PowerBuilder開發系列（5）

PB .NET Assembly應用實例

文／圖　黃智祥

本系列文章最後一篇，介紹的是PowerBuilder 11基於.NET Interoperability技術，使得PowerScript能方便地與 .NET語言所開發出來的Assemblies互動，以Assembly，進一步利用.NET Framework的諸多功能，擴展PowerBuilder的開發能力。

支援.NET應用開發的PowerBuilder 11可說是近幾代PowerBuilder（以下簡稱PB）工具中特色最鮮明的一代，可能有許多人以這個表象來判定這不過是單純市場導向的決策，但筆者卻認為實質上有更為深層的涵義。其發展過程中，整個資訊技術的改變也持續在演進，就程式語言來說，期間包含了Java以及.NET Framework。相對於這兩個陣營，PB語言的獨立性，與其他語言串聯的擴充性卻成為其最大的煩惱。

PB為了走出侷限的格局，做到與其他語言所產出的程式碼溝通，擴大開發的應用領域，從ole、Web Services到PBNI的支援等，這些都是歷代PB開發工具改進的重點。到了PB11，最重要的意義就是基於.NET Interoperability（.NET程式碼互通性）技術，使得PowerScript能夠更方便的與.NET各種語言所開發出來的Assemblies（組件）互動。以Assembly形式存在的各種.NET應用程式碼檔案，不但靈活、統一，也可與功能豐富的.NET Framework底層程式庫互動，也就是說，PB利用Assembly，可進一步利用.NET Framework的諸多功能，以擴展PB的開發能力。

接下來，筆者將簡介何謂Assembly，以及在PB11中如何將NVO物件部署成Assembly，又如何在PB11內建的4個.NET相關Target中使用Assembly。最後將使用一個flicker網站為範例，說明PowerBuilder如何利用和Assembly的互通（Interoperability）關係，來和flicker網站互動，讀取網站的圖片內容資訊。

.NET Assembly技術結構

在正式介紹PB開發.NET Assembly之前，PB開發者最好對Assembly有一些基本的了解，接下來筆者將簡單的介紹何謂Assembly、Namespace以及Shared Assembly。

- Assembly：此為一個泛用的標準名稱，.NET規範所有基於.NET Framework開發出來的檔案（如：*dll）都叫做Assembly。正如其名，它是一塊塊的程式積木，對熟悉PB的開發者而言，Assembly的角色就如同PB中的PBL，內儲許多的class（類別），這些class中存在

許多的屬性及方法，提供給其他使用者呼叫。此外，不同的Assemblies彼此間還可以參考、呼叫對方的方法，因此只要合乎邏輯，這些程式積木可以拿來隨意做組合，快速地產出一個全新的應用。.NET Assembly範例程式可參考圖1。

```
1: using System;
2:
3: namespace DaveCo.Util
4: {
5:
6:     public class NullableDemo
7:     {
8:
9:         int? i;
10:
11:        public NullableDemo()
12:        {
13:            Console.WriteLine(i.HasValue);
14:            Console.WriteLine(i.Value);
15:
```

圖1　.NET Assembly範例。

除了.NET開發者們可以透過私人開發的Assembly分享程式碼外，更重要的是.NET Framework本身就是由許多Assemblies所組成的標準函式庫，其中包含大量類別，提供所有.NET應用開發所需的基礎功能，也因此.NET開發者只要熟悉.NET Framework，開發上的許多細節，例如：記憶體管理、檔案管理、資料庫連線控制等，只要呼叫Framework中的標準方法去處理即可，如此一來只要用很精簡的程式碼，便可以設計出多樣功能的程式。換句話說，.NET Framework與Assembly interoperability可說是.NET組合開發方法的實作基礎。

- Namespace：在 .NET Framework標準的函式庫中，這麼多的class分散儲存在不同的Assemblies中，為了使用、查找方便，.NET特別將所有class組織成一套具有層次結構的Namespace（立體命名空間），如圖2。它是一個與Assembly檔名獨立的虛擬名稱結構。簡單的說Namespace就是一棵.NET class的分類儲藏樹，開發者撰寫程式碼時，若欲使用某一個class下的方法或屬性，則必須呼叫其Namespace全路徑，亦即由樹根到樹葉的名稱，而不使用Assembly實體檔案名稱。例如：當開發者要引用File類別中的Copy方法，則必須在程式中指名System.IO.File.Copy（）才有效，如圖3紅色方塊是實際儲存在Assembly中的類別。

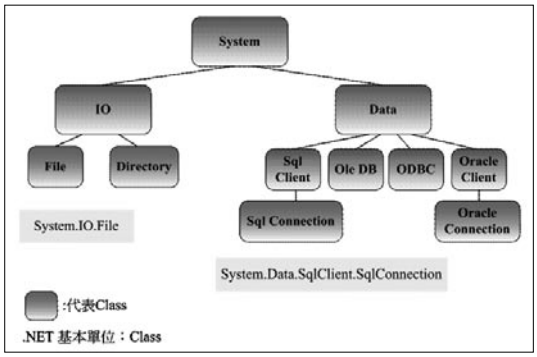

圖2　Namespace的樹狀結構。

圖3　實際儲存在Assembly中的類別。

Namespace存在最大的好處是方便開發者去查找需要的函式，微軟的MSDN已提供了這些函式詳盡的說明。在查詢MSDN類別使用方法時，開發者一方面可依常理邏輯去搜尋函式於Namespace中的位置，另一方面也能找到儲存該函式的class位於哪一個Assembly檔案中，如File class就是儲存在名為mscorlib.dll的檔案中，確認class所在檔案對Assembly的引用是很重要的，這在後面的文章中會說明。此外請注

意System是這個立體命名空間的基礎，而不是全部。

- Shared Assembly與Strong-Named Assembly：前幾期的文章裡我們曾介紹了GAC（Global Assembly Cache目錄：C:\WINDOWS\assembly；如圖4），這個目錄中存放的是該主機中能夠被共享的Assemblies群組，能夠存放在這個目錄中的Assembly有一個特別的名稱：Shared Assembly。其實.NET Framework就是一個典型的Shared Assembly集合，所有運作在這部主機上的.NET應用都能夠直接引用.NET Framework下Assembly的類別，而不需要額外在每個應用的目錄中去加入該Assembly檔案副本。但Shared Assembly有一個限制，它必須擁有全球唯一的識別名稱，或者說全球唯一識別性，而相對於Shared Assembly的Private Assembly則沒有這樣的限制，不過在使用上它必須緊密跟隨每一個引用它的應用程式作部署，比較麻煩。

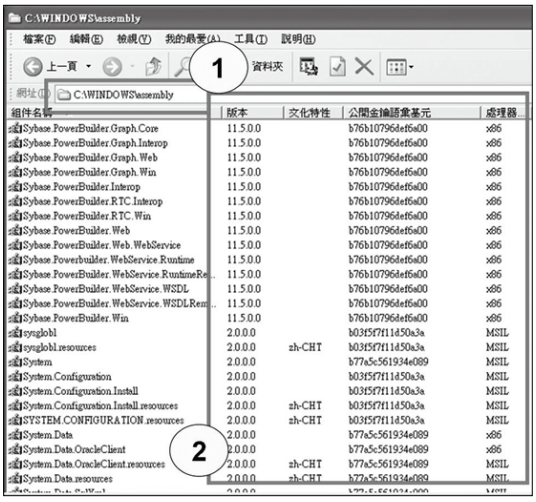

圖4　Global Assembly Cache目錄以及Shared Assemblies。

由於Assembly是可以被任意引用、組合的程式積木，因此如果有2個程式開發者不小心開發出檔名完全相同的Assemblies，則它們很容易被誤認為是有前後版本關係的Assembly，在公開的流傳使用上會造成很多困擾，因此Shared Assembly在開發時必須被賦予Strong Name簽名，來保證它的全球唯一識別性。

所謂的Strong Name其實是一個建立全球唯一識別碼的技術，它可以透過一組public-private key去產生一個全球唯一的識別碼內建在Assembly中，這樣一來便能夠幫助程式在載入時找到正確的Assembly。PB11.5所支援的Strong Name Assembly開發，筆者在稍後的文章中會做介紹。

在PB 11中建立.NET Assemblies

PB11建立Assembly的方式主要是將PB NVO物件透過PbtoCS編譯器轉成.NET Assembly，但整個開發上還是以PB傳統方式來進行。建立.NET Assembly共有3個步驟，依然是以精靈引導完成，以下順序介紹：

1. 新增.NET Assembly，如圖5。
2. 在NVO物件中加入程式碼，程式碼是以傳統的Power Scripts為主，如圖6。
3. 以.NET Assembly Project作部署，如同筆者前幾篇文章所述，PB11以統一的介面及流程

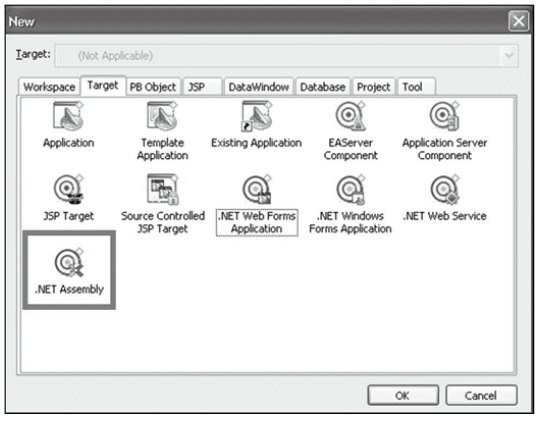

圖5　新增.NET Web Service Target。

圖6　建立Assembly的方法及屬性。

操作來達成部署的工作，PB開發者僅需要將相關屬性設定完成，就能讓Project自動完成Assembly的製作工作。

在.NET Assembly Project所有屬性頁中，最重要的是Object與Sign屬性頁。在Object屬性頁中，PB開發者必須決定有多少方法要加到這個Assembly中，一個Assembly中至少需加入一個方法，否則無法部署，如圖7編號3所示。此外讀者們也可以看到編號1的框中所設定的命名空間（Namespace）、Assembly檔名，以及編號2的類別名稱，都是我們先前所簡介Assembly時所提及的概念，值得注意的是PBL的名稱，在包裝時預設會被包裝成Assembly中的class name。

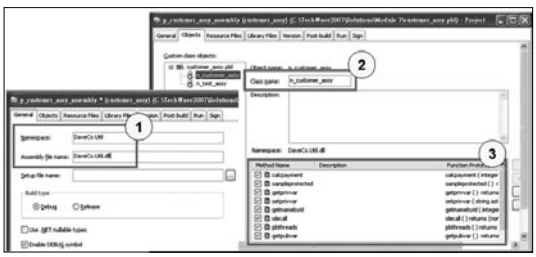

圖7　.NET Assembly Project部署設定。

Sign屬性頁（如圖8）規範了這個Assembly是否要部署成Strong-Named Assemblies，如同我們先前的介紹，擁有Strong Name便可以當作Shared Assembly，相關屬性項目介紹如下：

- Sign the assembly：勾選此屬性可以開啟下列其他屬性的設定，反之則其他選項都無法設定。
- Choose a strong name key file：這個屬性可以設定要附加到Assembly中的key檔，有兩個途徑可以加入副檔名為snk的key檔。若點選省略符號的按鈕（「...」），則會彈出一個瀏覽視窗，去選擇已存在的*.snk檔案；若點選「New...」按鈕，則PB會呼叫.NET Framework SDK中的SN.exe去產生一個新的.snk檔，而位於前端的Single Line Edit欄位則讓開發者可為這個key檔命名。
- Delay sign only：如果開發者希望Assembly的Strong Name簽名程序不要在開發環境中進行，而延遲至部署到runtime環境才執行，則可以勾選此選項。不過一旦這個選項被勾選，則無論run還是debug功能都無法執行。
- Mark the assembly with AllowPartiallyTrustedCallerAttribute：預設的情況下，Strong-Named Assembly只能被設定為完全執行權（full trust）的Assembly和其中的程式碼來呼叫使用，對於只有部分權限（partially trust）的Assembly和程式碼，則必須勾選此選項才能順利使用此Shared Assembly。

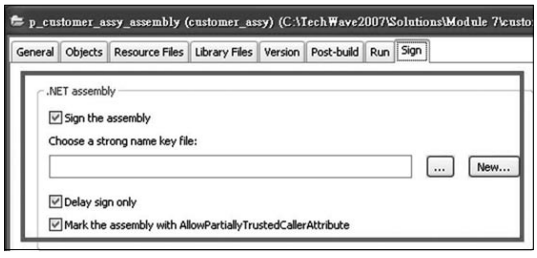

圖8　Sign屬性頁細部設定。

這裡須補充的是關於程式執行權限的設定，PB11.5可以透過設定CAS（Code Access Security）屬性，來限制程式在客戶端執行時的權限，如果程式被設定為禁止執行某些行為，

例如：IO，則稱為部分權限（partially trust）的程式碼。

執行部署後，一共會輸出4個部署用的檔案（如圖9左邊所示）。除此之外，開發者也可以直接將需要部署的檔案製作成.MSI檔以便自動部署，只要設定.NET Assembly Project General下的Setup file name屬性即可（如圖9右邊所示）。

圖9　PB .NET Assembly Project與產生的部署檔案。

PB11.5開發Assembly的能力，代表PB的程式碼能夠以.NET Assembly形式提供給它本身以外的語言服務，對使用不同開發工具的.NET開發團隊做出貢獻。

.NET Interoperability

PB11.5除了能夠開發Assembly，在它4個.NET相關的Target中也可引用.NET Assembly。接下來筆者將介紹PB開發者如何在這些Targets中去引用Shared Assembly和Private Assembly的方法，以及PB程式碼該如何與Assembly中的程式碼互相作用，即所謂的.NET interoperability。

- 引用Assembly：PB11.5在其.NET相關的Target中多了一個專門用來參考.NET Assembly實體檔案的屬性頁籤（如圖10），這個頁籤讓PB開發者把需要引用到的Assembly檔案放進來，其概念很類似於傳統PB C/S的Target把PBLs的路徑加到Library List中作為Library Search Path的概念。

這個頁籤中共有兩個主要的工具鈕，

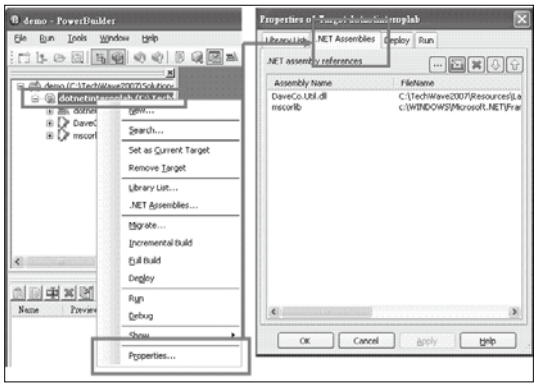

圖10　.NET Assemblies屬性頁籤。

分別可以引用Private Assembly與Shared Assembly。如圖11中編號1的部份為引用Private Assembly，而編號2的部份為Shared Assembly。讀者們需要注意的部份是，可參考的Assembly副檔名有dll、tlb、olb、ocx和exe，筆者先前提過Assembly是.NET應用程式檔案的統一名稱，*.dll僅僅是其中之一。

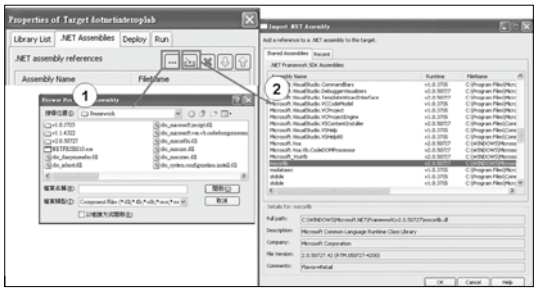

圖11　引用Private與Shared Assembly。

參考完成後PB開發環境的System tree中會出現以新圖形代表的Assembly物件，將之展開可以看到以樹狀圖顯示的Assembly以及其下的類別、方法。（圖12）

程式碼呼叫方式

完成Assembly實體檔案的參考，接下來就是直接撰寫程式呼叫使用。但使用Assembly的程式必須被寫在Condition compiler（條件式編譯區塊）區塊之中，這個區塊中使用的語法與

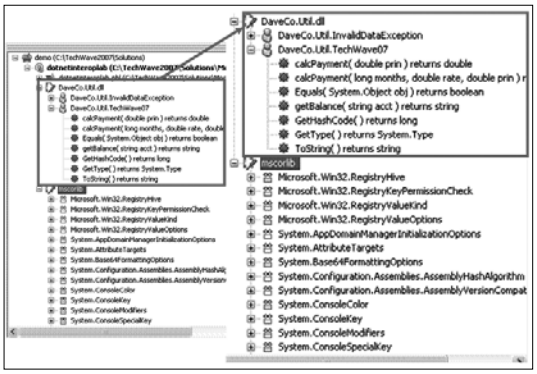

圖12　新的Assembly物件圖示。

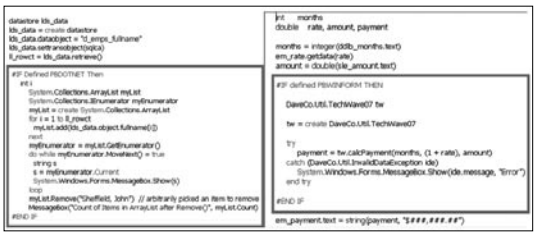

圖13　Conditional compiler中使用Assembly的語法。

傳統的PowerScript有些差異，但主要的架構並沒有改變，最主要的原因是為了要因應.NET Namespace的規則。

在圖13中讀者們可以看到傳統的PowerScript語法被寫在Conditional Compiler Area之外。Conditional compiler Area是一個以#IF Defined PBDOTNET開頭而以#END IF結尾的空間。其中容納.NET與PowerScript混合的程式碼，有幾件事值得注意：

1. 條件式編譯區塊外所宣告的變數可以直接帶進條件式編譯區塊中使用，相反的條件式編譯區塊中的變數也可直接帶出使用。
2. 條件式編譯區塊中仍然使用create關鍵字來建立某個類別的instance，但呼叫Assembly中的類別或方法名稱時，需要撰寫命名空間所預定的全路徑名稱。
3. 條件式編譯區塊中所使用的資料還是使用PowerScript標準的資料型態，但須注意呼叫Assembly中的函式時，若需要傳入參數或接收回傳值時，則必須依照PB/.NET Data type映射表來傳遞，如圖14表列所示。需額外注意的是陣列的宣告的語法還是PowerScript格式，如：string mystring[]，使用.NET格式的宣告方式，如：string[] mystring會造成部署時的編譯錯誤。

PB Data Type	Runtime .NET Implementation	.NET type used
any	Sybase.PowerBuilder.PBAny	object
blob	Sybase.PowerBuilder.PBBlob	byte[]
boolean	Sybase.PowerBuilder.PBBoolean	bool
char	Sybase.PowerBuilder.PBChar	char
date	Sybase.PowerBuilder.PBDate	System.DateTime
datetime	Sybase.PowerBuilder.PBDateTime	System.DateTime
decimal	Sybase.PowerBuilder.PBDecimal	decimal
double	Sybase.PowerBuilder.PBDouble	double
int	Sybase.PowerBuilder.PBInt	System.Int16
long	Sybase.PowerBuilder.PBLong	System.Int32
real	Sybase.PowerBuilder.PBReal	float
string	Sybase.PowerBuilder.PBString	System.Int16
time	Sybase.PowerBuilder.PBTime	System.DateTime
uint	Sybase.PowerBuilder.PBUInt	System.UInt16
ulong	Sybase.PowerBuilder.PBULong	System.UInt32
longlong	Sybase.PowerBuilder.PBLongLong	System.Int64

圖14　PB/.NET Data Type映射對照表。

4. 條件式編譯區塊中流程控制敘述句，例如If Then/Else、FOR Next、列舉式資料型態、例外處理（try/catch/finally）等語法，用的都是使用標準的PowerScript格式。
5. 條件式編譯區塊內的每一行程式皆不需以分號結尾，但如果單行程式碼過長而折到下一行則需要用&符號相連。

其他一些限制如下所述：

- 條件式編譯區塊內也可以容納傳統的PowerScript語法，但有某些限制須注意。PowerScript中的functions或events不能使用.NET的class、interfaces、structures，或列舉式資料型態作為參數或回傳值。
- PowerBuilder標準的NVOs、User Objects、Windows或Window Controls等，不能繼承自.NET的類別；此外，也不能implement .NET中的interface與其之下所包含的方法，只能直接使用。
- 如果有某個.NET的function執行時會回頭呼叫

圖15　Sybase官網上提供的UseFlickr範例。

PB的函式，則這個.NET function在條件式編譯區塊內會被禁止使用。

- 目前還不支援.NET Generic類別和以DYNAMIC或POST關鍵字去呼叫Assembly中的函式。

Useflickr應用範例

粗略了解PB11 Assembly開發與在程式中加入Assembly參考後，筆者最後以一個實際的程式碼範例來說明.NET interoperability概念的實踐，並藉此說明PB引用Assembly的意義。讀者們可以上Sybase官方網站的程式交換區（http://www.sybase.com/developer/codexchange）去下載此範例，如圖15所示。

這個程式範例是一個PB .NET Web Form應用。使用Flickr網站（http://www.flickr.com/）所提供的Assembly作為API，讓使用者開發具有照片搜尋功能的應用，下載檔案解壓縮後便可執行（其中有詳細的實作說明，故筆者在此省略）。該應用程式執行時的畫面如圖16，當使用者輸入欲查找的照片關鍵字後，被搜尋到的相關的照片集合即呈現在下方的DataWindow之中。

在圖17中，讀者們可以觀察到需要撰寫的程式碼並不多，主要的功能都是藉由參考FlickrNet.dll、System.Web.dll和System.Configuration.dll來完成。在這個範例中，PB開發者透過了Assembly的幫助，開發了過去本身無法支援完成的應用。

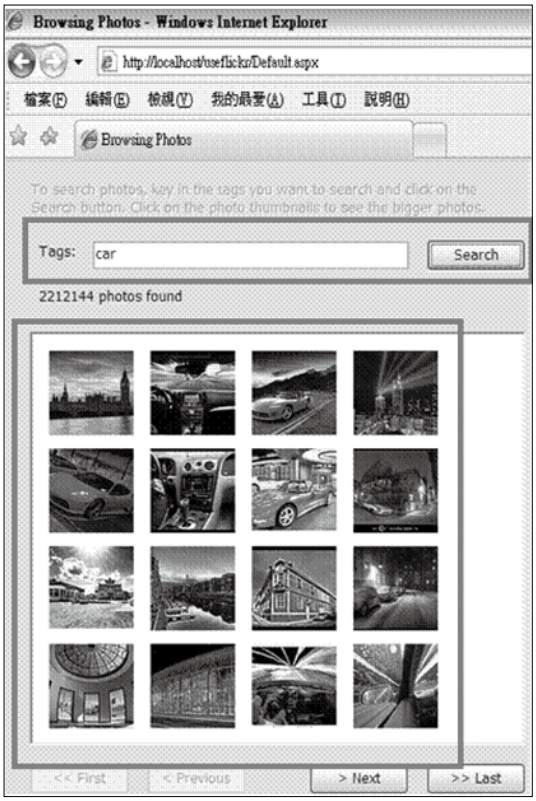

圖16　PB Web Form使用Flickr網站提供的API，製作出具有照片搜尋功能的應用。

圖17　UseFlickr範例程式的內容。

結語

本期文章是整個系列的最後一篇，回顧這

一系列文章，我們介紹了PB11如何維持了過去一貫的開發方式，卻以不同種類的Project和程式碼轉譯的策略去開發.NET Web Form、.NET Window Form、.NET Web Service和.NET Assembly等不同的應用，這些應用看上去各自獨立且互不相干，但Assembly與.NET interoperability技術告訴了我們，它們其實能被彈性地、完整地組合在一起，且集合不同應用特性的架構，往往更能開發出適合使用者實際需求的應用程式。

從本文最後的範例來看，我們至少可以發現：

- .NET interoperability對PB程式的助益：如今PB不但能夠開發出多元的應用程式，在許多PB原本不擅長的領域中，例如：Web應用領域，開發者現在也能夠借力使力，快速地完成開發工作。相對而言，PB的DataWindow也同樣藉由這條途徑，對更多需要它的應用做出更多的貢獻。
- .NET已成為主流的應用架構之一，因此大多數應用都會提供API給.NET應用。PB在這方面則可受惠於這樣的支援，而不必煩惱找不到適當的介面去使用許多優異的應用，同時PB也能較及時地支援更多的應用程式標準。

責任編輯／洪羿漣

作者介紹

黃智祥

目前服務於倍力資訊加值服務部經理，負責PowerBuilder系列產品講師。從早期的PowerBuilder 4.0開發人員，到現在負責PowerBuilder的技術推廣。曾在TPUG等多項活動中發表專題演說，撰寫過PowerBuilder 8.0分散式進階應用一書。專長為程式語言、物件導向技術，擁有SCJP、SCWCD以及IBM Rational等認證。

訊息子彈

嘉航科技 PTC Pro/ENGINEER Wildfire 5.0 發表會

新版的PTC Pro/ENGINEER Wildfire 5.0支援多種強化的功能，能協助使用者克服主要障礙並提昇設計產能。特別是新加入的數位人體工學建模解決方案（Manikin），可協助設計較佳的人機互動介面，無論是機械工程師、專業CAD繪圖師，或人體工學/人因工程分析師，這個新的模組皆可提供加速3D CAD細部流程的設計。喜歡嘗鮮的你，千萬不能錯過嘉航科技所舉辦的PTC Pro/ENGINEER Wildfire 5.0發表會！活動網址：www.jetsoft-tech.com；報名專線：0800-710-711陳小姐。

- **新竹　2009年8月11日**
  工研院51館2A教室（新竹縣竹東鎮中興路四段195號）
- **彰化　2009年8月18日**
  鹿港立德文教會館301會議室（彰化縣鹿港鎮中正路588號3樓）
- **台北　2009年8月20日**
  福朋喜來登飯店東廳（台北縣中和市中正路 631 號3樓）
- **桃園　2009年8月26日**
  中科院創意育成中心（W48館）第二會議室（桃園縣龍潭鄉龍園路134巷566號）