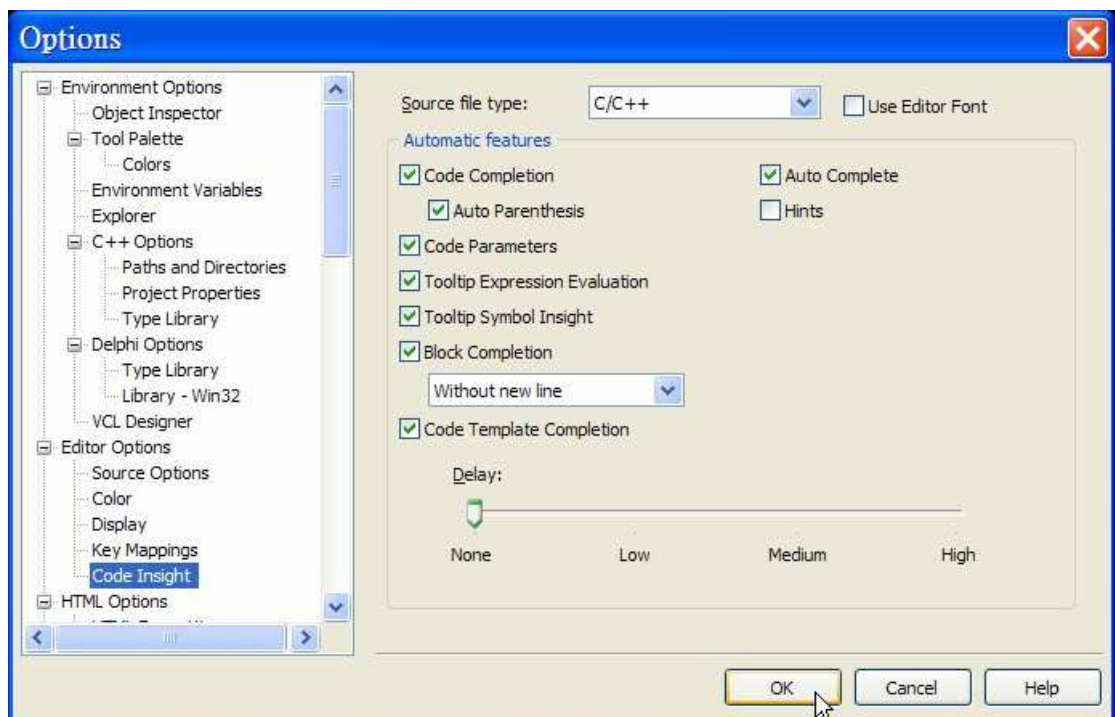


## 善用 C++Builder 2009 的 Pre-Compiled header 精靈

也許是我的記憶已經很模糊了，我記得在 C++Builder 5 時能夠提供了背景編譯的能力，允許 BCB 的開發人員在 IDE 中編譯專案時能夠啟動在背景編譯，如此一來 BCB 的開發人員就可以在等待 BCB 編譯專案的同時在 IDE 中執行一些其他的工作，BCB 之所以提供這個功能實則是因為 C++ 是一個 3-pass 的編譯器，因此需要遠多於 One-Pass 的 Delphi 編譯器更多的編譯時間。但當時 BCB 5 的背景編譯有許多的限制，例如開發人員無法異動正在編譯中的專案，也無法異動編譯專案的圖形使用者介面設計等，因此大部份 BCB 開發人員使用背景編譯編譯 BCB 專案時，大都是在 BCB 的 IDE 中對其他的專案進行同時開發的工作。

另外一個 BCB 非常重要的功能就是 Code Insight，這個功能能夠幫助開發人員大幅減少需要撰寫打字程式碼，進而增加開發的生產力。但是我知道很多 BCB 的開發人員關閉了這項功能，因為在早期的 BCB 版本中這個功能實在太慢了，導致許多 BCB 的開發人員抱怨為什麼 BCB 無法像 Delphi 的 Code Insight 一樣那麼的快速。其實 BCB 的 Code Insight 太過緩慢的問題我個人也是感同身受，因為我記得每次在做 BCB 的活動時，為了避免在 BCB 編輯器中撰寫程式碼反應太過緩慢的問題，我也都是關閉了 BCB 的 Code Insight 功能。



最後一個我要討論的問題就是 BCB 的 Pre-Compiler Header 了，雖然 BCB 很早就提供了 Pre-Compiler Header 功能以加快編譯速度，但老實說早期 BCB 提供的 Pre-Compiler Header 雖然的確能夠幫助開發人員加快編譯速

度，但這個 Pre-Compiler Header 在 C++Builder 2009 之前已經有數年沒有改善了，因此仍然有很大的進步空間。

OK，看到這裡您可能會想，為什麼同時敘述上述的三個問題呢？是它們有什麼共點嗎？不然這篇文章到目前看起來是令人摸不著頭緒的。OK，現在就讓我們回到本篇文章的正題。

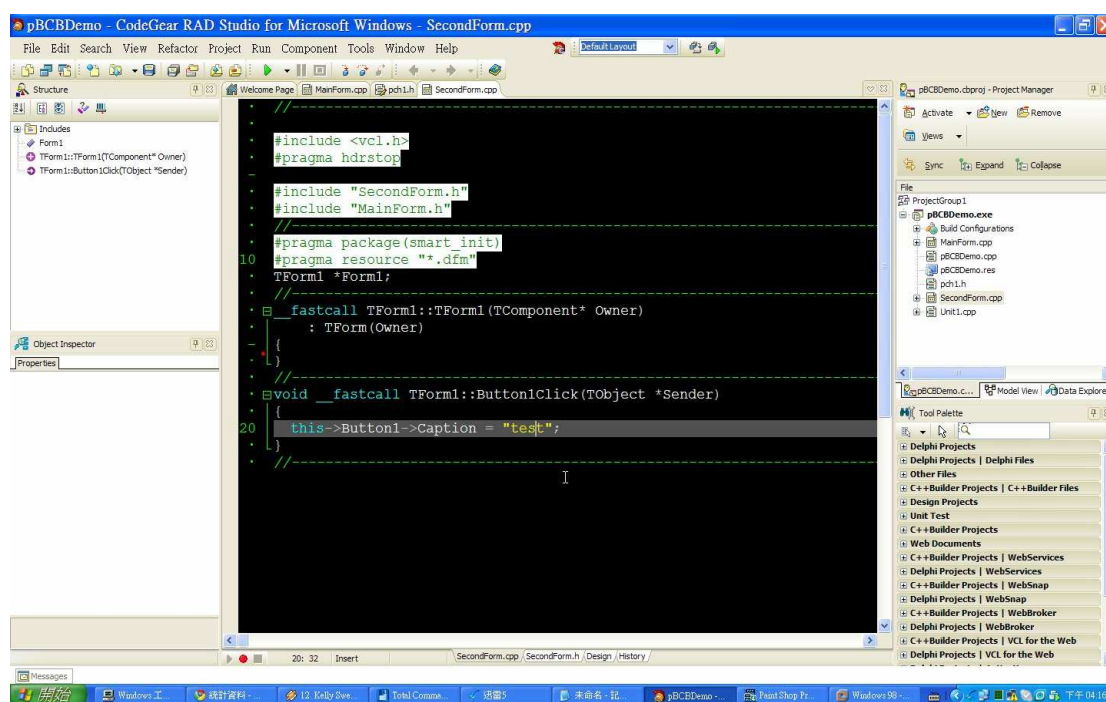
下圖是我在 RAD Studio 2009 中的一個範例 BCB 專案，在沒有使用新的 Pre-Compiled header 精靈之前，在第一次編譯這個 C++Builder 專案時 RAD Studio 仍然會為這個專案建立 Pre-Compiled header，如此一來當開發人員稍後再次編譯這個專案時，編譯速度就會加快。例如如果我修改下圖中的

```
this->Button1->Caption = "test";
```

為

```
this->Button1->Caption = "測試";
```

接著再次編譯，那麼此時 C++Builder 2009 需要再次編譯 **21281** 行的程式碼，雖然整個編譯速度算是相當快速，但仍然令人奇怪，為什麼只是修改一行的程式碼卻需要編譯 21281 行。



其實仔細觀察原始程式就可以發現問題所在，因為在原始程式中存在如下的程式碼：

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

```
#include "SecondForm.h"  
#include "MainForm.h"
```

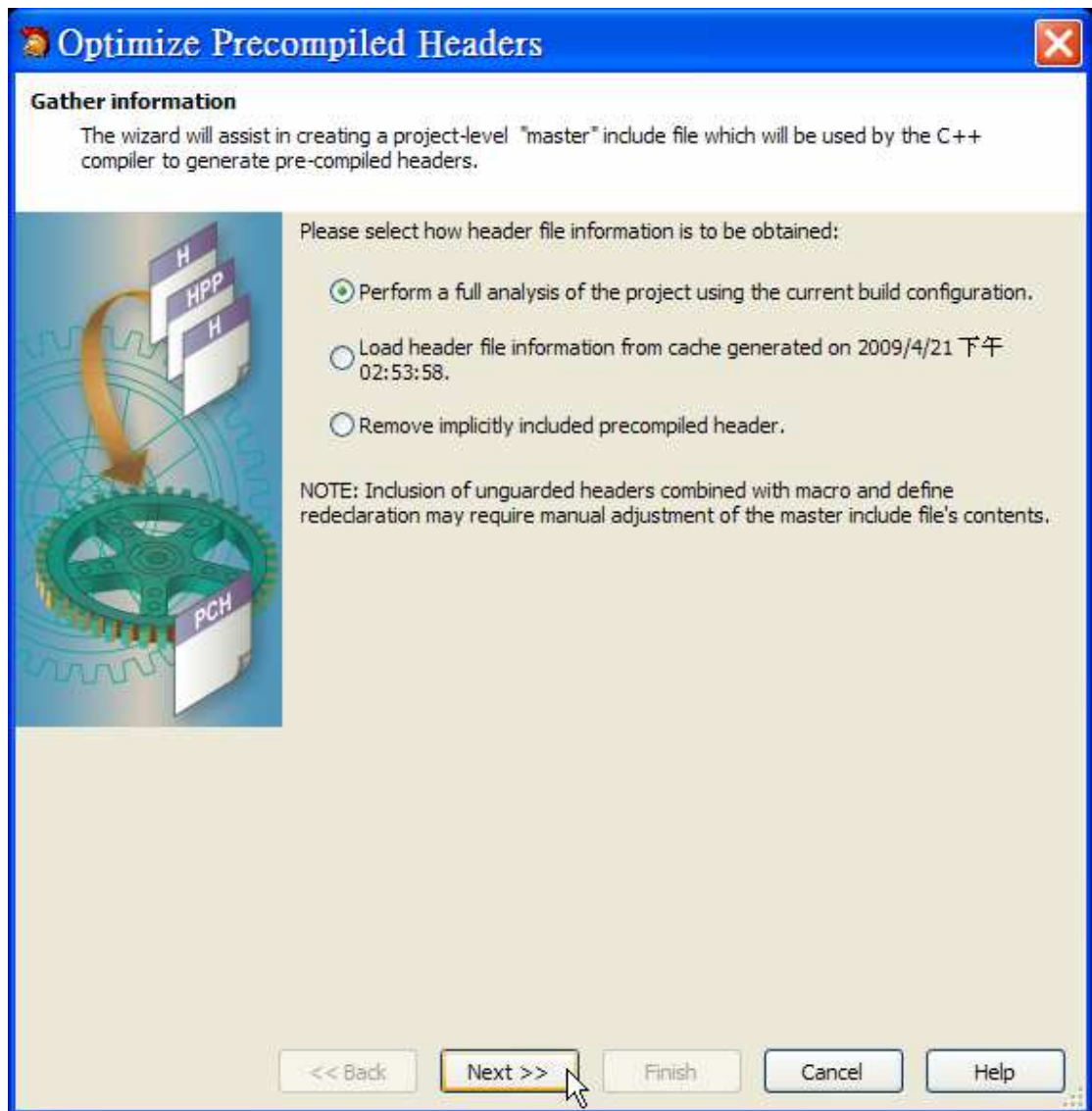
舊的 **BCB Pre-Compiled header** 精靈只把 **VCL.H** 相關的程式碼預先編譯，但再看看原始程式的表頭檔，我們卻發現了下面的程式碼：

```
#include <Classes.hpp>  
#include <Controls.hpp>  
#include <StdCtrls.hpp>  
#include <Forms.hpp>  
#include <DB.hpp>  
#include <DBCtrls.hpp>  
#include <DBGrids.hpp>  
#include <ExtCtrls.hpp>  
#include <Grids.hpp>
```

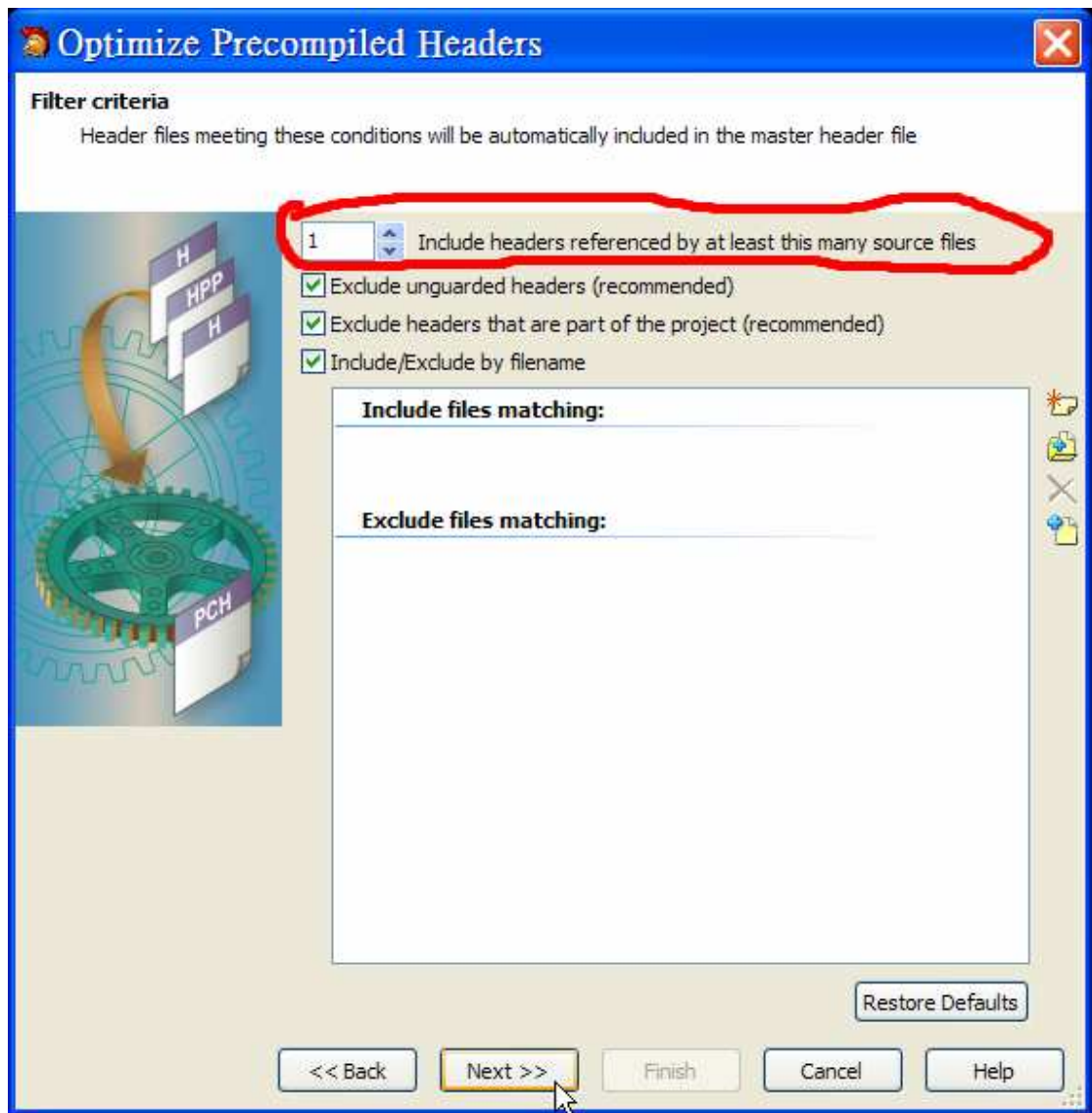
問題似乎就出現在這裡，因此我們需要一個能夠分析整個專案的 **Pre-Compiled header** 精靈幫助我們產生最佳的預先編譯表頭資訊，並且能夠讓開發人員進行客製化的調整。

**C++Builder 2009** 新的 **Pre-Compiled header** 精靈就可以幫助我們完成這些工作，讓我們現在就來看看它的功效如何。

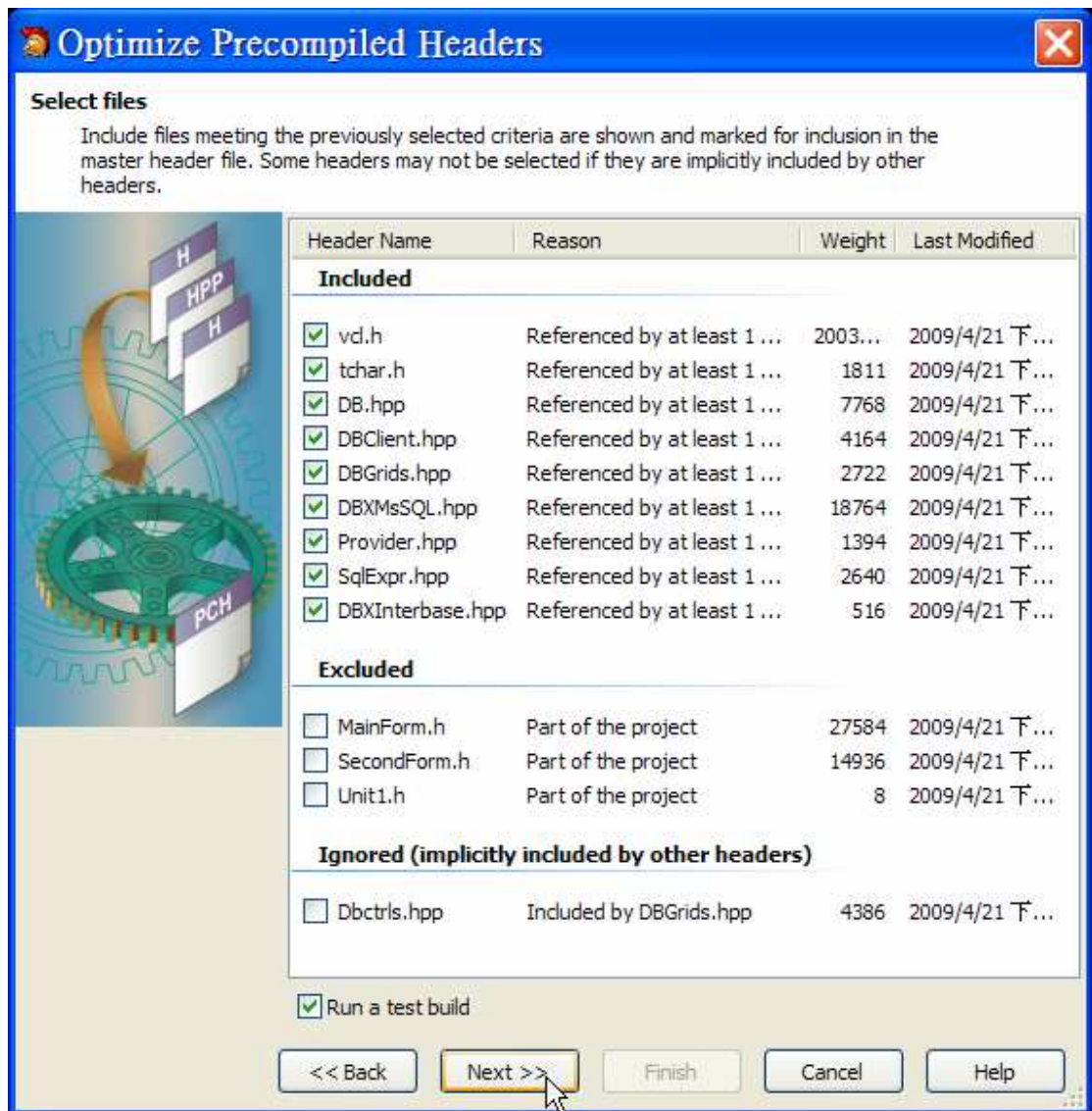
首先點選 **Tools | Pre-Compiled header Wizard...** 功能表啟動精靈，如下圖所示，如果你是第一次使用這個精靈，那麼請選擇第一個選項為專案進行一次完整的分析：



在精靈成功分析之後，它會顯示如下的對話盒，詢問你客製化的設定，例如你可以設定要包含那些表頭檔或是排除特定的表頭檔。



點選『Next>>』按鈕之後會顯示最後精靈進行的設定，它決定了預先編譯表頭檔中包含的資訊：



繼續點選『Next>>』按鈕，最後 BCB 會產生一個新的表頭檔 pch1.h，例如在我的範例中最後的 pch1.h 包含如下的資訊：

```

/*
  This precompiled header include file was generated on 2009/4/21 下午
03:00:09
  by the RAD Studio Precompiled Header Wizard with the following settings:

Project: G:\MyBogs\20090421\Demos\pBCBDemo.cbproj
AllowUnguarded = 0
ExcludeProjectFiles = -1
IncludePathsOn = -1
IncludePaths =
ExcludePaths =

```

```

IncludeCount = 1
ManageHeader = -1
*/

#ifndef pch1_H
#define pch1_H
#include <vcl.h>
#include <tchar.h>
#include <DB.hpp>
#include <DBClient.hpp>
#include <DBGrids.hpp>
#include <DBXMsSQL.hpp>
#include <Provider.hpp>
#include <SqlExpr.hpp>
#include <DBXInterbase.hpp>
#endif

```

在精靈產生了 `pch1.h` 並且加入到專案之中後，請先進行一次完整的專案 `Build` 讓 `BCB` 編譯器建立新的預先編譯表頭檔。有了新的預先編譯表頭檔之後如果我再回到前面修改：

```
this->Button1->Caption = "測試";
```

為

```
this->Button1->Caption = "第 2 次測試";
```

接著選擇 `make` 專案，那麼現在 `BCB` 只會編譯 **86** 行的程式碼，比使用舊的預先編譯方式快了好幾倍的速度。

更棒的是，使用新的 `Pre-Compiled header` 精靈之後由於它會產生最佳化的表頭資訊，因此此時如果你再使用 `BCB` 的 `Code Insight`，你會發現 `Code Insight` 快速的不得了，例如在我的機器中此時再使用 `BCB` 的 `Code Insight` 功能，它的反應速度幾乎和 `Delphi` 的 `Code Insight` 一樣快了，現在我再也不需要關閉 `BCB` 的 `Code Insight` 功能了。

BTW，`RAD Studio 2009` 的 `Update 3` 不但修改了許多的臭蟲，`IDE` 的速度又加快了不少，例如對於相同的 `BCB` 專案，使用 `Update 3` 的編譯速度硬是比 `Update 1` 和 `Update 2` 又快了許多。

OK，解決了預先編譯和 `Code Insight` 的問題之後，最後就是 `BCB` 的背景編譯了，在 `C++Builder 2006/2007/2009` 中背景編譯都被移除了，因為舊的背景編譯限制太多，在 `BCB 5` 那時沒有多核 `CPU` 的時代，舊的背景編譯架構

也無法利用現在最新的多核 CPU，不過我知道下一版的 BCB 也許將提供全新的背景編譯技術，不但速度快，限制又少，甚至可以讓開發人員在背景編譯專案的同時又進行相同專案的持續開發，這對於大型的 BCB 專案來說實在太棒了。

試著想想，未來在 BCB 中我們將可結合預先編譯表頭，背景編譯和快速的 Code Insight，那麼使用 BCB 來進行 C++ 專案的開發將會非常的愉快，因為如此一來 BCB 即可讓 C++ 的開發人員享受類似 Delphi，C# 和 Java 等快速開發的環境了。