推薦序 I

在電腦運算發展的歷程中，「人機介面」毫無疑問扮演了關鍵性的角色。猶記得當初做學生的 70 年代必須靠著讀卡機與電算中心裡冷氣玻璃房後可望不可及的電腦，一遍又一遍辛苦的送件溝通，然後望穿秋水的期盼它能印出正確的報表與結果。然而在接下來的這幾十年中，“Dumb Terminal” 與鍵盤、視窗與滑鼠、瀏覽器 Hypertext 介面、手寫筆、觸控螢幕，以劃時代的姿態陸續登場，每一項的創新與演變都拉近了人與電腦的距離，擴展了電腦的使用族群，驅動了嶄新的應用。

大家漸漸理解到，電腦本質上是服務人群的工具，過去以人遷就電腦所能接受的溝通方式是既不得已也不合理的。正確的方向應該是讓電腦來迎合人們日常所熟悉的溝通方式，於是乎，使用者經驗（User Experience）與自然人機介面（Natural User Interface）成為當今研發與創新的顯學。

比爾蓋茲很早就提出了電腦能聽、能看、能想的願景，同時微軟研究院也長期默默從事這方面的研究與創新。Kinect 裝置可以說是一個最令萬眾矚目、能夠達到商業化與普及化的具體成果與里程碑。從網路上，我們看到各式各樣讓人驚艷、引人莞爾且意想不到的創新應用，如果再過幾十年後回頭看，在自然人機介面的領域，它可能就像我前述的讀卡機時期一般陳舊。可以預見，Kinect 中的各種技術將會持續快速發展，並且跳脫出單一的機體，融入各式各樣的智慧型裝置 – 汽車、家電、娛樂、教育、醫療、通訊……等，也正因為如此，它是我們當今專業資訊朋友們以及青年學子，必須開始瞭解、體驗與開發創新應用的關鍵性技術。

劉念臻
台灣微軟 開發工具暨平台推廣處總經理

本書的作者王森，相信對眾多的程式開發人員都不陌生，而我有幸與他在微軟共事逾五年。在認識他之前，我對他的印象是位活躍的資訊社群意見領袖、文章書籍作家、評書專家。在接下來朝夕相處的多年中，我更發現他還是個幽默風趣、觀察力敏銳、做事極嚴謹、自我要求極高的好同事好夥伴。不出手則已，一旦出手必定全力以赴。當我翻閱本書這麼完整與周延的內容與範例時，腦中所浮出的景象是遠在屏東傳說中“Moli 城堡”裡，無論豔陽高照揮汗如雨或夜涼如水挑燈夜戰，他的身影埋在堆積如山的技術資料與自行架設的各種電腦設備中，反覆的構思、撰寫、測試、驗證與校對這本他精心策畫已久的書籍。

我衷心的祝賀他如願完成這本書。更期待所有的讀者，能夠在吸收 Kinect for Windows 的精髓與技巧後，能夠開發出令我和大家都耳目一新的創新應用！

推薦序II

我還是懵懂的大學生時，就已經拜讀過王森許多專欄、技術教學的文章，他對於我踏入資訊科學、軟體開發的領域有著很深遠的影響，那時怎麼也想不到有一天我會為他的著作寫序，也是一種「造化弄人」吧！（笑）

Kinect 的出現，顯示微軟一直在為整個資訊科技的下一個可能性做投資與研究，微軟研究院（Microsoft Research）於 2010 年在家用主機 Xbox 360 上推出了 Kinect for Xbox 360，將遊戲的操作帶向新的領域，在其他遊戲主機還要玩家拿著特殊的裝置才能完成體感操作時，Kinect 已能直接辨識人的骨架、關節位置、手勢來完成操作，這也使得愈來愈多人對它產生興趣，開始研究如何將它應用於其他領域、降低原本操作的難度，或是以全新的思考方式來設計操作介面，許多技術高手開始以各種方式分析出 Kinect 的訊號，發展各式各樣的 SDK 來開發軟體。

同一時間，微軟也提供了由官方開發的 SDK 及 Windows 上的驅動程式，讓有興趣針對 Kinect 特性來開發軟體的開發人員，能夠在官方的支持下用穩定及一致的開發環境來開發 Kinect 軟體，發展至此，Kinect 不僅只是能夠接上 Xbox 360 主機做娛樂用途，更能連接上 PC 實現更多的可能，而有了 Kinect for Windows 的名詞誕生。

要開發 Kinect for Windows 的相關應用，除了要瞭解 Kinect 裝置的能力、差異（除了 Xbox 360 主機專用的 Kinect，目前也有市售的 Kinect for Windows 裝置，支援更近距離的偵測）、SDK 的用法之外，還要瞭解

上官林傑
台灣微軟　應用開發技術經理

影像音訊處理、電腦視覺等專門知識，本書在作者近乎龜毛的要求之下，不但開宗明義便讓讀者瞭解 Kinect 裝置的細節、SDK 的架構，更花上許多篇幅在介紹色彩、影像、視覺等專業知識，使讀者在唸完本書之後，都能立即開始設計 Kinect for Windows 的應用程式。

雖然 Kinect for Windows 的 SDK 發展時間不算長（目前最新的正式版本為 1.5），但我們已經能夠看到許多有趣的應用，像是醫療復健、手術訓練、教育互動……等，這都證明了只要您有想法，目前的 Kinect 裝置及 SDK 都能幫助您實現夢想。期待閱讀此書的讀者，都能創造出改變世界的軟體及應用！

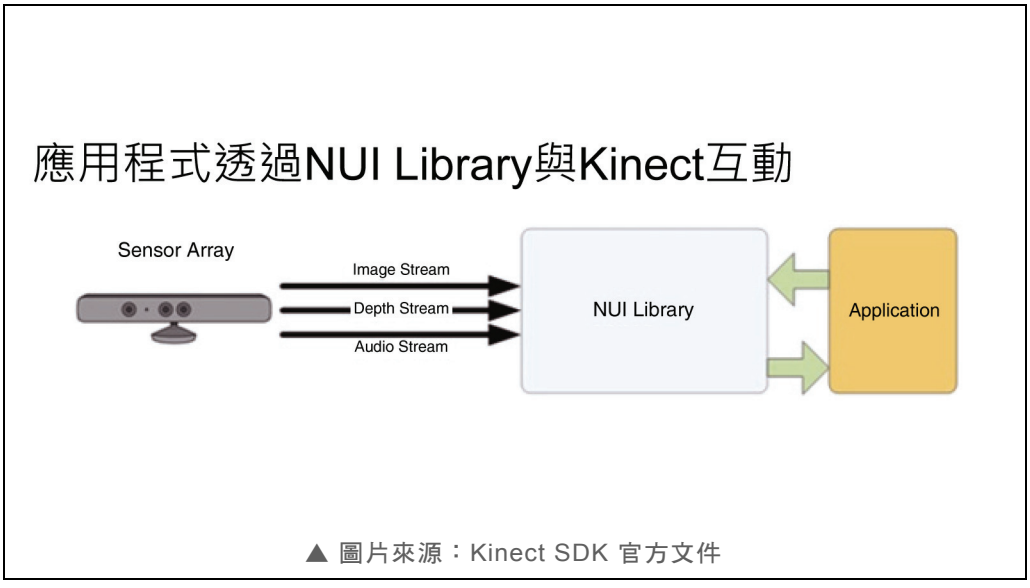
Kinect 開發初體驗

04

在本章中，我們將學習如何利用 Developer Toolkit Browser 觀看範例並下載範例原始碼，並練習 Kinect Studio 的使用。

此外，我們會分別使用 C# 與 C++ 做最簡單的 Kinect 裝置控制 – 改變俯仰角（又稱 Tilt、Elevation Angel），這些範例除了用來控制 Kinect，也簡單示範了一個穩定性夠強的 Kinect 應用程式應該具備的基本條件。

4-1 操控 Kinect



▲ 圖片來源：Kinect SDK 官方文件

Kinect 應用程式皆透過 NUI Library 操控 Kinect 所有相關設定參數，並藉由 NUI Library 擷取 Kinect 感應器中的資訊，包括：

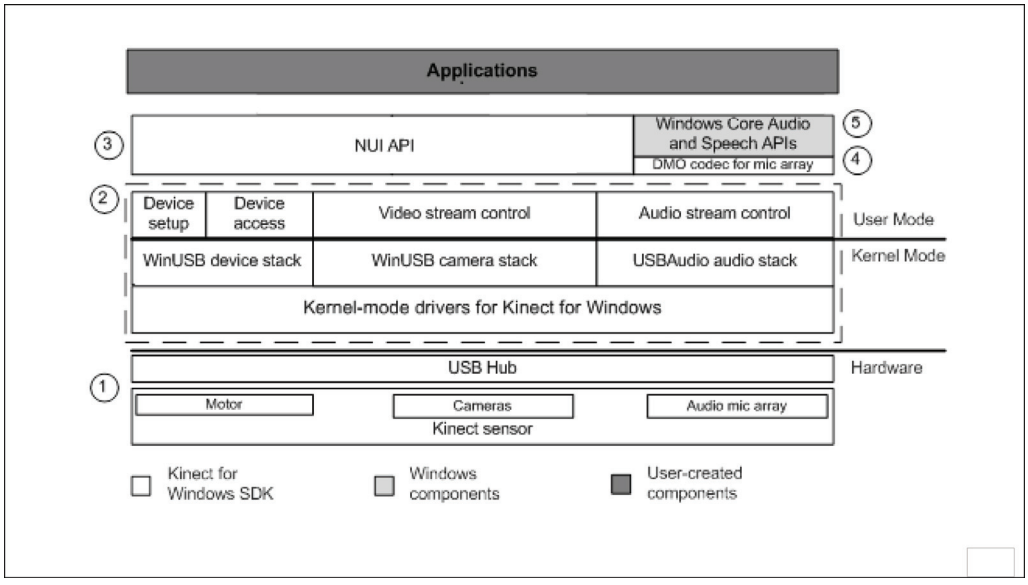
彩色影像資訊（Stream，串流）

深度影像資訊（Stream，串流）

聲音資訊（Stream，串流）

C++ 應用程式直接連結 NUI Library（動態連結函式庫），.NET 應用程式則使用一個名為 Microsoft.Kinect.dll 的組件存取 NUI Library。

簡單來說，Microsoft.Kinect.dll 組件對 NUI Library 進行了簡單的重組，處理了低階資料型態與 .NET 資料型態的差異，讓 .NET 能用純物件導向的方式與 Kinect 互動。



上圖之中，標示為 1 的區域為 Kinect 硬體，透過 USB 與 PC 連結。標示為 2 的區域為驅動程式部分。標示為 3 的區域為高階的 NUI Library。

值得注意的是，標示為 4 的部分，DirectX Media Object（DMO）為 DirectX 既有的組成分子，被 Kinect 拿來作為接收聲音訊號與分析聲音來源之用。Windows 7 內建 DirectX，因此 DMO 本身就存在於我們的機器之中，並非 Kinect SDK 所安裝。

標示為 5 的部分為標準的 Windows 7 API，用來處理聲音，語音等這些與多媒體相關的應用，本身就內建於 Windows 7 之中，並非 Kinect SDK 所安裝。相關的 API 說明可以在 Windows SDK 與 Microsoft Speech SDK 之中找到。

從這張圖可以了解到，在本書中提到聲音處理與語音辨識的範例，多半**只是把 Kinect 作為一個普通麥克風（音源輸入）使用**，就算沒有 Kinect，平常 PC 所使用的麥克風一樣可以讓這些範例正常運作。

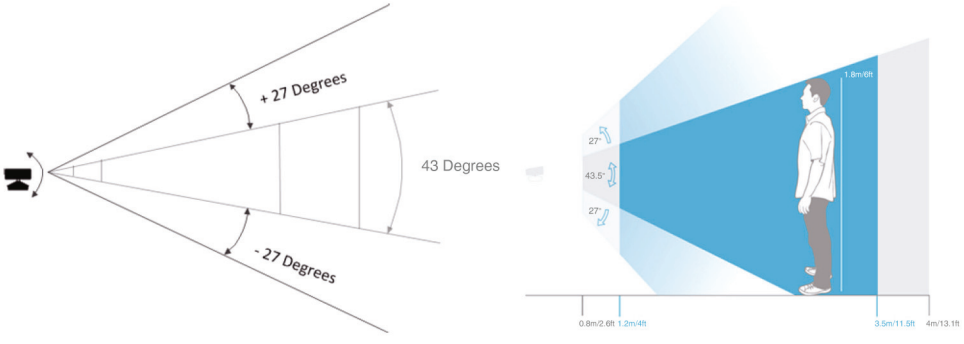
4-2 Kinect 硬體特性

感應項目	有效範圍
顏色與深度	0.8 ~ 4 公尺 (近距離模式 0.4 ~ 3 公尺)
視野角度	水平 57 度、垂直 43 度
底座馬達旋轉	上下各 27 度
每秒畫格	12、15、30 FPS (可由API改變)
深度解析度	VGA (640 x 480) QVGA (320 x 240) (80x60) 皆為4:3
顏色解析度	XSGA (1280 x 960) VGA (640 x 480) 皆為4:3
聲音格式	16KHz, 32位元 單聲道 PCM(Pulse Code Modulation)
聲音輸入	麥克風陣列、24 位元類比數位轉換 (ADC)、 聲學迴音消除(AEC)`、自動增益功能(AGC)、抑制噪音

▲ 硬體規格

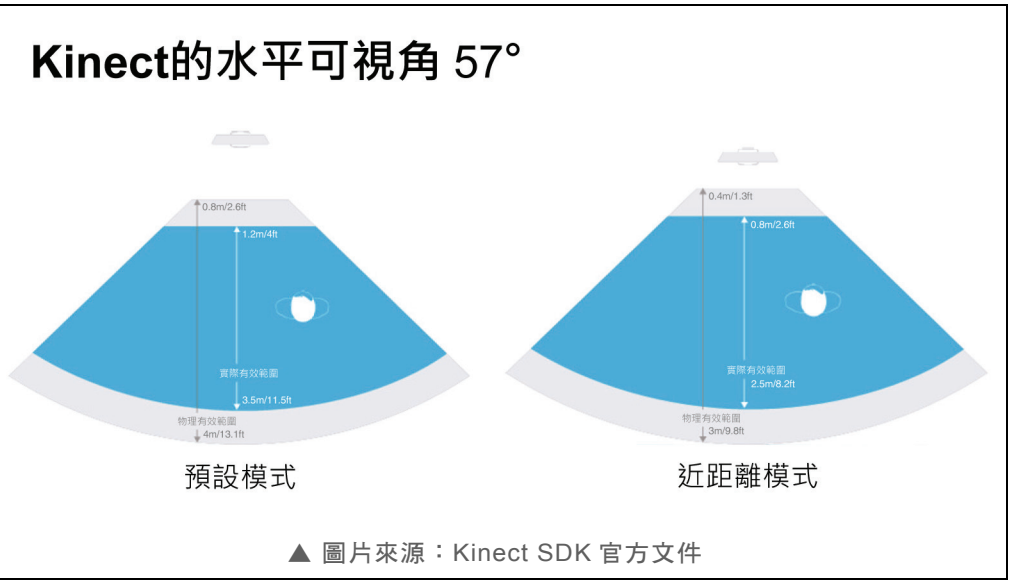
Kinect的垂直可視角43°

Kinect的Tilt(傾斜)角度
俯角27度(-27°)　仰角27度(+27°)



▲ 圖片來源：Kinect SDK 官方文件

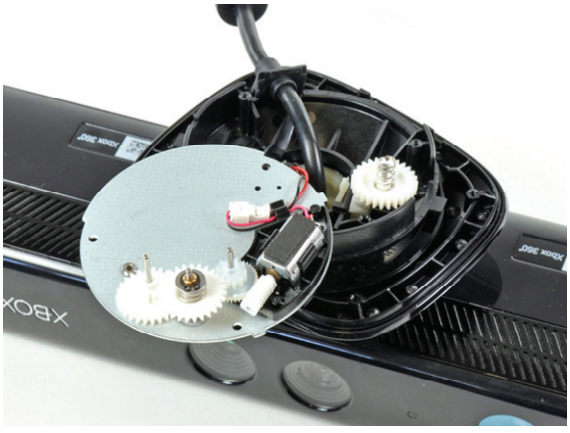
我們可以用 Kinect 改變 Kinect 感測器的垂直擺動角度（Tilt），由上圖來看，藉由上下各 27° 的變化，Kinect 在垂直方向上，大概可以看見共 27+43+27 = 97° 。



▲ 圖片來源：Kinect SDK 官方文件

Kinect 的水平可視角為 57°，大約是感應器為中心，左右各 28.5°。

Kinect的Rotate(旋轉)角度

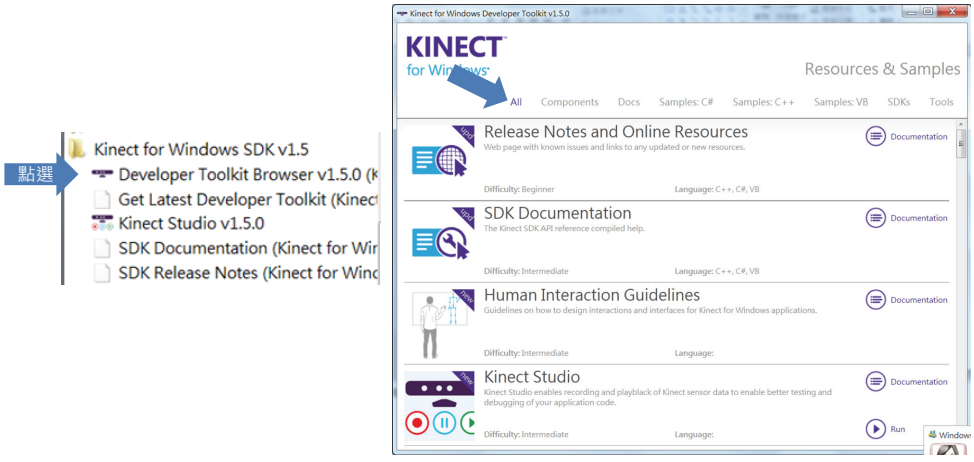


▲ 圖片來源：http://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/4066/1

關於水平可視角，最常見的誤解就是以為 Kinect 具有水平轉動的功能，這個誤解通常來自於網路上這張 Kinect 底座的拆解圖。實際上，**並沒有任何 API 可以控制 Kinect 做水平轉動**。

如果您曾經在 Xbox 上使用 Skype 搭配 Kinect 進行語音通話，其臉部追蹤功能也會讓人誤以為 Kinect 可以水平轉動，其實該功能是利用數位變焦的方式所達成，在 PC 上透過臉部追蹤函式庫我們一樣可以做到。

4-3 使用 Developer Toolkit Browser



▲ 開啟 Developer Toolkit Browser

Developer Toolkit Browser 開啟後，預設顯示 All 分頁。由於 All 分頁會把所有 Developer Toolkit Browser 內附的東西全部一起展示出來，建議直接切換到適當的分頁，比較容易在短時間內找到我們所需要的東西。

Component 分頁：語音辨識用的 Language Pack、可以簡化 .NET 應用程式開發的套件

Docs 分頁：連接 MSDN 線上文件、也可以下載到一份在 Kinect 人機互動上相當重要的一份文件 – Human Interaction Guidelines

SDKs 分頁：各種執行範例程式所需要的額外 SDK，像是 DirectX 相關 SDK、XNA 相關 SDK、Microsoft Speech Platform。

Tools 分頁：可以直接執行 Kinect Studio 和前面章節所提到的 Kinect Explorer。其中，Kinect Explorer 還提供原始碼供我們參考。

▲ 範例程式

範例程式分別提供 C#、Visual Basic(VB)、C++ 三種。以數量來說，C# 的範例最多最完整（Kinect Explorer 本身就是 C# 所撰寫），C++ 次之，VB 最少。VB 的範例全部皆以 WPF 撰寫，C++的幾乎全部都需要搭配 DirectX 進行開發。C#的範例除了 WPF 之外，也有結合 XNA 的例子。

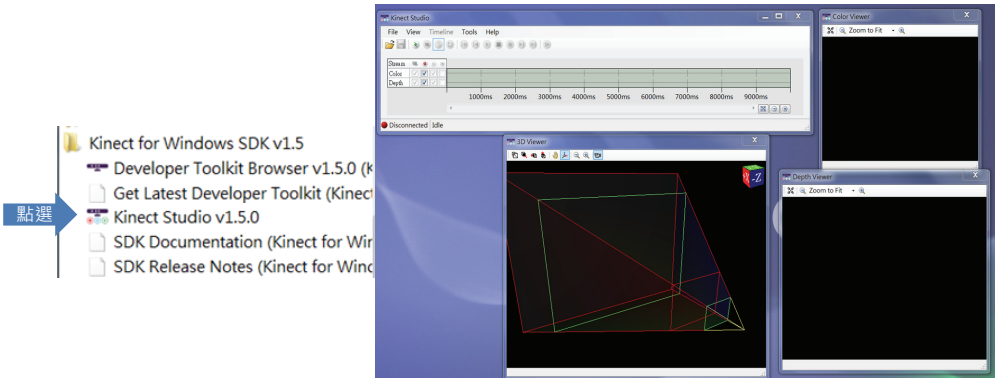
每個範例的右手邊如果出現

Documentation 圖示，代表可以連結到 MSDN 網站上關於範例的說明。

Install 圖示，按下後將詢問您想要將該範例原始碼放在何處，存放後可用 Visual Studio 開啟，自行編譯與執行。

Run 圖示，代表可以直接執行。如果沒有 Run 圖示，代表並沒有預先編譯好的執行檔，我們必須先 Install 原始碼之後自行編譯與執行。

4-4 使用 Kinect Studio

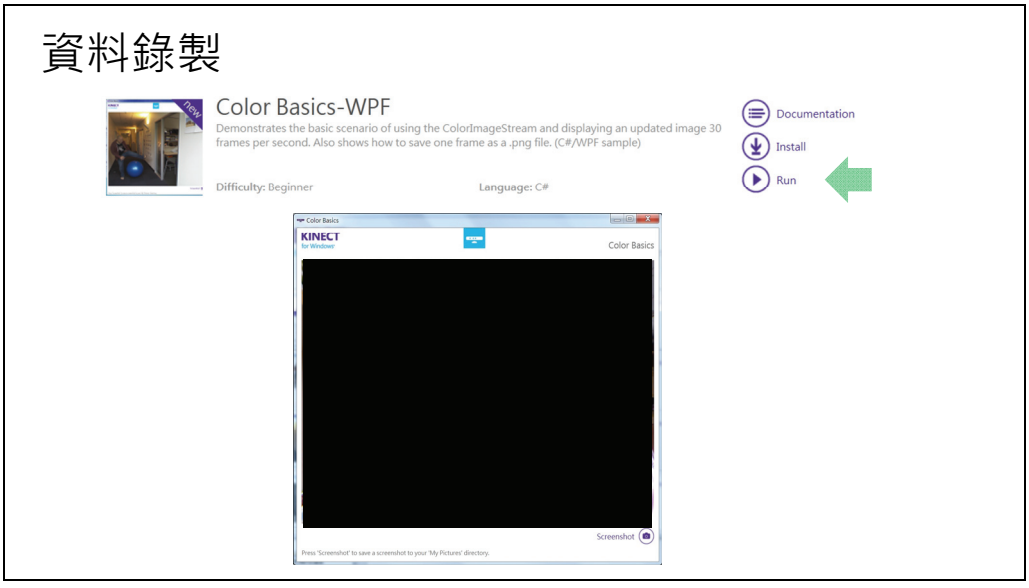


▲ 開啟 Kinect Studio

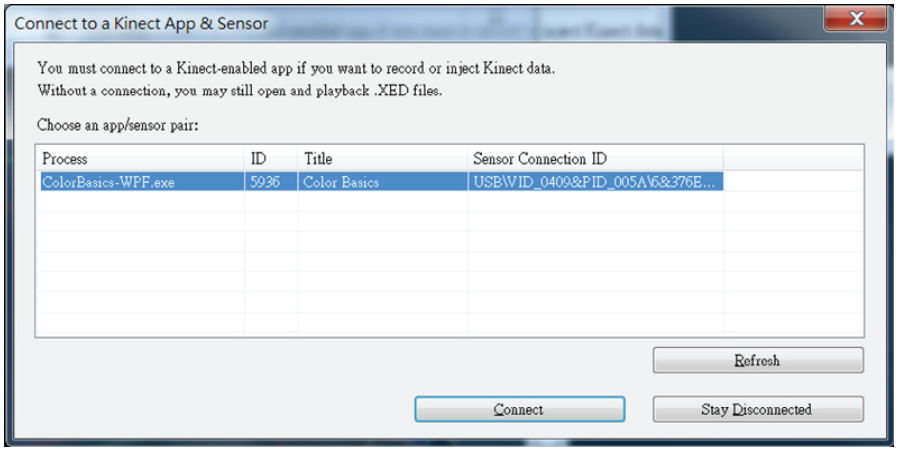
Kinect Studio 主要用來錄製 Kinect 回傳的彩色影像、深度影像、聲音等資訊，最主要的用途在於測試與分析資料。

在測試的用途上，我們可以先請人在機器面前擺出我們希望他們所擺出的各種姿勢和動作，然後用 Kinect Studio 錄製好各種測試案例。之後即使當事人不在場，我們一樣可以把測試案例作為資料輸入我們的 Kinect 應用程式中，再反覆調整（最佳化）應用程式。

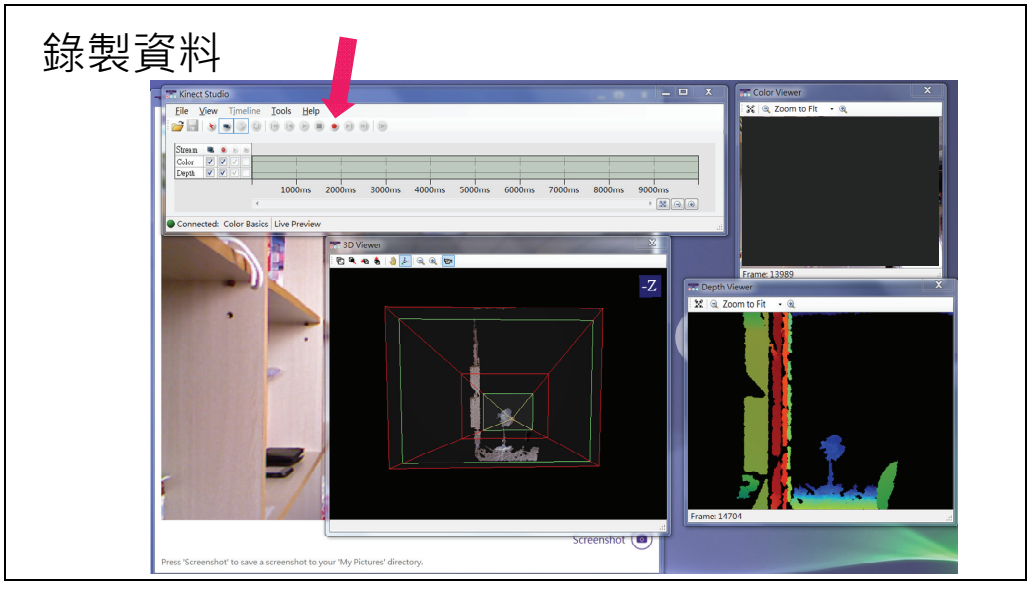
在分析資料的用途上，Kinect Studio 可以讓我們停留在錄製資料的每個時間點，分別讓我們以彩色影像（Color Viewer）、深度影像（Depth Viewer）以及 3D 空間（3D Viewer）觀看資料的內涵，當我們在設計各種演算法時（尤其是手勢動作的設計），對每個時間點使用者所對應到的各種資料，都有助於我們設計更好的演算法。



Kinect Studio 必須和某個 Kinect 應用程式連接之後，才能進行資料的錄製。這裡我們以範例的 Color Basics-WPF 做練習。

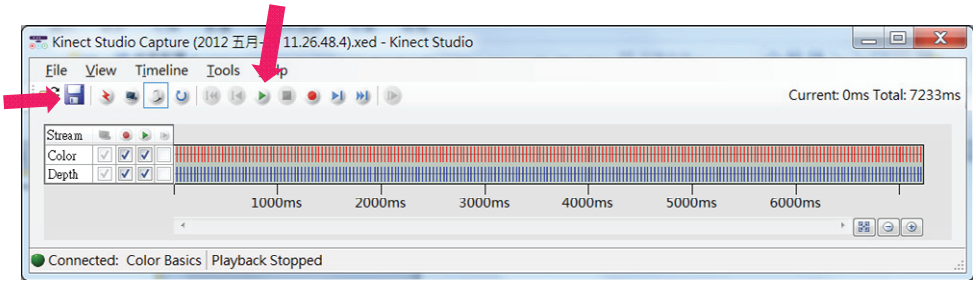


Kinect Studio 會自動抓到所有相關的 Kinect 應用程式（如果沒看到請按 Refresh），選擇之前啟動的 ColorBasics-WPF.exe 接著按下 Connect 即可。



按下上方的紅色按鈕（Record）開始錄製資料。

重播

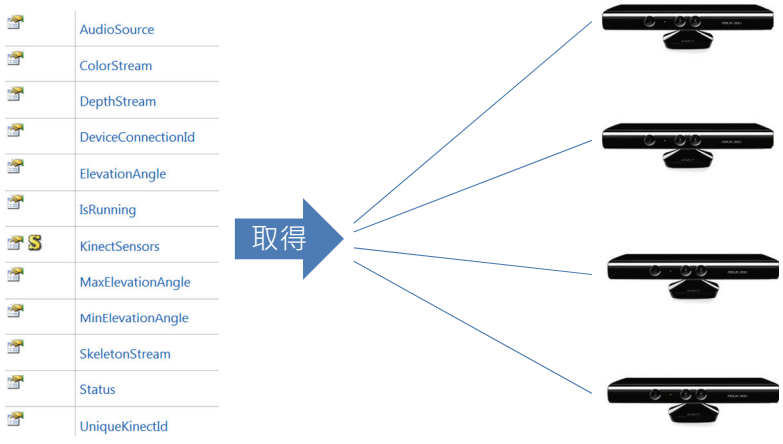


按下綠色箭頭（Play/Pause）就可以重新播放。您可以直接在時間軸上選擇特定的時間範圍播放，也可以要求同時間只單獨播放彩色影像或深度影像。

請按下磁碟片圖示（Save）儲存剛剛所錄製的資料，資料的附檔名為 .xed。接著關閉 Color Basics-WPF 範例，改成啟動 Depth Basics-WPF 範例，並將其連結 Kinect Studio。

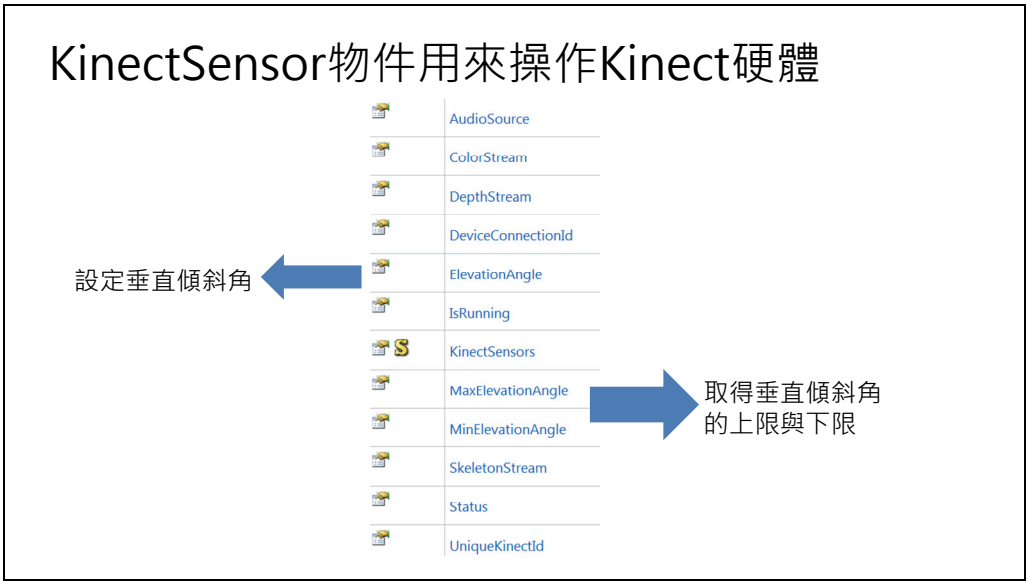
使用 File ➔ Open（或資料夾圖示）開啟之前儲存的 .xed 檔，您將發現剛剛所錄製的資料其實與特定應用程式本身無關，這些一樣資料可以用來測試其他 Kinect 應用程式。

4-5 Kinect 硬體與 .NET 的連結



▲ KinectSensor 物件

要從 .NET 控制 Kinect 硬體，第一步就是從 KinectSensor 物件的 KinectSensors 中取出所有連接到 PC 上的 Kinect 裝置。



一旦取得代表每台 Kinect 硬體的 KinectSensor 物件，我們就可以利用其屬性來控制相關參數，或是從 Kinect 硬體取得彩色影像、深度影像、聲音等資料。在本章中，我們利用 MaxElevationAngle 得知硬體的最大仰角上限，MinElevationAngle 得知硬體的最大俯角上限（雖然官方數據皆為 27 度，請務必不要在程式中寫死，可以增加未來的彈性）。我們也會設定 ElevationAngle 來改變 Kinect 感應器的俯仰角。

4-6 程式範例 - 控制 Kinect 垂直角度（使用 C#）

- 連接Kinect Sensor
- 控制Kinect Sensor的俯仰角
- 增加應用程式穩定度
  - 邊界值處理
  - Kinect硬體保護

▲ 練習重點（使用 C#）

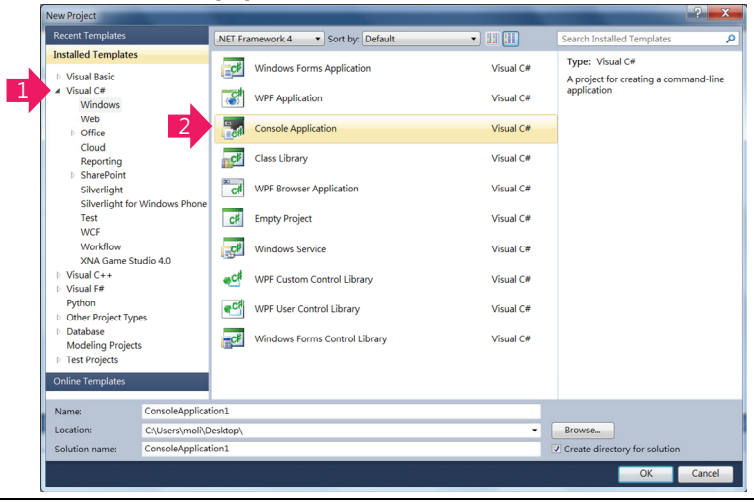
本範例將嘗試用最簡單的方法連結應用程式與 Kinect。

連接完成之後，我們會呼叫 Kinect SDK 標準函式庫改變 Kinect 的俯仰角。

藉由用程式控制俯仰角的改變，可以學習到一些基本的 Kinect 程式設計觀念。

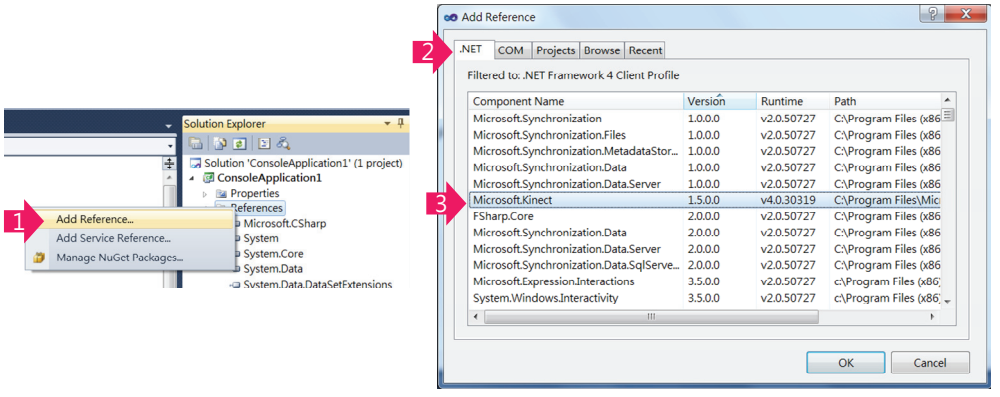
請先打開 Visual Studio 建立新專案:

建立Console Application專案（C#）



建立 Console Application 專案（C#），輸入專案名稱後按下 OK 即可。

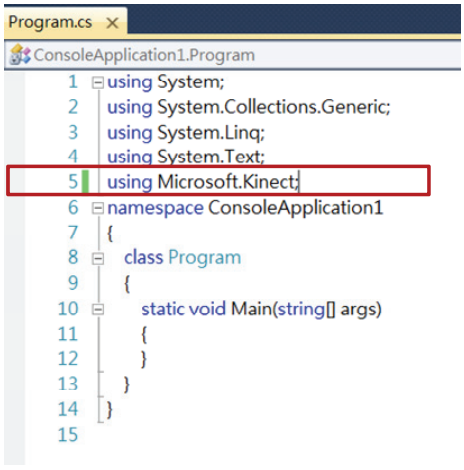
增加Kinect組件參考

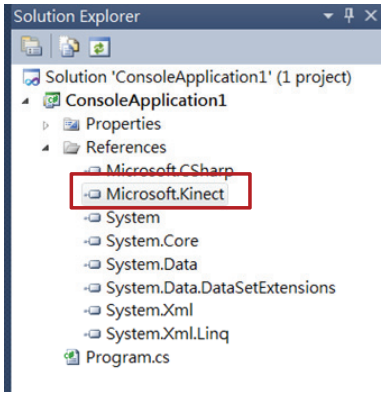


在專案底下的 References 節點按下滑鼠右鍵，選擇 Add Reference，進入 .NET 組件選擇畫面。

按下上方的 .NET 分頁，點選名為 Microsoft.Kinect 的組件（Assembly）即可。

加入命名空間（Namespace）





如果組件參考設定正常，就可以在 References 節點底下看到名為 Microsoft.Kinect 的組件。在我們的原始碼 Program.cs 中，加入

```
using Microsfot.Kinect;
```

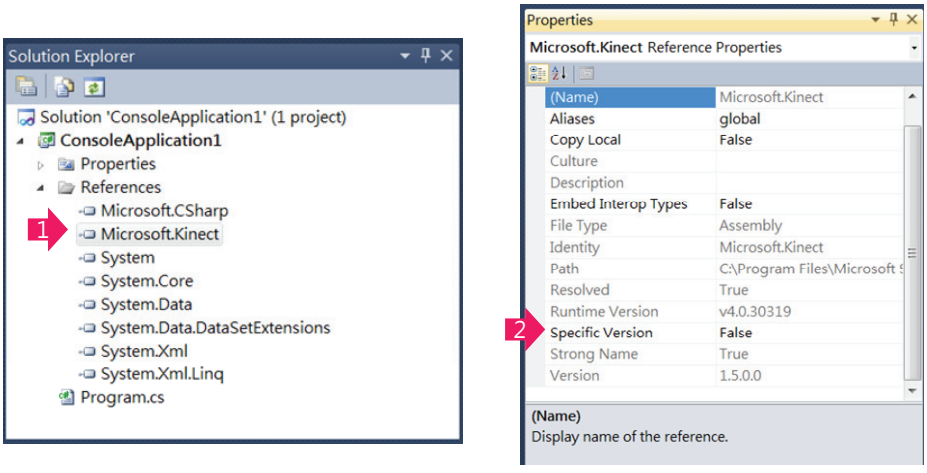
指令，如此基本的設定大功告成。

任何應用程式，只要完成上述兩個步驟：

1. 加入 Microsoft.Kinect 組件
2. 加入命名空間 Microsoft.Kinect

就能夠連結 Kinect 並控制它。

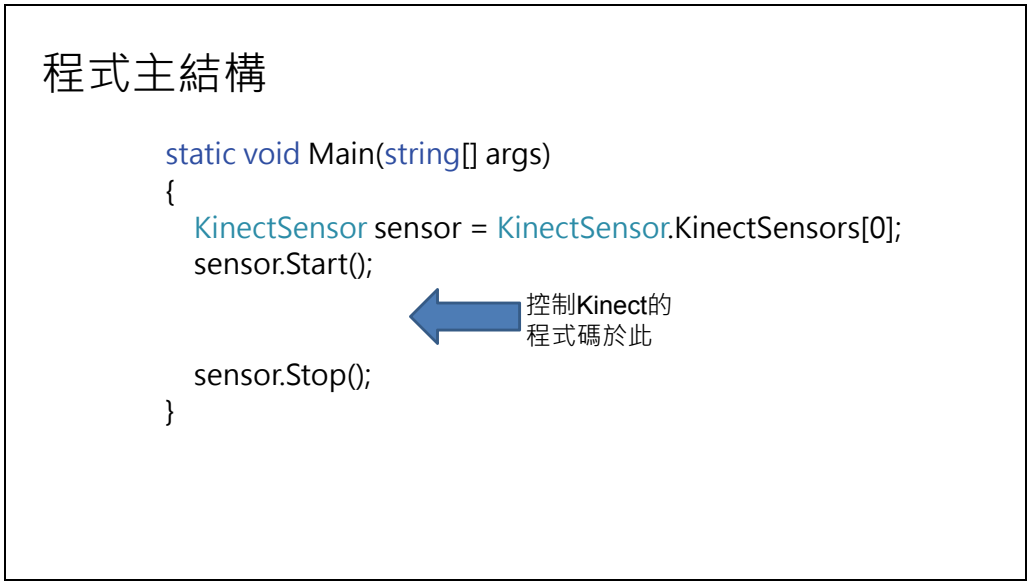
組件的版本問題



在預設的情況下，Visual Studio 會認定您的應用程式需要倚賴特定版本的 .NET 組件，這會造成組件改版時(例如安裝了新版 Kinect SDK)，有可能找不到特定版本的組件導致無法編譯。這個時候您可以點選 Microsoft.Kinect 組件，然後將 Properties 視窗中的 Specific Version 改為 False（預設為 True），就可以避免上述問題。

舉例來說，如果您曾用過 Kinect SDK 1.0 撰寫程式，當您安裝了 Kinect SDK 1.5 之後，Visual Studio 會顯示找不到版本為 1.0.0.0 的 Microsoft.Kinect 組件，只要您將 Specific Version 改為 False，Visual Studio 就會自動找到版本號碼為 1.5.0.0 的 Microsoft.Kinect 組件。

不過，**這個方法僅適用於相容的組件（Kinect 1.0 與 Kinect 1.5 完全相容）**，如果遇到不相容的不同版本組件，需重新設定組件的參考，這裡介紹的方法無法解決問題。



我們使用

```
KinectSensor sensor = KinectSensor.KinectSensors[0];
```

取得連接在 PC 上的第一個 Kinect 感應器。這個方法並非最好的方法，但是最簡單。這行程式假設 Kinect 感應器已經安裝在 PC 上，因此執行時，如果 PC 上沒有任何進入可用狀態的 Kinect 裝置，將引發例外。

取得 KinectSensor 物件後，應該先檢查是否為 null，再繼續底下工作。如果不是 null，接著我們可以利用 Start() 方法啟動 Kinect 感應器，程式結束前，請利用 Stop 方法關閉 Kinect。

設定Kinect Sensor俯仰角

```
static void Main(string[] args)
{
   KinectSensor sensor = KinectSensor.KinectSensors[0];
   sensor.Start();

   sensor.ElevationAngle = 0;

   sensor.Stop();
}
```

我們利用

```
sensor.ElevationAngle = 0;
```

這行指令，將 Kinect 的角度設定為 0，也就是硬體出廠時的狀態。

有些人拿到的 Kinect 硬體時，不一定處於此狀態，因此會用暴力的方式手動改變其角度，這種方法有可能會傷害硬體，建議使用此程式輕鬆地恢復出廠狀態。

您可以試著改變角度，重新執行程式，請注意此數值的有效範圍為 -27 度到 +27 度，任何超過有效值的設定都會引發例外。

動態改變Kinect Sensor俯仰角度

```
static void Main(string[] args)
{
   KinectSensor sensor = KinectSensor.KinectSensors[0];
   sensor.Start();

   sensor.ElevationAngle = 0;  // 一開始先設定成水平0度

   ConsoleKey presskey ;
   while ( (presskey =Console.ReadKey().Key ) != ConsoleKey.Spacebar)
   {
      if (presskey == ConsoleKey.DownArrow)
      {
         sensor.ElevationAngle = sensor.ElevationAngle - 5;
      }
      else if (presskey == ConsoleKey.UpArrow)
      {
         sensor.ElevationAngle = sensor.ElevationAngle + 5;
      }
      Console.WriteLine("現在角度：" + sensor.ElevationAngle);
   }

   sensor.Stop();
}
```

利用 .NET 提供的函式庫，我們可以根據使用者所按下的按鈕即時改變 Kinect 的狀態。

上述範例程式可以在使用者按下方向鍵的上與下時，以每次變化 5 度的方式改變 Kinect 的俯仰角，使用者按下空白鍵時則結束程式的執行。

您會發現有時候角度改變之後，Kinect 並沒有到達我們所要求的角度，而且通常會有正負 1 度的差距，這是因為 Kinect 使用步進馬達，底層無法精確地達

到某個特定的角度，由於 Kinect 並非精密儀器，這樣的小誤差不會對 Kinect 的應用產生影響。

此範例最大缺點在於沒有對超過邊界值的角度進行保護，因此當我們按下方向鍵嘗試讓 Kinect 轉動到它無法到達的角度時，程式會產生例外。

另外一個缺點，就是如果連續按下方向鍵的時間間隔太短，也會引發例外，這是 Kinect 的硬體保護機制所引發的執行時期例外。

如何增加Kinect程式的穩定度

- 設定ElevationAngle會造成Exception
  - 為了保護馬達過熱的措施
  - 一秒內最多改變一次Tilt角度
  - 連續改變15次角度以後，強制休息20秒
  - 沒事盡量不要更改Tilt角度
- ElevationAngle有邊界值
- Kinect Sensor的數量假設

要讓 Kinect 應用程式有一定的穩固性，大體上需要：

1. Kinect 感應器數量的假設，如果沒有 Kinect 裝置應該停止執行
2. 確保程式不會要求超過 Kinect 能力範圍以外的角度
3. 配合 Kinect 步進馬達的保護措施

Kinect感應器數量的處理

LINQ

```
static void Main(string[] args)
{
    KinectSensor sensor = (from sensorToCheck in KinectSensor.KinectSensors
                           where sensorToCheck.Status == KinectStatus.Connected
                           select sensorToCheck).FirstOrDefault();

    if (sensor == null)
    {
        Console.WriteLine("找不到任何可用的Kinect裝置，結束執行");
        return;
    }

    sensor.Start();
```

要解決穩定度的問題，首先可以利用 LINQ 選出所有已經進入連結狀態的 Kinect 感應器。

如果沒有任何進入連結狀態的 Kinect 感應器則結束執行。

LINQ語法可以用迴圈改寫

```
foreach (var potentialSensor in KinectSensor.KinectSensors)
{
   if (potentialSensor.Status == KinectStatus.Connected)
   {
      sensor = potentialSensor;
      break;
   }
}
```

如果您不熟悉 LINQ 語法，那麼可以將前述的 LINQ 語法改為 foreach 迴圈，兩者意義相同。

邊界值處理

```
while ( (presskey =Console.ReadKey().Key ) != ConsoleKey.Spacebar)
{
   if (presskey == ConsoleKey.DownArrow)
   {
      if (sensor.ElevationAngle - 5 < sensor.MinElevationAngle)
         sensor.ElevationAngle = sensor.MinElevationAngle;
      else
         sensor.ElevationAngle = sensor.ElevationAngle - 5;
   }
   else if (presskey == ConsoleKey.UpArrow)
   {
      if (sensor.ElevationAngle + 5 > sensor.MaxElevationAngle)
         sensor.ElevationAngle = sensor.MaxElevationAngle;
      else
         sensor.ElevationAngle = sensor.ElevationAngle + 5;
   }
   Console.WriteLine("現在角度：" + sensor.ElevationAngle);
}
```

確保程式不會要求超過 Kinect 能力範圍以外的角度，可以在設定 ElevationAngel 前先偵測系統預設的邊界值，如此就不會出現超過邊界值而產生的例外。

配合Kinect步進馬達的保護措施

```
while ( (presskey =Console.ReadKey().Key ) != ConsoleKey.Spacebar)
{
   if (presskey == ConsoleKey.DownArrow)
   {
      if (sensor.ElevationAngle - 5 < sensor.MinElevationAngle)
         sensor.ElevationAngle = sensor.MinElevationAngle;
      else
         sensor.ElevationAngle = sensor.ElevationAngle - 5;
   }
   else if (presskey == ConsoleKey.UpArrow)
   {
      if (sensor.ElevationAngle + 5 > sensor.MaxElevationAngle)
         sensor.ElevationAngle = sensor.MaxElevationAngle;
      else
         sensor.ElevationAngle = sensor.ElevationAngle + 5;
   }
   Console.WriteLine("現在角度 : " + sensor.ElevationAngle);
   Thread.Sleep(1000);
}
```

規格上限制每秒最多改變一次角度，我們可以在改變角度後，利用 Thread.Sleep(1000) 強制讓程式沉睡一秒鐘，如此就可以解決使用者兩次按鈕間距時間太短的問題。

要使用 Thread 物件，請記得加入底下命名空間：

```
using System.Threading;
```

這個程式還有一點小問題，就是鍵盤的輸入會在程式沉睡時繼續存放在鍵盤的緩衝區中。如果您可以預先把您要做的動作輸入，就可以等待應用程式一秒一秒慢慢地回應這些輸入的動作。或許這是您想要的結果，但一般的情況下，我們會希望程式在沉睡時的鍵盤輸入都視為無效，我們把這個議題留給讀者練習。

4-7 程式範例 - 控制 Kinect 垂直角度（使用 C++）

- 使用C++
- 使用Microsoft Kinect SDK
- 調整Visual Studio環境參數
- 連接Kinect Sensor
- 控制Kinect Sensor的俯仰角

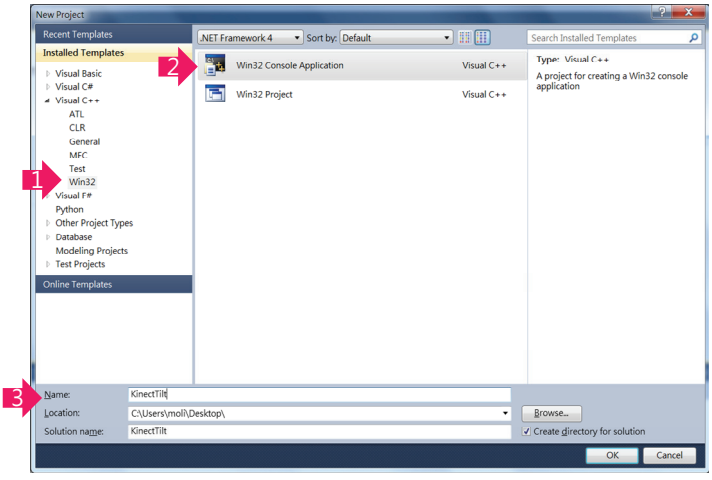
▲ 練習重點（使用 C++）

此程式範例與之前的幾乎相同，只是改用 C++ 程式語言實作。

相較於.NET 平台的開發，使用 C++進行原生（Native）開發相對複雜一些，不過對於熟悉 C/C++的朋友來說不至於產生困擾。

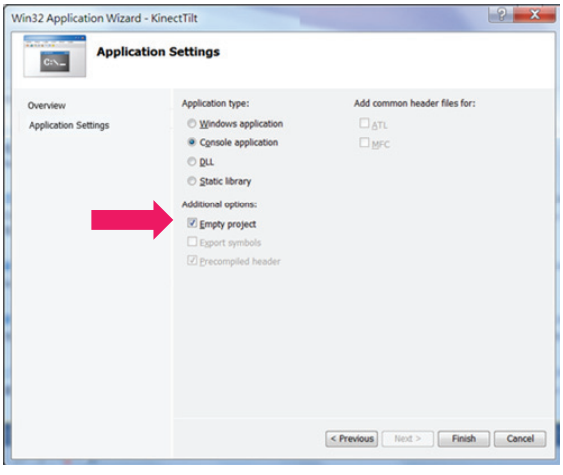
使用 C++ 與 C# 最大的不同在於所叫用的函式庫完全不同，這是因為 C++ 直接呼叫最基礎的 Kinect 函式庫（NUI Library），而 C# 所叫用的 Microsoft.Kinect.dll 組件已經針對底層的 NUI Library 做了更高階的包裝。

建立C++ 專案



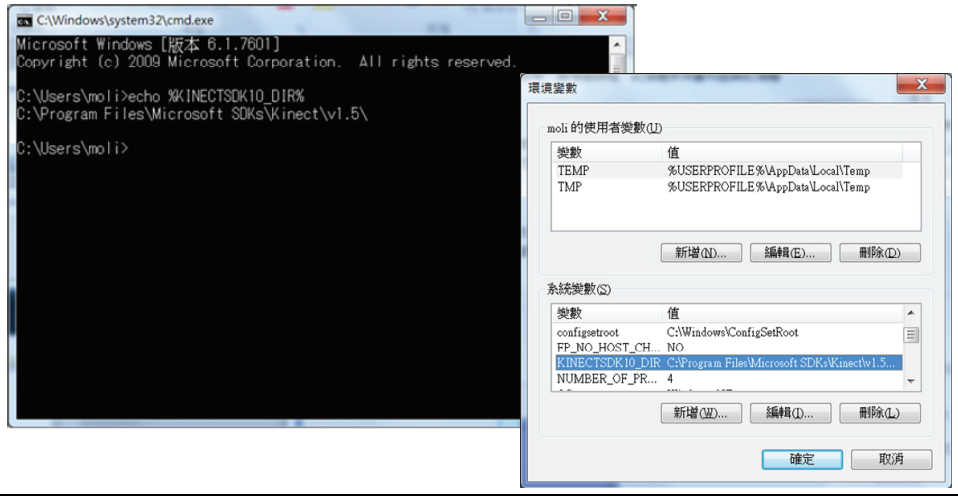
輸入專案名稱後按下 OK 即可。

設定精靈



Visual Studio 的設定精靈會詢問一些相關的設定，請勾選 Empty project，避免設定精靈產生太複雜的樣板與程式碼。

查詢Kinect環境變數



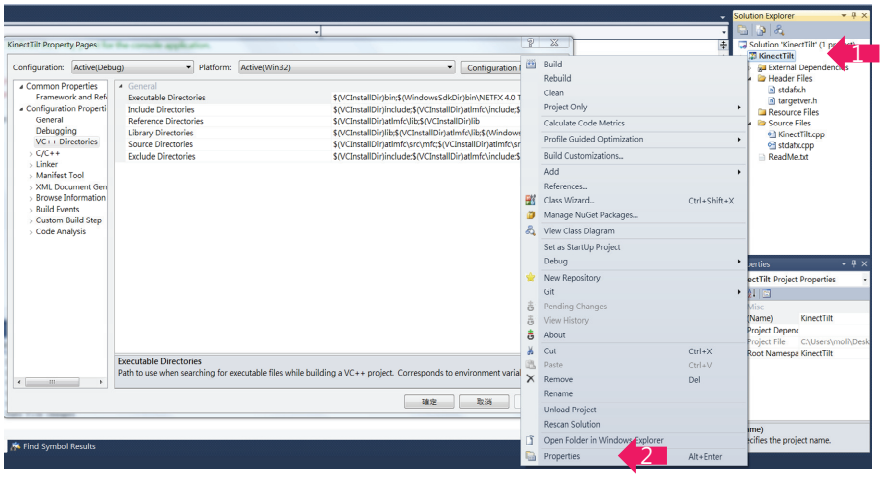
前面章節提過 Kinect SDK 會設定環境變數 KINECTSDK10_DIR，使其指向 Kinect SDK 相關的位置。

您可以像左圖開啟命令列模式，輸入

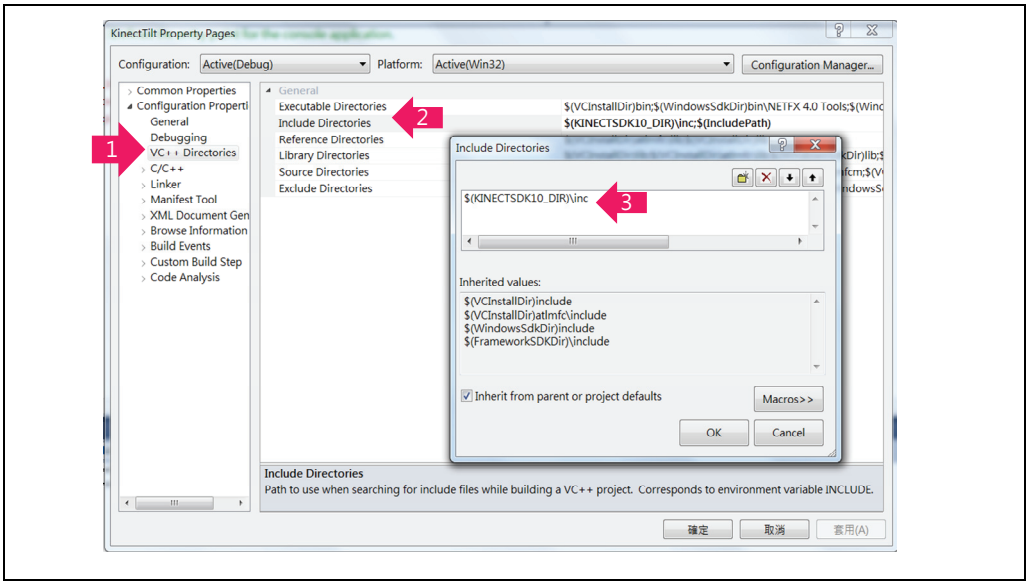
```
echo %KINECTSDK10_DIR%
```

確認是否設定正確。也可以像右圖一般查詢設定值。

調整Visual Studio環境參數，加入Kinect標頭檔位置



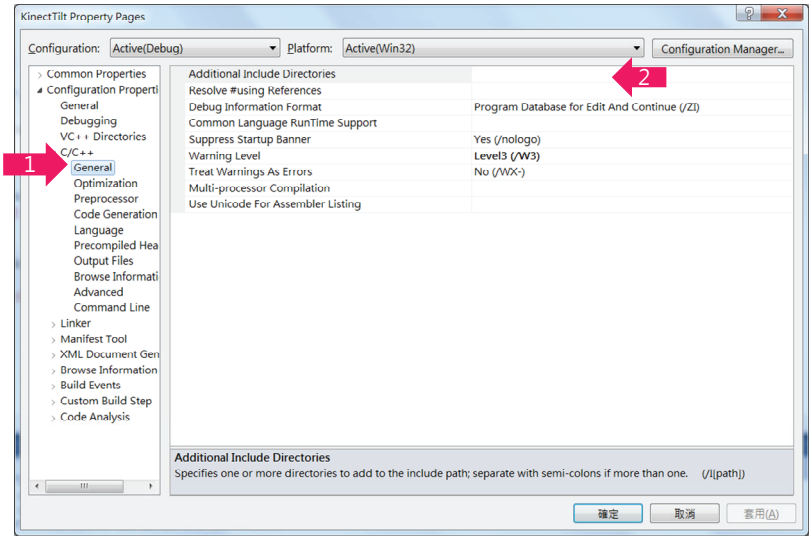
在專案上按下滑鼠右鍵，選擇 Properties。



點選 VC++ Directories，接著變更 Include Directories 的內容，請增加

```
$(KINECTSDK10_DIR)\inc\
```

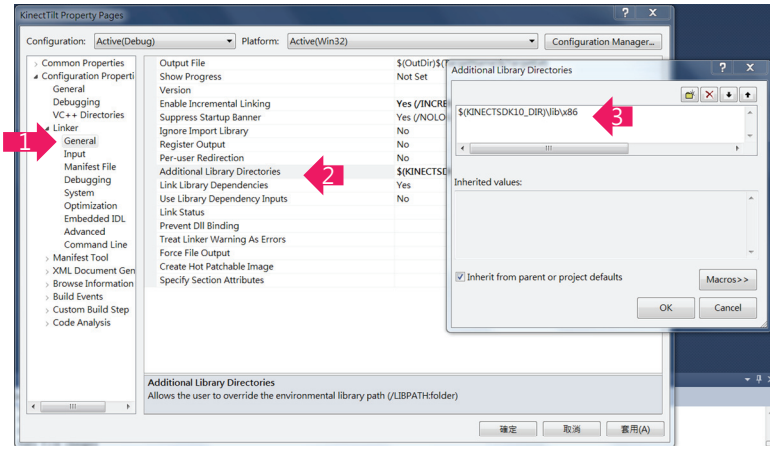
除了可以直接在 VC++ Directory 裡面增加，也可以進入 C/C++ ➔ General 裡面增加（如下圖所示）。



選擇 C/C++ ➔ General，於 Additional Include Directories 中增加

```
$(KINECTSDK10_DIR)\inc\
```

調整Visual Studio環境參數，加入 .lib檔位置與 .lib檔

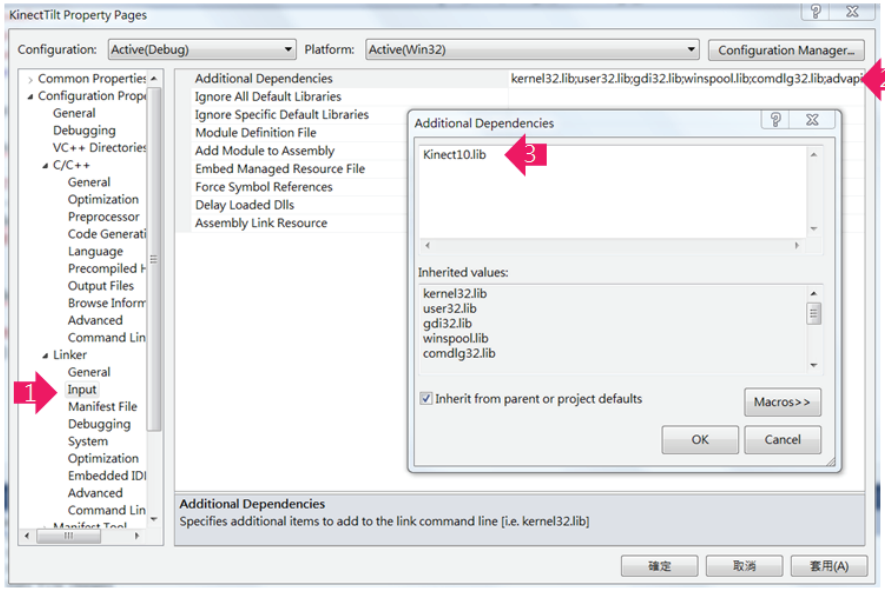


點選 Linker ➔ General，變更 Additional Library Directories 的內容，可以改為

```
$(KINECTSDK10_DIR)\lib\x86 或 $(KINECTSDK10_DIR)\lib\amd64
```

如果您不清楚您的機器上該使用 x86 或 amd64，建議您先使用 x86，遇到問題再改用 amd64。

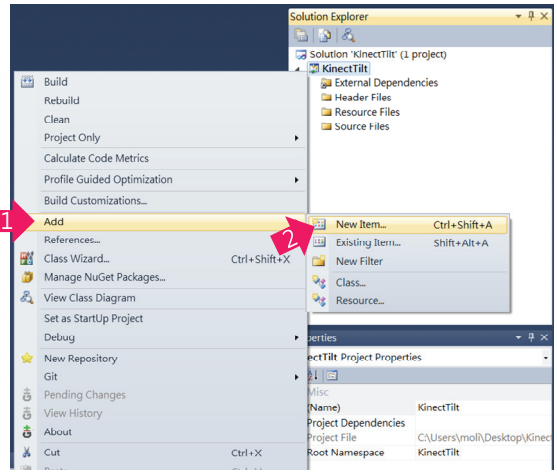
調整Visual Studio環境參數，加入Kinect10.lib檔



點選 Linker ➔ Input，於 Additional Dependencies 中增加 Kinect10.lib。

如果忘了上述兩個步驟，將出現 Unsolved external symbol 的編譯錯誤訊息。

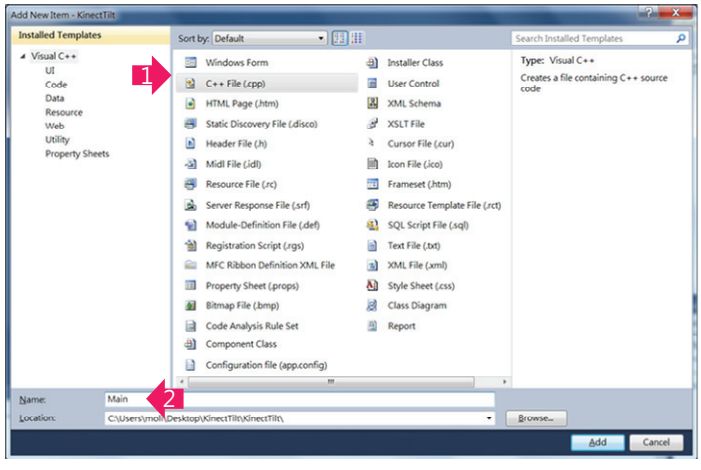
新增專案項目



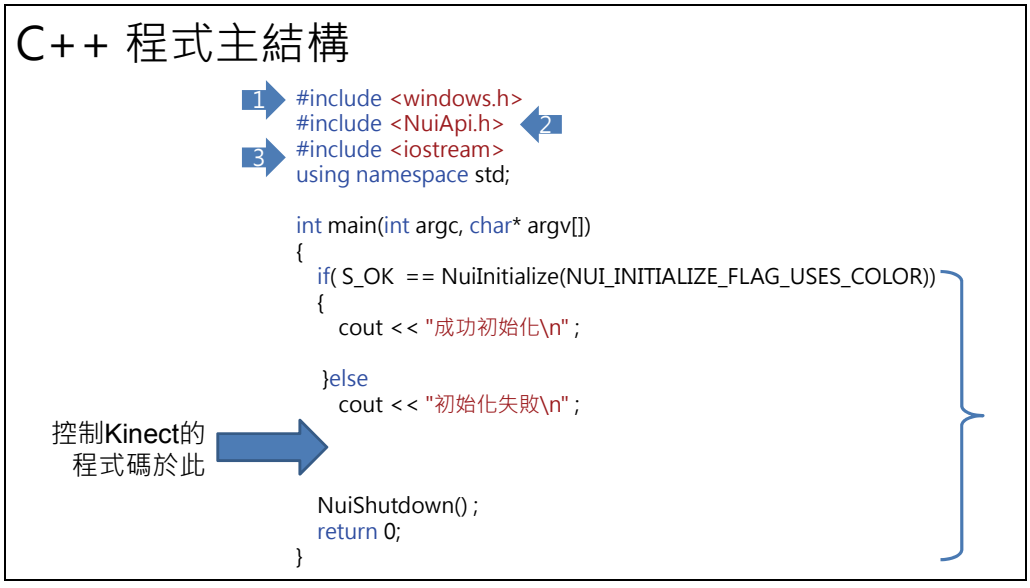
接著我們要在專案中加入 C++程式碼。

請在專案節點上按下滑鼠右鍵，選擇 Add → New Item 添加新的項目

新增C++ 檔案作為主程式



選擇 C++ File（.cpp），填好檔名後按下 OK。



C++ 直接透過 NUI Library 控制 Kinect，因此首先需要

```
#include <NuiApi.h>
```

引入 NUI Library。由於 NUI Library 內部用了很多 Windows 的相關設定，因此必須額外加入

```
#include <windows.h>
```

方能讓 NUI Library 正常運作。

此外，為了做簡單的輸入與輸出，請加入

```
#include <iostream>
using namespace std;
```

使用 NUI Library 時，必須先使用 NuiInitialize() 啟動 Kinect 方能進行後續的其他動作，結束時請使用 NuiShutdown() 關閉 Kinect 感應器。

設定Kinect Sensor俯仰角

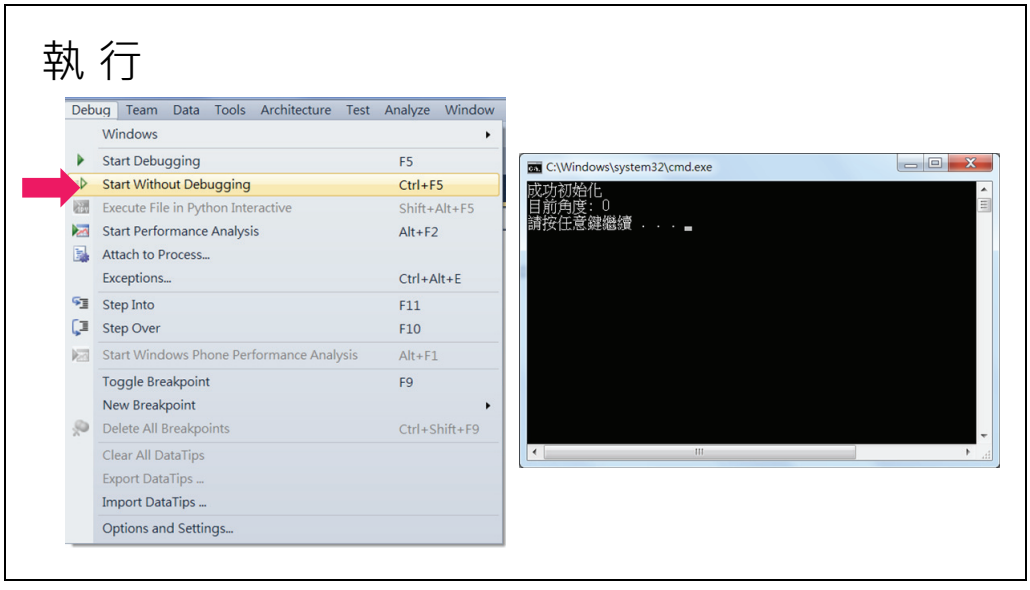
```
int main(int argc, char* argv[])
{
  if( S_OK  == NuiInitialize(NUI_INITIALIZE_FLAG_USES_COLOR))
  {
    cout << "成功初始化\n" ;
    long degree = 0 ;
1   NuiCameraElevationGetAngle(&degree) ;
    cout << "目前角度: " << degree  << endl;

    degree = 20 ;
    NuiCameraElevationSetAngle(degree) ;  2

  }else
    cout << "初始化失敗\n" ;

  NuiShutdown() ;
  return 0;
}
```

一旦成功地啟動 Kinect 感應器，我們可以隨時利用 NuiCameraElevationGetAngle() 取得目前 Kinect 的角度，也可以使用 NuiCameraElevationSetAngle() 設定 Kinect 的角度。



為了避免主控台程式一執行完畢就立刻關閉，請使用 Debug ➔ Start Without Debugging 執行程式。如果設定都沒問題，Visual Studio 會開始編譯，編譯成功後直接執行程式。

由於本書除了此範例使用 C++，其他接採用 C#。我們將不再針對如何用 C++ 控制 Kinect 進行討論。

本範例一樣有穩固性的問題：

1. Kinect 感應器數量的假設

   本範例只能使用一台 Kinect，如果要控制多台，必須使用 NuiGetSensorCount() 取得感應器數量，並搭配 NuiCreateSensorByIndex() 回傳 INuiSensor 介面，

   就能夠同時控制多台 Kinect。

2. 確保程式不會要求超過 Kinect 能力範圍以外的角度
3. 配合 Kinect 步進馬達的保護措施

我們把這些問題的修正作為給讀者的練習題。