(sDID,
aDataSet->FieldByName("PMESSAGE")->AsString) && bResult;
        aDataSet->Next();
      }
  }
  else
     bResult = false;
  }
  __finally
  {
  System::TMonitor::Exit(dmIoTPushDemo);
  }

  return bResult;
}
```

由於在前面章節的已有有了可執行推播功能的 GCM 伺服器，因此我們只需要藉由它推播 Beacon 設備的訊息即可。因此請在 ServerMethodUnit 中加入一個 TDSRestConnection 元件並設定它連結到前面章節的 GCM 伺服器並產生 C++Builder 客戶端的表頭檔和程式檔：

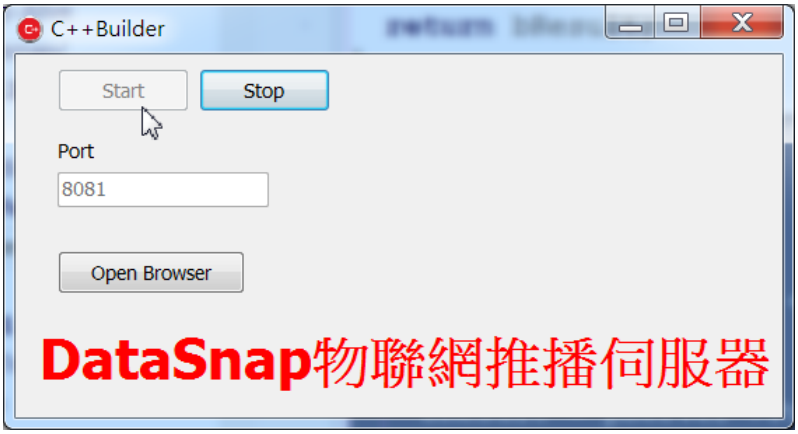


接著實作 PushMessageTo()方法呼叫前面章節的 GCM 伺服器中的 PushMessageTo()方法把 Beacon 設備的訊息推播到客戶端的手機中：

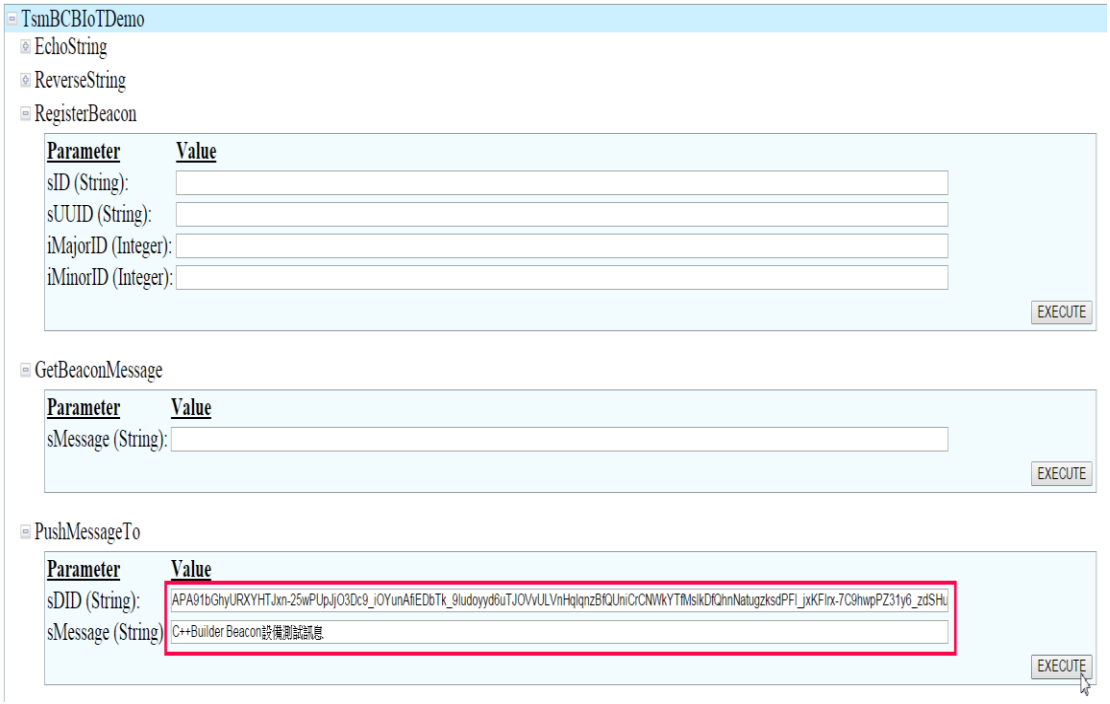
```
bool TsmBCBIoTDemo::PushMessageTo(const String sDID, const String
sMessage)
{

  bool bResult = false;

  TServerMethods1Client *pServer = new
TServerMethods1Client(this->DSRestConnection1);
  try
  {
  bResult = pServer->PushMessageTo(sDID, sMessage);
  }
  __finally
  {
  delete pServer;
```

```
  }

  return bResult;
}
```

現在請執行此範例 C++Builder DataSnap 伺服器：



開啟瀏覽器就可以看到範例 C++Builder DataSnap 伺服器輸出的服務方法，如果我們在瀏覽器中呼叫 PushMessageTo()方法：



就可以看到如下範例 C++Builder DataSnap 伺服器成功的呼叫前面的 GCM 伺服器：

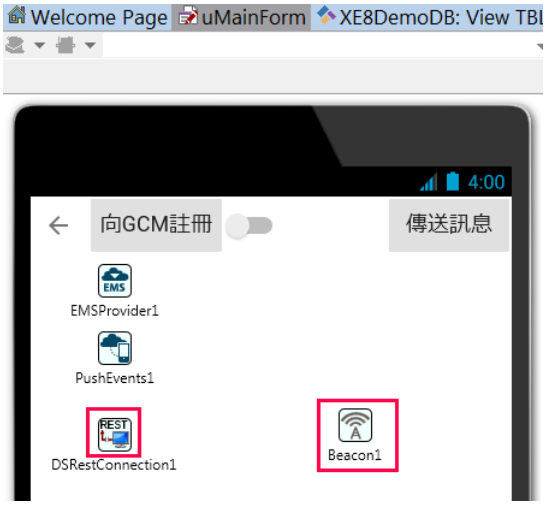
```
Executed: TsmBCBIoTDemo.PushMessageTo
        {"sDID":"APA91bGhyURXYHTJxn-
Result: 25wPUpJjO3Dc9_iOYunAfiEDbTk_9Iudoyyd6uTJOVvULVnHqIqnzBfQUniCrCNWkYTfMslkDfQhnNatugzksdPF1_jxKFIrx-
        7C9hwpPZ31y6_zdSHuCGo2sqtspTo4oBQ1Q7yx5YhiTBw","sMessage":"C++Builder Beacon設備測試訊息","result":true}
```

而範例 C++Builder DataSnap 伺服器也藉由 GCM 伺服器推播訊息到客戶端的手機中：

接下來就來開發移動客戶端 App 並結合 Beacon 的功能。

14-3 移動客戶端

最後讓我們在客戶端 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：

```
 void __fastcall TfmMainForm::swActivateBeaconSwitch(TObject
*Sender)
{
  if (swActivateBeacon->IsChecked)
  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：

```
void __fastcall TfmMainForm::Beacon1BeaconEnter(TObject * const
Sender, IBeacon * const ABeacon,
        const TBeaconList CurrentBeaconList)
{
  try
  {
  ListView1->Items->Add()->Text = L"發現 Beacon : " +
GUIDToString( ABeacon->GUID);
  RegisterBeaconToServer(ABeacon);
  }
  catch (const Sysutils::Exception& ex)
```

```
  {
  ListView1->Items->Add()->Text = ex.Message;
  }
  TabControl1->TabIndex = 1;
}
```

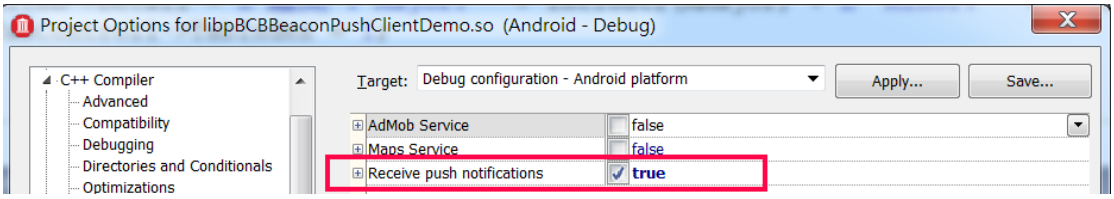
RegisterBeaconToServer()方法藉由 TDSRestConnection 元件自動產生的客戶端連結程式碼以連結範例 DataSnap 伺服器：

```
void TfmMainForm::RegisterBeaconToServer(IBeacon * const ABeacon)
{

  TsmBCBIoTDemoClient *aServer = new
TsmBCBIoTDemoClient(this->DSRestConnection1);
  try
  {
  try
  {
    if (aServer->RegisterBeacon(PushEvents1->DeviceToken,
GUIDToString(ABeacon->GUID), ABeacon->Major, ABeacon->Minor))
      ListView1->Items->Add()->Text = L"Beacon 註冊成功";
  }
  catch (const Sysutils::Exception& ex)
  {
    ;
  }
  }
  __finally
  {
  delete aServer;
  }
}
```


14-4 執行 DataSnap 伺服器和客戶端 App

在編譯並執行此 Beacon 推播 App 之前，記得要把藍牙存取許可打開：

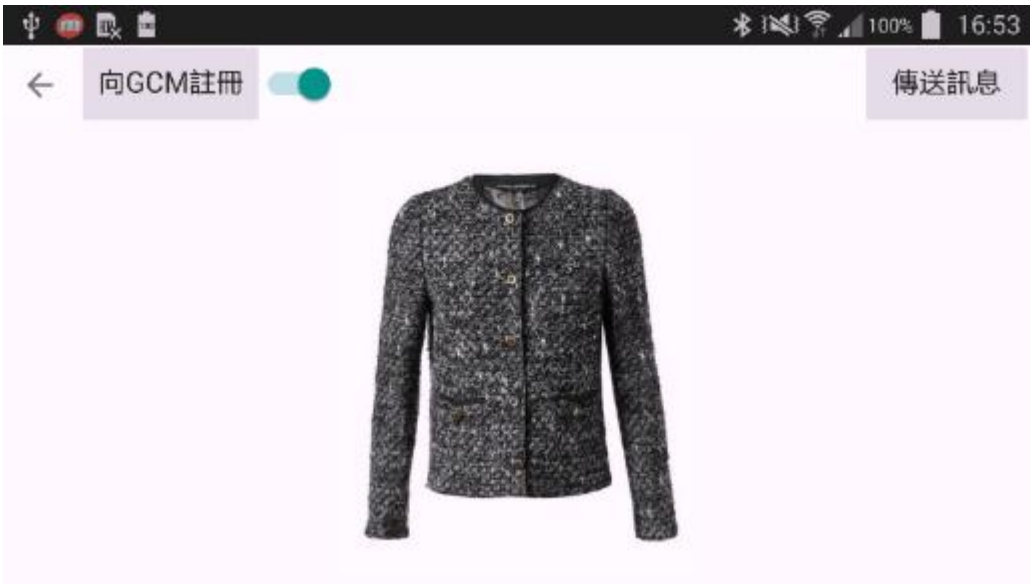
也需要把推播功能開啟：



最後要在範例專案的 AndroidManifest.template.xml 檔中加入如下的服務：

```
<%activity%>
<service
android:name="com.embarcadero.gcm.notifications.GCMIntentService
" />
```


完成上面的步驟後就可以編譯並執行 DataSnap 伺服器和客戶端 App，就可在客戶端 App 看到如下的執行結果，客戶端 App 能夠自動偵測 Beacon 設備，自動接收推播訊息，並且可在客戶端 App 中查詢到圖形的推播資料了：



15 新的 JSON 類別庫

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，C++Builder 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 C++Builder 的 DataSnap，REST 和開發人員的 App 中就更顯需求了。因此在 C++Builder 中開始提供新一代的 JSON 框架。

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 C++Builder 物件之間的序列化功能
- 可提供開發人員客製化的能力
- 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

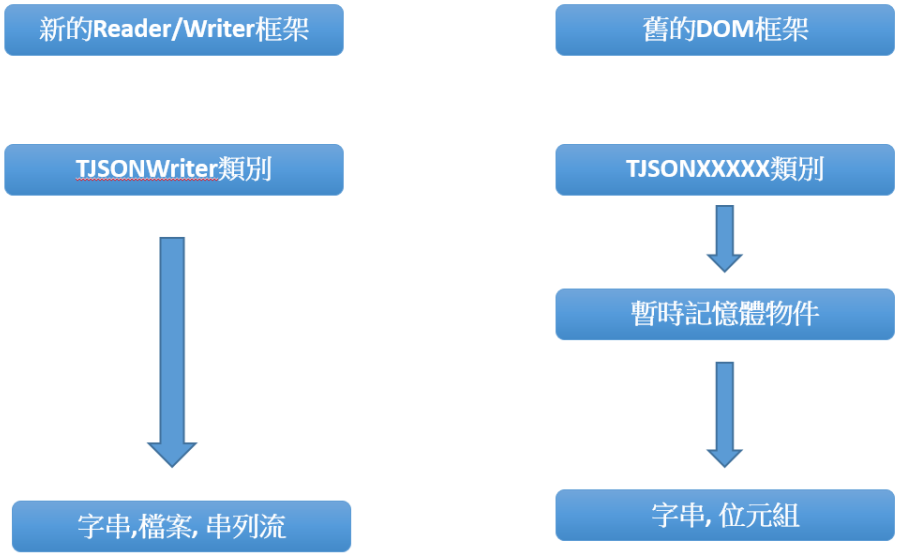
當然其中還有更多的類別無無一一列出，不過開發人員基本上只需要使用上述的類別應該就能完成大部份的工作。

15-1 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。

在之前的版本中 C++Builder 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：



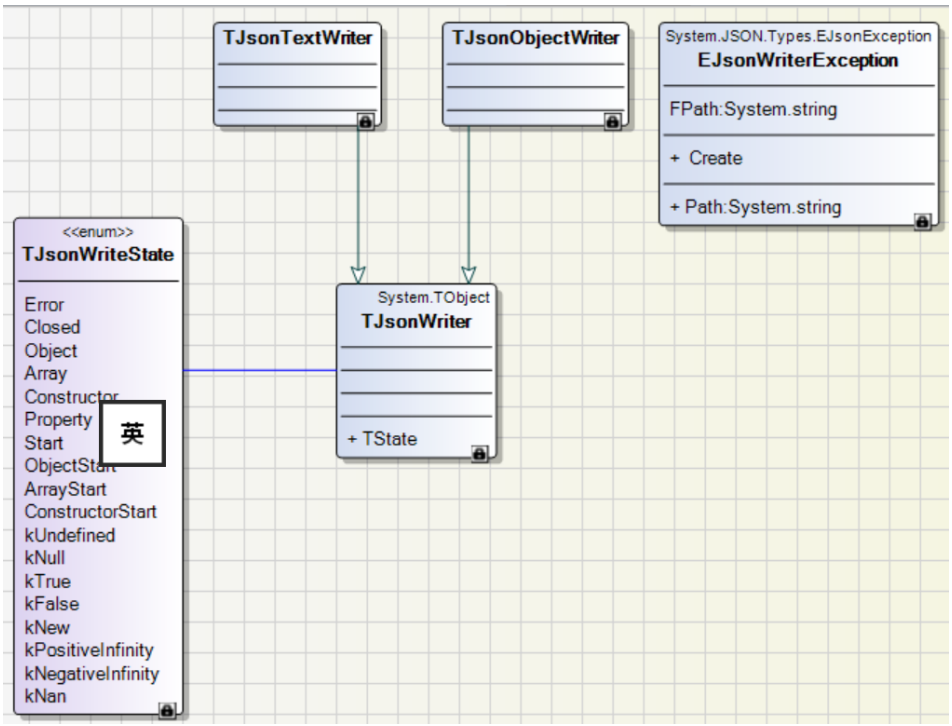
其實新的 JSON 框架使用上非常的簡單，讓我們使用下的 JSON 資料來說明如何藉由 TJsonWriter 的相關類別輸出下面 JSON 格式的資料，在下一小節再使用 TJsonrReader 的相關類別再把資料讀出：

```
{
        "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。

15-1-1 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：

```
__fastcall TJsonTextWriter(System::Classes::TTextWriter* const
TextWriter);
```

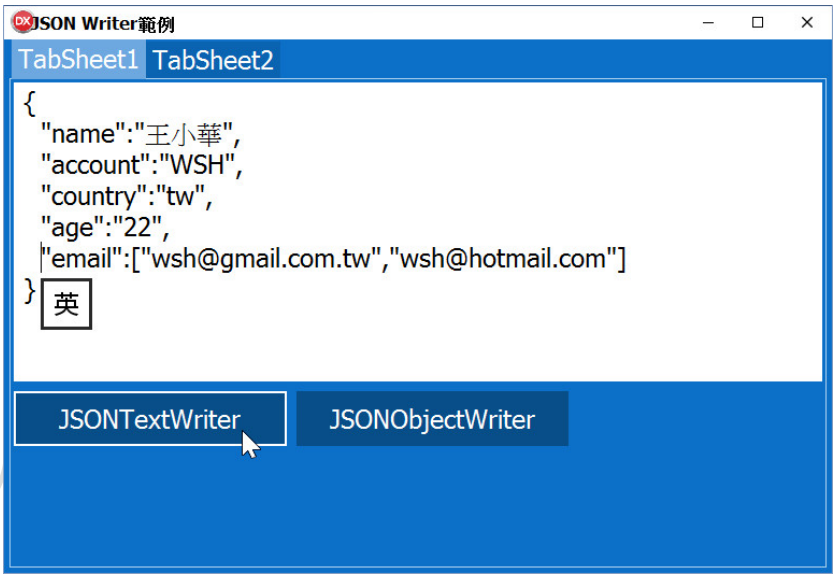
TJsonTextWriter 類別提供了許多的 Write 方法讓程式師呼叫，例如上面的資料格式是 JSON 物件，因此 TJsonTextWriter 類別提供了 WriteStartObject/WriteEndObject 方法，如果要寫出 JSON 陣列物件，那可呼叫 WriteStartArray/WriteEndArray 方法。要寫出 JSON Pair 的名稱可用 WritePropertyName，而要寫出 JSON Pair 的數值部分則可用各種複載的 WriteValue 方法。

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    void TfmMainForm::WritePair(TJsonTextWriter *jtw, const
```

```
String sName, const String sValue)
002   {
003     jtw->WritePropertyName(sName);
004     jtw->WriteValue(sValue);
005   }
006
007   void __fastcall TfmMainForm::Button1Click(TObject *Sender)
008   {
009     TJsonTextWriter *jtw;
010     TStringWriter *sw = new TStringWriter();
011     try
012     {
013       jtw = new TJsonTextWriter(sw);
014
015       jtw->WriteStartObject();
016       WritePair(jtw, "name", L"王小華");
017       WritePair(jtw, "account", "WSH");
018       WritePair(jtw, "country", "tw");
019       WritePair(jtw, "age", "22");
020
021       //JSON 陣列
022       jtw->WritePropertyName("emails");
023       jtw->WriteStartArray();
024       String eMail1 = "wsh@gmail.com.tw";
025       jtw->WriteValue(String("wsh@gmail.com.tw"));
026       jtw->WriteValue(String("wsh@hotmail.com"));
027       jtw->WriteEndArray();
028       jtw->WriteEndObject();
029       FDemoData = sw->ToString();
030       Memo1->Lines->Text = FDemoData;
031     }
032     __finally
033     {
034       jtw->Close();
035       delete jtw;
036       delete sw;
037     }
038   }
```

010 行先建立一個 TStringWriter 物件，準備把 JSON 資料寫入，013 行建立 TJsonTextWriter 物件。由於"王小華"是使用 JSON 物件封裝，因此 015 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 023 行再呼叫 WriteStartArray()方法寫出'['，最後 027，028 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：



同樣的工作我們也可以使用 TJsonObjectWriter 類別來完成：

```
001    void TfmMainForm::WritePair(TJsonObjectWriter *jow, const
String sName, const String sValue)
002    {
003      jow->WritePropertyName(sName);
004      jow->WriteValue(sValue);
005    }
006
007    void __fastcall TfmMainForm::Button2Click(TObject *Sender)
008    {
009      TJsonObjectWriter *jow = new TJsonObjectWriter(true);
010      try
011      {
012        jow->WriteStartObject();
013        WritePair(jow, "name", L"王小華");
```

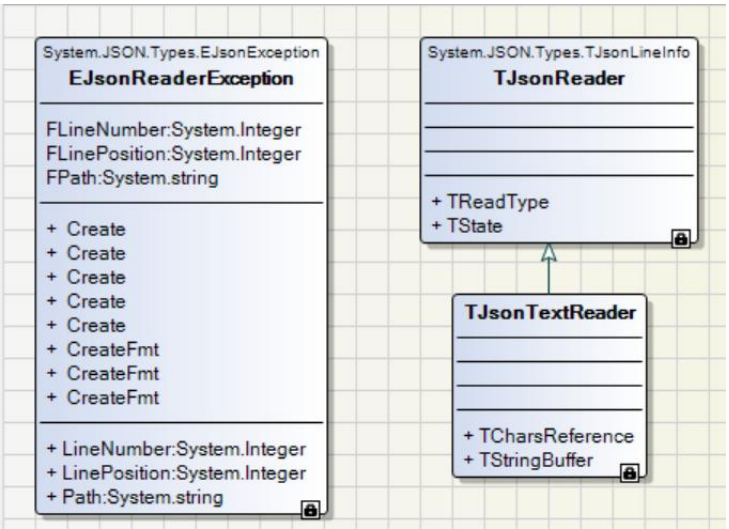
```
014       WritePair(jow, "account", "WSH");
015       WritePair(jow, "country", "tw");
016       WritePair(jow, "age", "22");
017
018       //JSON 陣列
019       jow->WritePropertyName("emails");
020       jow->WriteStartArray();
021       String eMail1 = "wsh@gmail.com.tw";
022       jow->WriteValue(String("wsh@gmail.com.tw"));
023       jow->WriteValue(String("wsh@hotmail.com"));
024       jow->WriteEndArray();
025       jow->WriteEndObject();
026       FDemoData = jow->JSON->ToString();
027       Memo1->Lines->Text = FDemoData;
028     }
029     __finally
030     {
031       jow->Close();
032       delete jow;
033     }
034   }
```


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 026 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。

15-1-2 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.hpp 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：

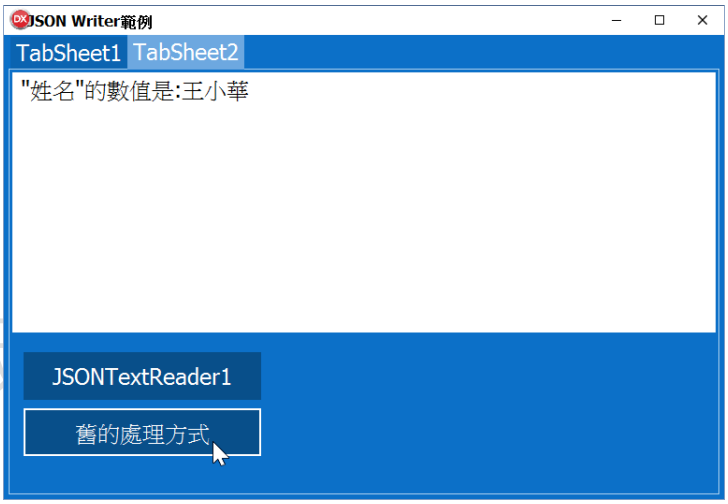

```
{
  "姓名" : "王小華"
}
```

那麼在以前可以使用如下的程式碼：

```
void __fastcall TfmMainForm::Button4Click(TObject *Sender)
{
  TJSONObject *jo;

  TJSONValue *jv = TJSONObject::ParseJSONValue(SPEOPLEDATA);
  try
  {
  TJSONObject *jo = dynamic_cast< TJSONObject* > (jv);
  if (jo != 0)
  {
    TJSONPair *jp = jo->Pairs[0];
    String sData = L"\"" + jp->JsonString->Value() + L"\"的數值是:"
+ jp->JsonValue->Value();
```

```
    Memo2->Lines->Add(sData);
  }
  }
  __finally
  {
  delete jv;
  }
}
```

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：

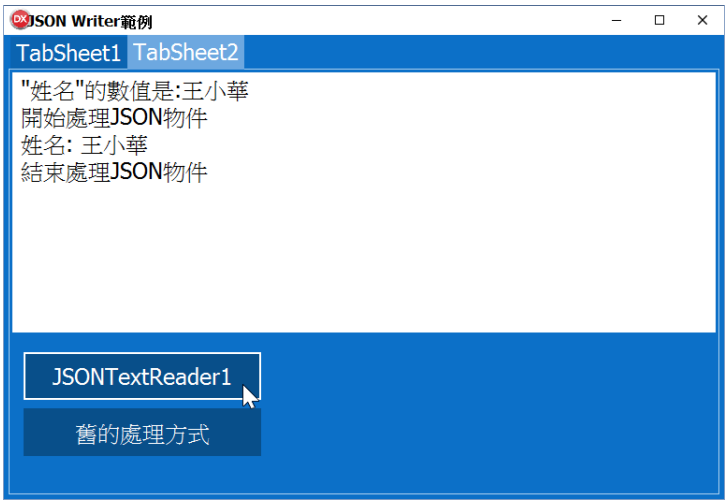


在新的 JSON 框架中可以使用如下的程式碼來解析 JSON 資料，006 行先把 JSON 資料傳入 TStringReader 物件，再於 007 行建立 TJsonTextReader 物件，011 行開始呼叫 TJsonTextReader 物件的 Read()方法讀取 JSON 資料，接著可使用一個 switch 子句來判斷目前讀取到的內容再決定要如何處理：

```
001    const String SPEOPLEDATA = L"{\"姓名\" : \"王小華\"}";
002
003    void __fastcall TfmMainForm::Button3Click(TObject *Sender)
004    {
005      String sData;
006      TStringReader *sr = new TStringReader(SPEOPLEDATA);
007      TJsonTextReader *jr = new TJsonTextReader(sr);
008      try
009      {
010        jr->Rewind();
```

```
011       while (jr->Read())
012       {
013         switch (jr->TokenType)
014         {
015           case TJsonToken::StartObject:
016             Memo2->Lines->Add(L"開始處理 JSON 物件");
017             break;
018           case TJsonToken::PropertyName:
019             sData = jr->Value.ToString() + ": " +
jr->ReadAsString();
020             break;
021           case TJsonToken::String:
022             sData = sData + jr->ReadAsString();
023             break;
024           case TJsonToken::EndObject:
025             Memo2->Lines->Add(sData);
026             Memo2->Lines->Add(L"結束處理 JSON 物件");
027             break;
028         }
029       }
030     }
031     __finally
032     {
033       delete sr;
034       delete jr;
035     }
036   }
```


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。

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 String SDEMODATA = L"{\"name\" : \"王小華\",
\"account\" : \"WSH\", \"country\" : \"tw\", \"age\" : \"22\",
\"email\" : [\"wsh@gmail.com.tw\",\"wsh@hotmail.com\"]}";
002
003    void __fastcall TfmMainForm::Button5Click(TObject *Sender)
004    {
005      String sData;
006      TStringReader *sr = new TStringReader(SDEMODATA);
007      TJsonTextReader *jr = new TJsonTextReader(sr);
008      try
009      {
010        jr->Rewind();
011        while (jr->Read())
012        {
013          switch (jr->TokenType)
014          {
015            case TJsonToken::StartObject:
016              Memo2->Lines->Add(L"開始處理 JSON 物件");
017              break;
018            case TJsonToken::PropertyName:
```

```
019             if (jr->Value.ToString() != "email")
020             {
021               sData = jr->Value.ToString() + ": " +
jr->ReadAsString();
022               Memo2->Lines->Add(sData);
023             }
024             break;
025           case TJsonToken::EndObject:
026             Memo2->Lines->Add(L"結束處理 JSON 物件");
027             break;
028           case TJsonToken::StartArray:
029             Memo2->Lines->Add(L"開始處理 JSON 陣列");
030             while (jr->Read())
031             {
032               switch (jr->TokenType)
033               {
034                 case TJsonToken::String:
035                   Memo2->Lines->Add(jr->Value.ToString());
036                   break;
037                 case TJsonToken::EndArray:
038                   Memo2->Lines->Add(L"結束始處理 JSON 陣列");
039                   break;
040               }
041             }
042             break;
043         }
044       }
045     }
046     __finally
047     {
048       delete sr;
049       delete jr;
050     }
051   }
```


上面的關鍵點在 028 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

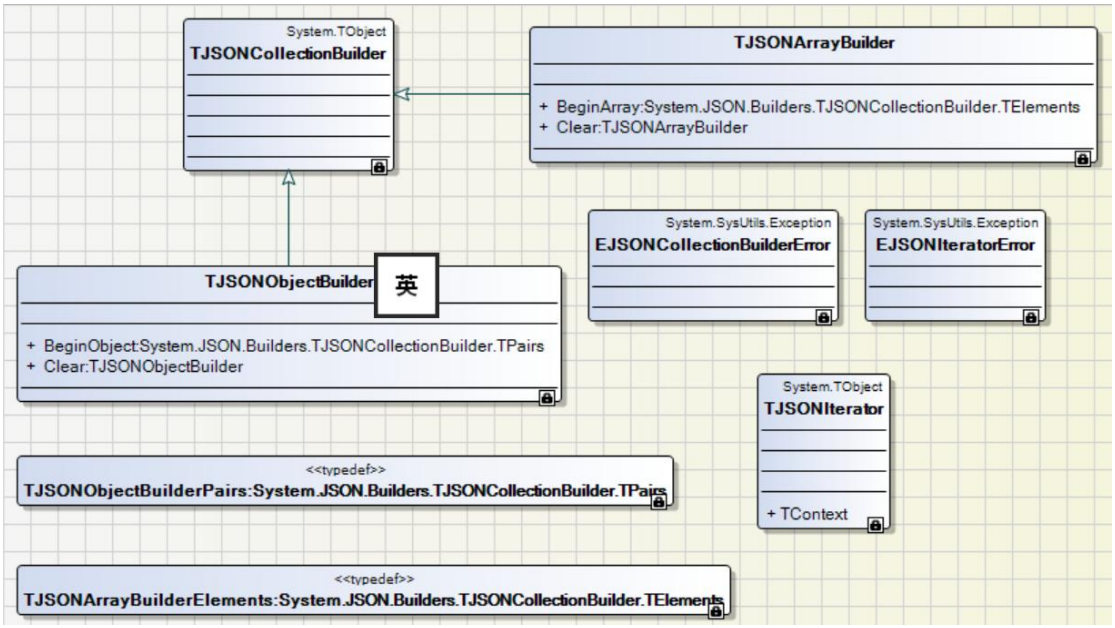
存取陣列中的每一個元素值，下面就是執行的結果，我們可以看到 TJsonTextReader 物件可以正確解析和讀取出 JSON 資料的內容：

15-2 JSON Builder 類別

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 JSONl 資料中的內容

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:

```
class PASCALIMPLEMENTATION TJSONObjectBuilder : public
TJSONCollectionBuilder
{
  typedef TJSONCollectionBuilder inherited;

public:
  HIDESBASE TJSONCollectionBuilder::TPairs* __fastcall
BeginObject(void);
  TJSONObjectBuilder* __fastcall Clear(void);
public:
```

```
  /* TJSONCollectionBuilder.Create */ inline __fastcall
TJSONObjectBuilder(System::Json::Writers::TJsonWriter* const
AJSONWriter)/* overload */ : TJSONCollectionBuilder(AJSONWriter)
{ }
  …
  /* TJSONCollectionBuilder.Destroy */ inline __fastcall virtual
~TJSONObjectBuilder(void) { }

};
```

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 複載方法可在 JSON 物件中加入資料，請注意的是 Add 複載方法又回傳原先的 TPairs 物件：

```
  class PASCALIMPLEMENTATION TPairs : public
TJSONCollectionBuilder::TBaseCollection
  {
      typedef TJSONCollectionBuilder::TBaseCollection inherited;

  public:
      HIDESBASE TJSONCollectionBuilder::TPairs* __fastcall
Add(const System::UnicodeString AKey, const System::UnicodeString
AValue)/* overload */;
      HIDESBASE TJSONCollectionBuilder::TPairs* __fastcall
Add(const System::UnicodeString AKey, const int AValue)/* overload
*/;
      HIDESBASE TJSONCollectionBuilder::TPairs* __fastcall
Add(const System::UnicodeString AKey, const unsigned AValue)/*
overload */;
  …
  }
```


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：

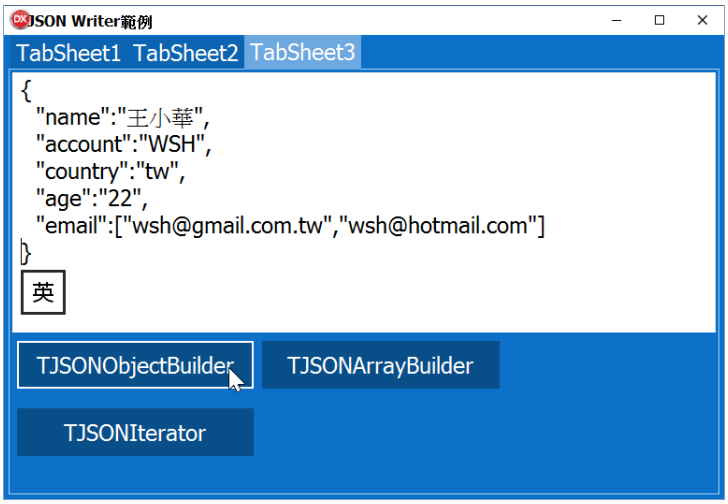
```
    void __fastcall EndAll(void);
```

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：

```
001   void __fastcall TfmMainForm::Button6Click(TObject *Sender)
002   {
003     TStringWriter *sw = new TStringWriter();
004     TJsonTextWriter *jtw = new TJsonTextWriter(sw);
005     TJSONObjectBuilder *joBuilder = new
TJSONObjectBuilder(jtw);
006     try
007     {
008       joBuilder->BeginObject()
009        ->Add("name",String(L"王小華"))
010        ->Add("account",String(L"WSH"))
011        ->Add("country",String(L"tw"))
012        ->Add("age",String(L"22"))
013        ->BeginArray("email")
014         ->Add(String(L"wsh@gmail.com.tw"))
015          ->Add(String(L"wsh@hotmail.com"))
016        ->EndAll();
017
018        Memo3->Lines->Text = sw->ToString();
019     }
020     __finally
021     {
022       delete joBuilder;
023       delete jtw;
024       delete sw;
025     }
026   }
```


003 和 004 行分別建立了 TStringWriter 和 TJsonTextWriter 物件，005 行再建立 TJSONObjectBuilder 物件並把 TJsonTextWriter 物件傳入。從 008 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：



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：


```
class PASCALIMPLEMENTATION TJSONArrayBuilder : public
TJSONCollectionBuilder
{
  typedef TJSONCollectionBuilder inherited;

public:
  HIDESBASE TJSONCollectionBuilder::TElements* __fastcall
BeginArray(void);
  TJSONArrayBuilder* __fastcall Clear(void);
public:
  …
}
```

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：

```
  class PASCALIMPLEMENTATION TElements : public
TJSONCollectionBuilder::TBaseCollection
  {
```

```
      typedef TJSONCollectionBuilder::TBaseCollection inherited;

  public:
      HIDESBASE TJSONCollectionBuilder::TElements* __fastcall
Add(const System::UnicodeString AValue)/* overload */;
      HIDESBASE TJSONCollectionBuilder::TElements* __fastcall
Add(const int AValue)/* overload */;
      HIDESBASE TJSONCollectionBuilder::TElements* __fastcall
Add(const unsigned AValue)/* overload */;

              ...

      HIDESBASE TJSONCollectionBuilder::TPairs* __fastcall
BeginObject(void)/* overload */;
      HIDESBASE TJSONCollectionBuilder::TElements* __fastcall
BeginArray(void)/* overload */;
      HIDESBASE TJSONCollectionBuilder::TParentCollection*
__fastcall EndArray(void);
      TJSONCollectionBuilder::TElements* __fastcall AsRoot(void);
  public:
      /* TBaseCollection.Create */ inline __fastcall
TElements(TJSONCollectionBuilder* const AOwner, const int
ARootDepth) : TJSONCollectionBuilder::TBaseCollection(AOwner,
ARootDepth) { }

  public:
      /* TObject.Destroy */ inline __fastcall virtual
~TElements(void) { }

  /* Hoisted overloads: */

  };
```

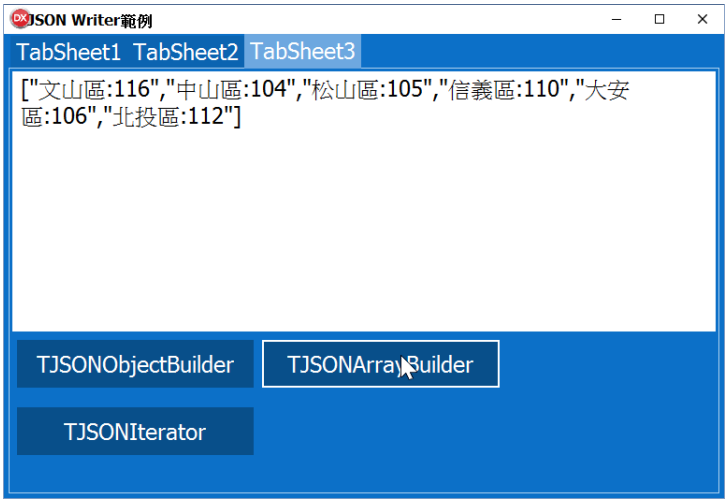

例如下面的程式碼可以建立一個包含台北市郵遞區號的 JSON 陣列：

```
001    void __fastcall TfmMainForm::Button7Click(TObject *Sender)
002    {
003      TStringWriter *sw = new TStringWriter();
```

```
004     TJsonTextWriter *jtw = new TJsonTextWriter(sw);
005     TJSONArrayBuilder *jaBuilder = new TJSONArrayBuilder(jtw);
006     try
007     {
008       jaBuilder->BeginArray()
009        ->Add(String(L"文山區:116") )
010        ->Add(String(L"中山區:104") )
011        ->Add(String(L"松山區:105") )
012        ->Add(String(L"信義區:110") )
013        ->Add(String(L"大安區:106") )
014        ->Add(String(L"北投區:112") )
015       ->EndAll();
016
017       Memo3->Lines->Text = sw->ToString();
018     }
019     __finally
020     {
021       delete jaBuilder;
022     }
023   }
```


003 和 004 行同樣分別建立了 TStringWriter 和 TJsonTextWriter 物件，005 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。

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())
{
  …
}
```

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 標的是什麼：

```
  __property System::Json::Types::TJsonToken Type = {read=FType,
nodefault};
```

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 String SDEMODATA = L"{\"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：

```
001    void __fastcall TfmMainForm::Button8Click(TObject *Sender)
002    {
003      String sData = "";
004      TStringReader *sr = new TStringReader(SDEMODATA);
005      TJsonTextReader *jr = new TJsonTextReader(sr);
006      TJSONIterator *jInerator = new TJSONIterator(jr);
007      try
008      {
009        while (jInerator->Next())
010        {
011          if ((jInerator->Index == -1) && (jInerator->Type !=
TJsonToken::StartArray))
```

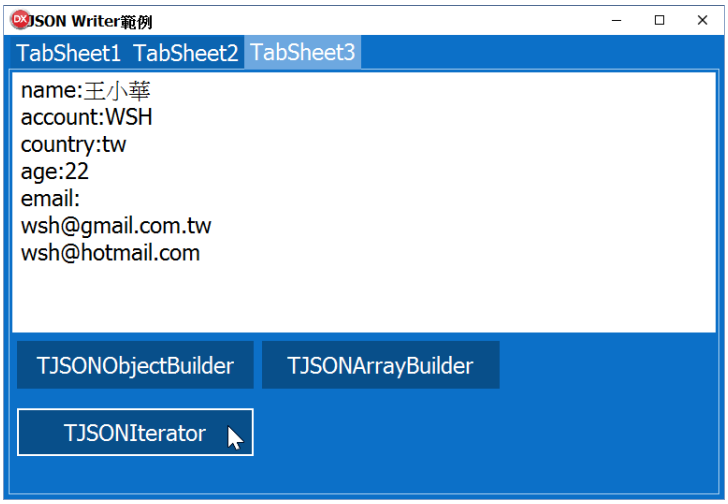
```
012           sData = sData + jInerator->Key + ":" +
jInerator->AsString + "\n";
013         else
014         {
015           if (jInerator->Index == -1)
016             sData = sData + jInerator->Key + ":";
017           else
018             sData = sData + "\n" + jInerator->AsString;
019           jInerator->Recurse();
020         }
021       }
022       Memo3->Lines->Text = sData;
023     }
024     __finally
025     {
026       delete jInerator;
027       delete jr;
028       delete sr;
029     }
030   }
```


004 行使用 TStringReader 物件輸入"王小華"這筆 JSON 物件，005 行再建立 TJSONTextReader 物件，006 行建立 TJSONIterator 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：

```
__property System::UnicodeString Key = {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。

015~019 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:



15-3 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。

15-3-1 TBsonWriter

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：

```
  __fastcall TBsonWriter(System::Classes::TBinaryWriter* const
ABinaryWriter)/* overload */;
  __fastcall TBsonWriter(System::Classes::TStream* const Stream)/*
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：

```
  virtual void __fastcall WriteStartObject(void);
  virtual void __fastcall WriteEndObject(void);
  virtual void __fastcall WriteStartArray(void);
  virtual void __fastcall WriteEndArray(void);
```

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()複載方法可讓程式師寫入資料：

```
  virtual void __fastcall WriteValue(const System::UnicodeString
Value)/* overload */;
  virtual void __fastcall WriteValue(int Value)/* overload */;
  …
```

其中一個重要的方法就是 WriteToken()方法：

```
  void __fastcall WriteToken(System::Json::Readers::TJsonReader*
const Reader)/* overload */;
  void __fastcall WriteToken(System::Json::Readers::TJsonReader*
const Reader, bool WriteChildren)/* 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 進位形式的資料嗎?)：

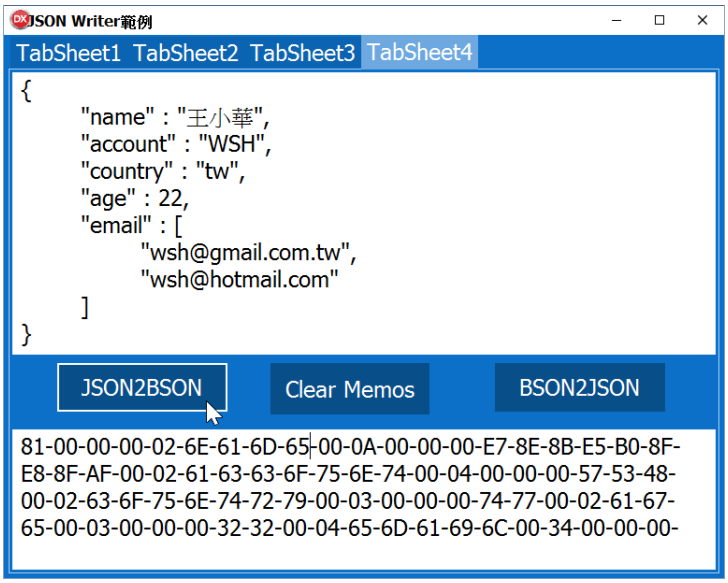
```
void __fastcall TfmMainForm::Button9Click(TObject *Sender)
{
  Memo5->Lines->Text = Bytes2String(Json2Bson(SDEMODATA));
}
```

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，005 行建立 TBytesStream 物件，006 行再建立 TBsonWriter 物件並傳入 TBytesStream 物件。接著我們只需要在 011 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 System::TArray__1<System::Byte>型態回傳轉換後的 BSON 格式資料：

```
001    System::TArray__1<System::Byte>
TfmMainForm::Json2Bson(const String AJson)
```

```
002   {
003     TStringReader *sr = new TStringReader(AJson);
004     TJsonTextReader *jr = new TJsonTextReader(sr);
005     TBytesStream *Stream = new TBytesStream(NULL);
006     TBsonWriter *BsonWriter = new TBsonWriter(Stream);
007     System::TArray__1<System::Byte> bResult;
008
009     try
010     {
011       BsonWriter->WriteToken(jr);
012       bResult.set_length(Stream->Size);
013       Stream->Position = 0;
014       Stream->Read(bResult, Stream->Size);
015
016       return bResult;
017     }
018     __finally
019     {
020       delete jr;
021       delete sr;
022       delete BsonWriter;
023       delete Stream;
024     }
025   }
```


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：



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 System::TArray__1<System::Byte>轉回 JSON：

```
001    String TfmMainForm::Bson2Json(const
System::TArray__1<System::Byte> ABson)
002    {
003      String sResult = "";
004      TBytesStream *Stream = new TBytesStream(ABson);
005      TBsonReader *BsonReader = new TBsonReader(Stream, false);
006      TStringWriter *sw = new TStringWriter();
007      TJsonTextWriter *jtw = new TJsonTextWriter(sw);
008      jtw->Formatting = TJsonFormatting::Indented;
009      try
010      {
011        jtw->WriteToken(BsonReader);
012        sResult = sw->ToString();
013      }
014      __finally
015      {
016        delete jtw;
017        delete sw;
018        delete BsonReader;
019        delete Stream;
020      }
```

```
021
022     return sResult;
023   }
```

004 行先使用 TBytesStream 接收傳入的包含 BSON 格式資料的 System::TArray__1<System::Byte> ， 005 行使用下一小節介紹的 TBsonReader物件讀入並解析BSON格式資料，011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：

```
 void __fastcall WriteToken(System::Json::Readers::TJsonReader*
const Reader)/* overload */;
 void __fastcall WriteToken(System::Json::Readers::TJsonReader*
const Reader, bool WriteChildren)/* overload */;
```

而 TBsonReader 是從 TJsonReader 類別繼承下來，因此它就可以把 BSON 轉成 JSON：

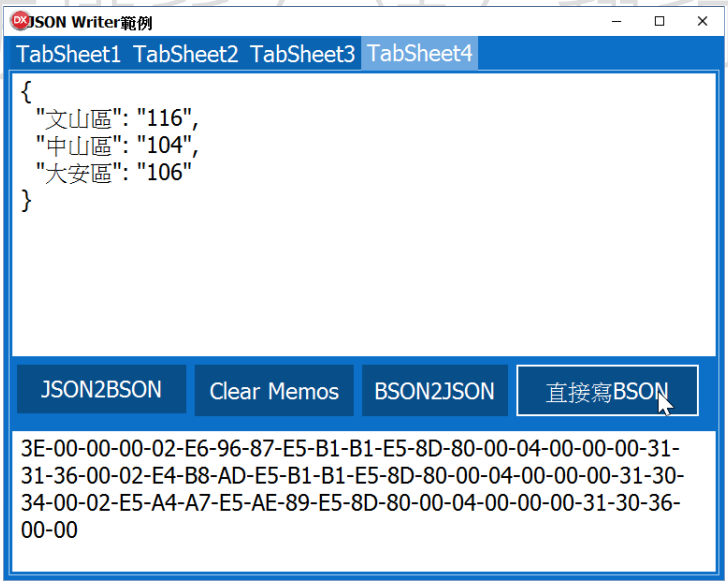

```
class PASCALIMPLEMENTATION TBsonReader : public
System::Json::Readers::TJsonReader
```

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   void __fastcall TfmMainForm::Button12Click(TObject *Sender)
002   {
003     TBytesStream *Stream = new TBytesStream(NULL);
004     TBsonWriter *BsonWriter = new TBsonWriter(Stream);
005     try
006     {
007       BsonWriter->WriteStartObject();
008       BsonWriter->WritePropertyName(L"文山區");
009       BsonWriter->WriteValue(String(L"116"));
010       BsonWriter->WritePropertyName(L"中山區");
011       BsonWriter->WriteValue(String(L"104"));
012       BsonWriter->WritePropertyName(L"大安區");
013       BsonWriter->WriteValue(String(L"106"));
```

```
014       BsonWriter->WriteEndObject();
015       BsonWriter->Close();
016       Button11Click(Sender);
017       Stream->Position = 0;
018       Memo4->Lines->Text = Bson2Json(Stream->Bytes);
019       Memo5->Lines->Text =
Bytes2String( Json2Bson(Memo4->Lines->Text) );
020     }
021     __finally
022     {
023       delete BsonWriter;
024       delete Stream;
025     }
026   }
```

下面是藉由上面的程式碼把台北市郵遞區號寫成 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。



15-3-2 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：

```
  __fastcall TBsonReader(System::Classes::TStream* const AStream,
bool AReadAsRootValuesArray);
```

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：

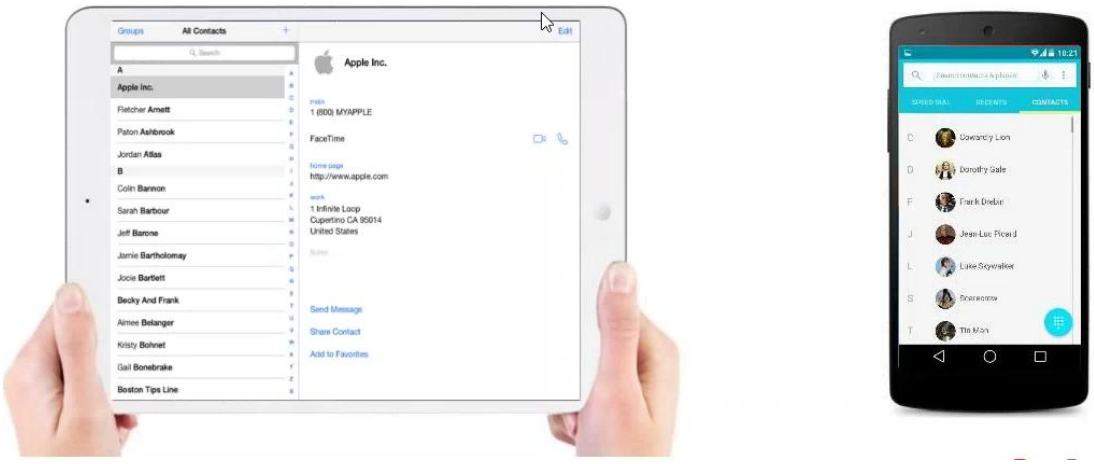
```
  virtual void __fastcall Rewind(void);
```

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。

16 新的 AddressBook 元件()

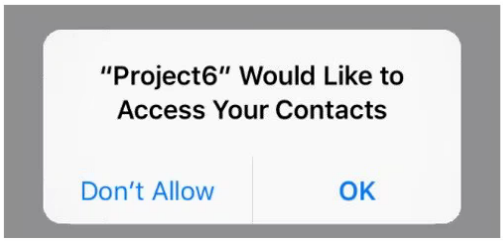
終於提供了許多開發人員多年來的要求，那就是一個跨平台的 TAddressBook 元件。

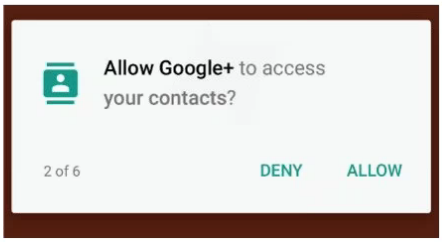
的新 TAddressBook 元件可以提供存取手機或平板的連絡資訊，並提供處理連絡資訊 CRUD 的能力，例如新增連絡人，新增連絡群組或是刪除連絡群組等功能:



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。

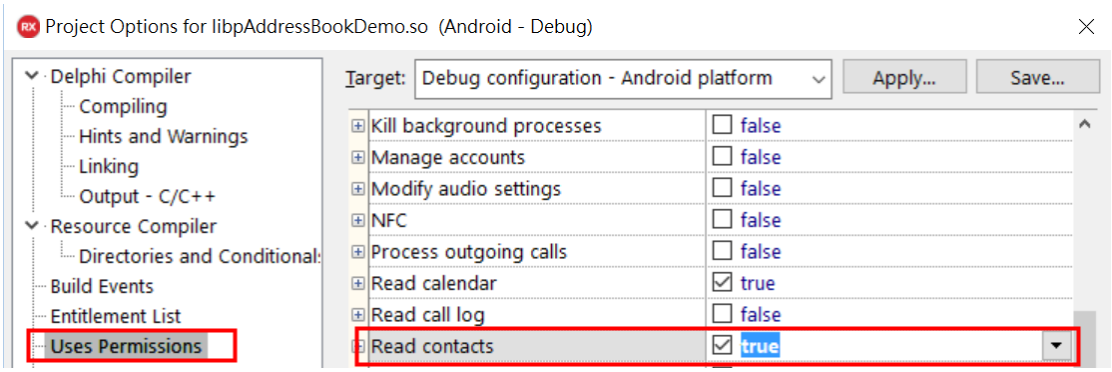
在使用 TAddressBook 元件存取連絡人資訊時開發人員需要設定存取權限否則無法存取連絡人資訊:





Permissions

在 C++Builder 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權限，之後會觸發 TAddressBook 元件的 OnermissionRequest 事件:

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

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。

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:

```
  TAddressBookContacts* __fastcall AllContacts(void);
```

AllContacts()方法回傳包含所有連絡人資訊的 TAddressBookContacts 物件：

```
class PASCALIMPLEMENTATION TAddressBookContacts : public
System::Generics::Collections::TObjectList__1<TAddressBookContac
t*>
{
  typedef
System::Generics::Collections::TObjectList__1<TAddressBookContac
t*>  inherited;
…
```

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：

```
class PASCALIMPLEMENTATION TAddressBookContact : public
System::TObject
{
  typedef System::TObject inherited;

protected:
  virtual System::UnicodeString __fastcall GetStringValue(const
Fmx::Addressbook::Types::TContactField AIndex) = 0 ;
  virtual void __fastcall SetStringValue(const
Fmx::Addressbook::Types::TContactField AIndex, const
System::UnicodeString AValue) = 0 ;
…
```


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。

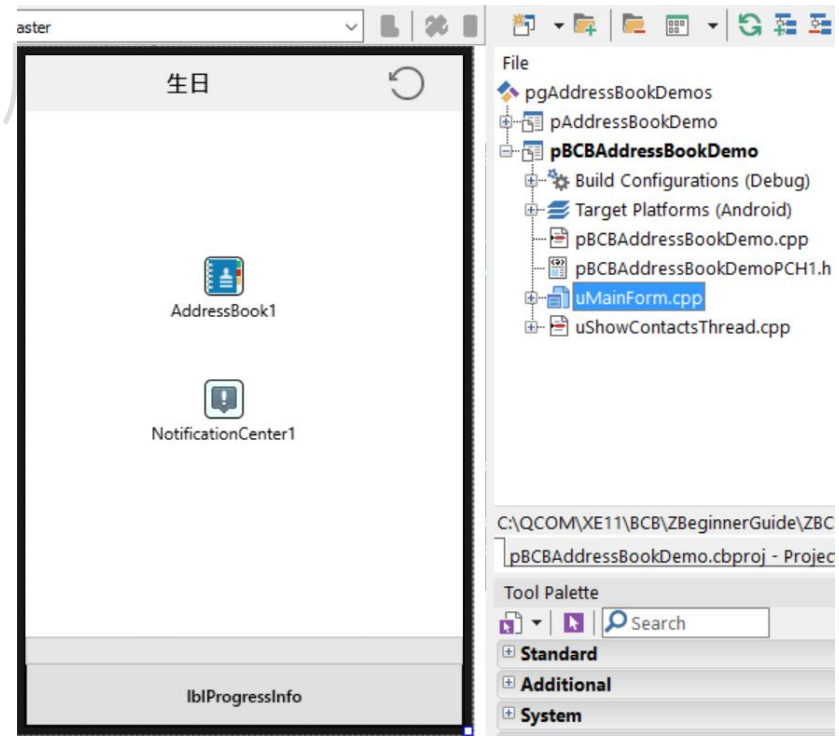
最後一旦取得了 TAddressBookContact 類別物件後，程式師就可以藉由存取它的特性值來取得連絡人資訊，例如連絡人名稱，連絡人 Email 和連絡人生日等重要的資訊：

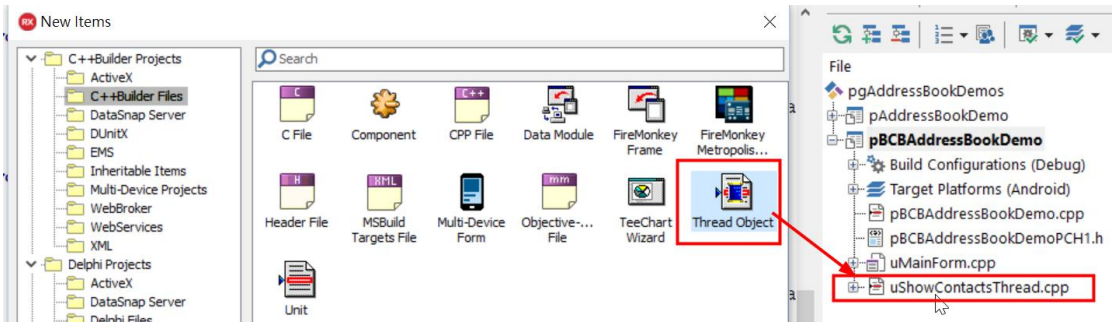
```
  __property System::UnicodeString DisplayName =
{read=GetDisplayName};
```

```
  __property Fmx::Addressbook::Types::TMultipleStringValues*
EMails = {read=GetListStringValue, write=SetListStringValue,
index=16};
  __property System::TDateTime Birthday = {read=GetDateTimeValue,
write=SetDateTimeValue, index=22};
…
```

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式的執行緒，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式的執行速度，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：





首先在主表單的 OnShow 事件中要求存取連絡人資訊的權限:

```
void __fastcall TfmMainForm::FormShow(TObject *Sender)
{
  AddressBook1->RequestPermission();
}
```

接著在 TAddressBook 元件的 OnPermissionRequest 事件中檢查是否取得權限，如果是的話就呼叫 FillContactsInfo 方法存取所有連絡人資訊:

```
void __fastcall
TfmMainForm::AddressBook1PermissionRequest(TObject *ASender,
const UnicodeString AMessage,
        const bool AAccessGranted)
{
  if (AAccessGranted)
  FillContactsInfo();
  else
  lblProgressInfo->Text = u"使用者不允許存取資訊!";
}
```


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：

```
void TfmMainForm::FillContactsInfo()
{
  delete FContacts;
  FContacts = AddressBook1->AllContacts();
  if ( (FThread != NULL) && (! FThread->Finished))
  {
```

```
  FThread->Terminate();
  FThread->WaitFor();
  }
  delete FThread;

  FThread = new TShowContactsThread(FContacts);
  FThread->OnContactLoaded = ContactedLoaded;
  FThread->OnStart = ContactLoadingBegin;
  FThread->OnTerminate = ContactLoadingEnd;
  FThread->Start();
}
```

TShowContactsThread 類別的建構元會儲存主執行緒傳遞來的 TAddressBookContacts 物件：

```
__fastcall
TShowContactsThread::TShowContactsThread(TAddressBookContacts
*AllContacts)
  : TThread(true)
{
  FAllContacts = AllContacts;
}
```


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：

```
void __fastcall TShowContactsThread::Execute()
{
  //---- Place thread code here ----
  Synchronize(DoStart);
  FIndex = 0;
  for (int iLoop = 0; iLoop < FAllContacts->Count; iLoop++)
  {
  FContact = FAllContacts->Items[iLoop];
  Synchronize(DoLoadContact);
  FIndex++;
  }
}
```

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：

本書到此也告一段落了最後祝您使用 C++Builder 開發 iOS/Android App 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