

\$1600

# Unity3D + Photon 線上遊戲開發課程講義

(使用 Unity3D 4.X + Photon Server 3.X)



作者：紀曲峰

# 目錄

一 前言.....	1
1. 什麼是 Unity3D.....	1
2. Unity3D 的版本.....	1
3. 什麼是 Photon .....	2
4. 關於本教學的 Photon 版本 .....	2
二 開發工具.....	3
1. 下載及安裝 Unity3D.....	3
2. 下載 Photon .....	5
3. 解壓及安裝 Photon 授權 .....	6
4. 啟動 Server .....	8
三 Photon 入門教學 .....	10
1. 建立 Photon 專案 .....	10
(1). 建立 Visual Studio 方案 .....	11
(2). 建立 Server 專案 .....	11
(3). 加入參考 .....	12
(4). 建立 Server 端框架程式 .....	14
(5). 建立 Client 端測試程式 .....	22
(6). Server 端和 Client 端的訊息傳輸.....	28
2. 測試.....	37
四 Unity3D 與 Photon 的連接 .....	51
1. 建立專案.....	51
2. 建立 Unity for Photon 基本架構 .....	52
3. 連線到 Server .....	55
4. Unity 與 Server 的資料傳輸 .....	60
五 將命令代碼變成看得懂的東西吧.....	66
1. 建立列舉命令類別.....	66
2. 將 Server 端的命令碼換成列舉的類別 .....	72
3. 將 Client 端的命令換成列舉類別 .....	74
4. 將 Unity 的命令代碼換成列舉類別 .....	78

六 柚子星球多人聊天室原始碼說明.....	81
1. Server 架構圖 .....	82
2. 廣播的原理.....	82
3. Client 的委派與事件.....	85
七 結語.....	90

# 一 前言

## 1. 什麼是 Unity3D

Unity3D 是一個套裝的 3D 遊戲引擎，合理的價格加上不錯的效能，還有簡單易用的特色，吸引了非常多的公司與獨立製作者使用這套引擎來開發遊戲，尤其是移動平台裝置的遊戲開發，絕大部份都是使用 Unity 做為遊戲引擎，目前，Unity 已是全世界最多人使用的 3D 遊戲引擎，並且使用的領域也跨足到其他的商業應用，已不止是用在遊戲上。

Unity3D 目標平台支援 Windows、Mac、Linux、iOS、Android、Flash 等平台，但開發平台僅支援 Windows 和 Mac，兩個平台的專案可互通，也就是說一個 Team 可以同時使用 Windows 和 Mac 協同開發，在國內的情形很多是使用 3D Max 進行美術工作，由於 3D Max 不支援 Mac 因此必須使用 Windows 的機器進行開發，但若要開發 iOS 的 App 就必須使用 Mac 電腦，這時雙平台專案互通就變得很重要了。

## 2. Unity3D 的版本

本教學使用的 Unity3D 為 4.x 的版本，讀者可自行到 Unity3D 官網下載，Unity3D 有分為免費版本與收費版本，書中所有的教學及範例均可使用免費版本的 Unity Engine，因此在學習上不會出現不能使用的問題，這也是使用 Unity 的一個很大的好處，學習的成本很低。

Unity3D 的授權方式採用一次授權永久使用，也就是說若有需要使用專業功能並進行購買後，無論開發幾套遊戲都不需再行付費，而且可以開發 Online Game，這一點 Unity 確實是比起其他的引擎，如 UDK 或 CDK 好得多了，上線後也不需再給任何的授權費用。

### 3. 什麼是 Photon

Photon 是一個泛用型的 Socket Server 套裝軟體，可用於多人線上遊戲、聊天室、大廳遊戲，Client 端支援 Windows AP、Unity3D、iOS、Android、Flash 等平台。

Photon 內建一套大廳遊戲伺服器及一套 MMO 遊戲伺服器，都含有原始碼，使用者可以拿來修改成自己所需的 Server 或直接繼承後加入自己的遊戲邏輯，本書的內容並不教如何使用 Photon 提供的現成 Server，而是從零開始撰寫一套屬於自己的 Game Server，看完本手冊後再去修改 Photon 內建的原始碼便能更加的得心應手。

### 4. 關於本教學的 Photon 版本

本教學是以 Photon 3.x 為主，Photon3.x 比起 2.x 多了非常多的功能，特性上也比較完善，並捨棄了舊式的 Hashtable 全面改用 Dictionary 資料型態，對於資料的處理可以統一一種型態，對開發者來說方便多了。

更多的 Photon 特色及問題可直接到 Photon 官方網站查詢

<http://www.exitgames.com/>

## 二 開發工具

本書會用到不少的開發工具，請讀者自行備妥或依書中指示下載。

Visual Studio 2010 以上的版本

Unity3D 4.0 以上的版本(免費版即可)

Photon Server 3.0 以後的版本(開發或小型網路用免費版即可)

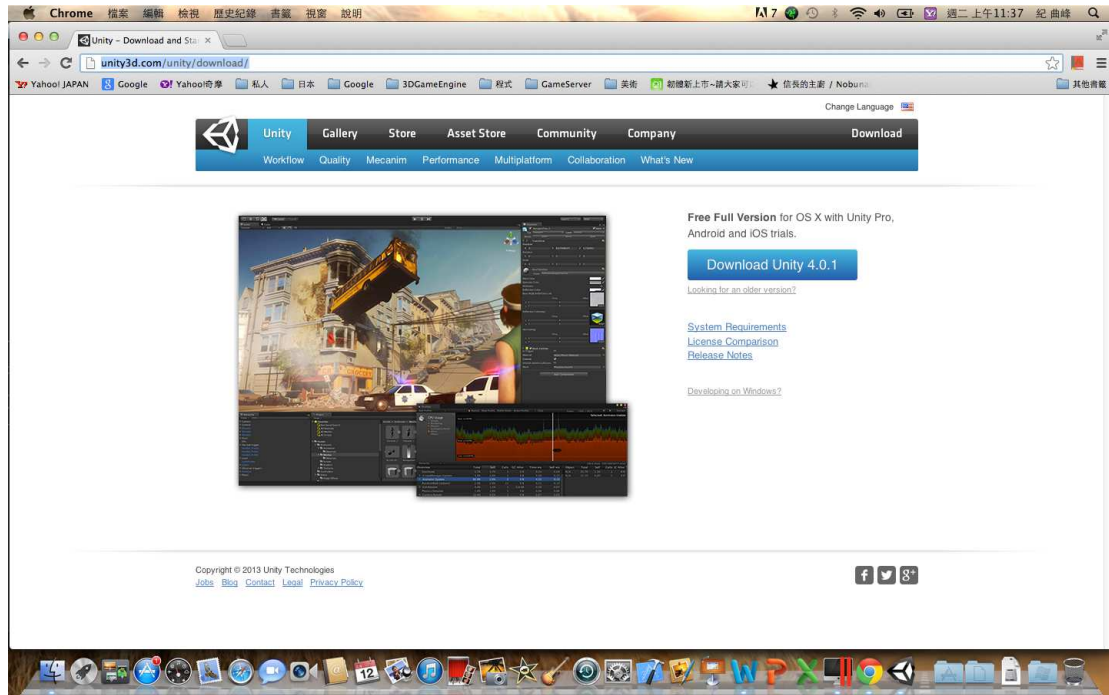
除此之外，電腦必須安裝好.net framework，不過這個在安裝 Visual Studio 或 Unity3D 都一定會安裝此元件，因此不需要另外下載安裝。

### 1. 下載及安裝 Unity3D

通常遊戲發行前都會購買專業版本以進行特效及資源的最佳化，但在開發階段可以直接到 Unity 官網下載免費版本安裝，若需要購買可洽尋國內代理商，好處是可以開發票，對於公司行號來說做帳方便，且台灣的代理商服務算是蠻不錯的，直接向國外官網刷卡購買也可以，不過只有國外的電子發票。

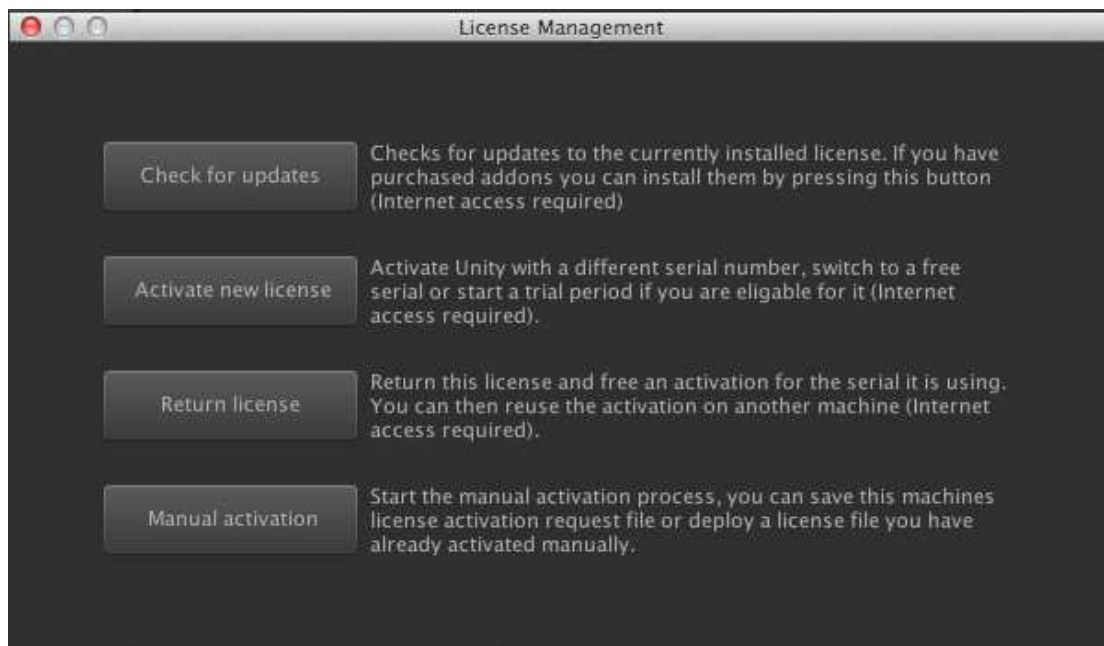
下載 Unity3D 只要到下面的網址

<http://unity3d.com/unity/download/>



點選「Download Unity X.X.X」進行下載，完成後直接安裝，照著指示下一步下一步，即可安裝完成，筆者用的是 Mac 電腦，Windows 電腦的下載及安裝方法也是一模一樣。

安裝完成後，可以直接使用專業版 30 天試用，等到時間到了再重新啟用授權為免費版本，若是 Windows 版本執行「Help > Manage License...」，若是 Mac 版請執行「Unity > Manage License...」，打開對話盒後執行「Activate new license」即可重新啟用為免費版，將來若購買了專業版也是用同樣方法重新啟用為專業版，不需要重新下載安裝。

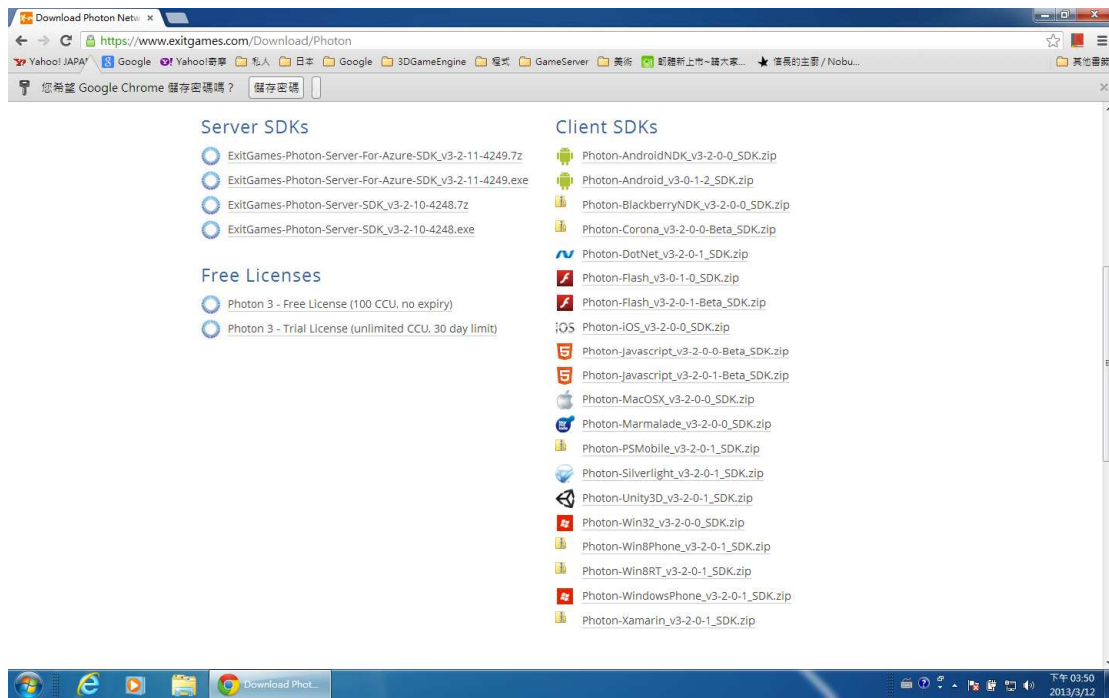


## 2. 下載 Photon

先到官網「<http://www.exitgames.com/>」下載 Photon，若尚未註冊請先註冊會員，請直接下載 3.0 最新版本，Client SDK 可選擇性下載，直接下載 Server SDK 會內含 .net 及 Unity 的 Client DK，不過另外下載的 Client SDK 會比較新一點。

進入官網後執行選單的「Download」，這時會要求登入會員，登入後可看到以下畫面，請在「ExitGames-Photon-Server-SDK\_v3-xxxxxxx.7z」這一項上面直接按滑鼠右鍵選另存將 SDK 存下來，下載時數字部份可能會有所不同，若沒有解壓軟體也可以下載副檔名「.exe」的自解壓縮檔下來執行解壓安裝。



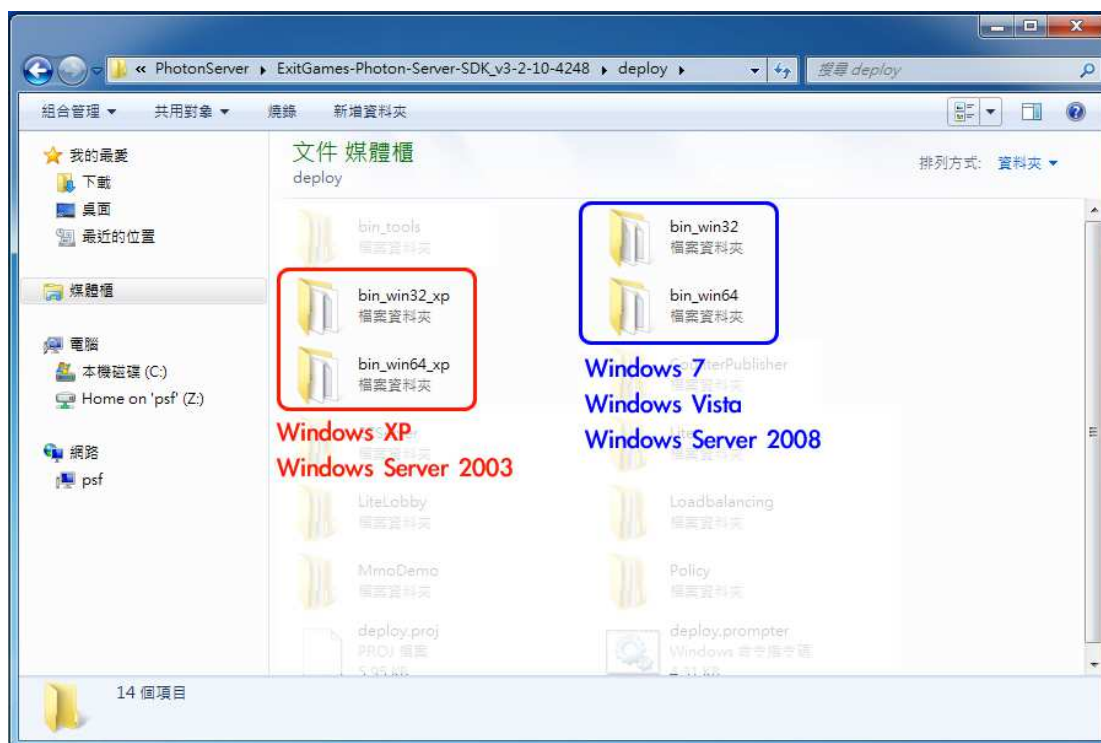


Licenses 可選無限制人數 30 天或 100 人免費，因為是開發用為了省麻煩，這裡請直接下載不限時間 100 人免費的授權「Free License(100 CCU, no expiry)」，請在連結上按下「另存新檔」存到自己的硬碟中。

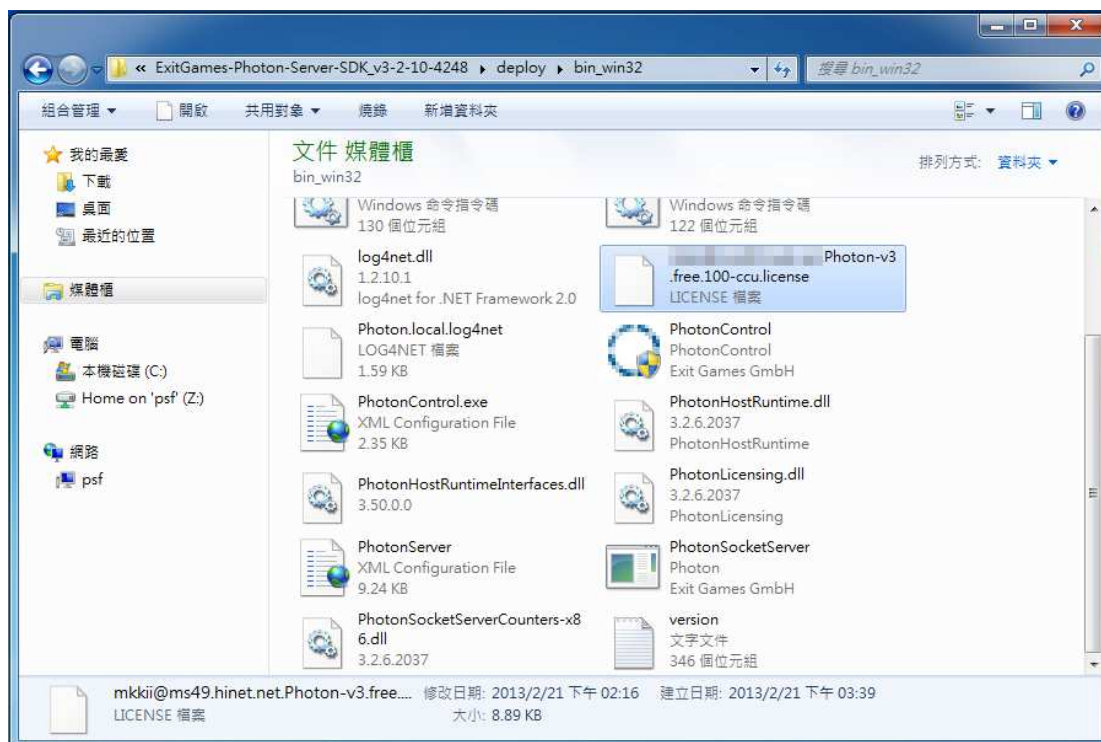
### 3. 解壓及安裝 Photon 授權

將 Server SDK 解壓到自己的硬碟，不需安裝，解開來即可使用，將來裝到主機上後可將 Photon 設為 Windows 服務。

依您的平台選擇正確的執行檔，先到解壓的目錄底下「deploy」目錄下，若您是 xp 或 Windows Server 2003 請進入 bin\_Win32\_xp(32 位元)或是 bin\_Win64\_xp(64 位元)，若您是 Win7 或 Windows Server 2008 以後版本請執行進入 bin\_Win32(32 位元)或是 bin\_Win64 (64 位元)。



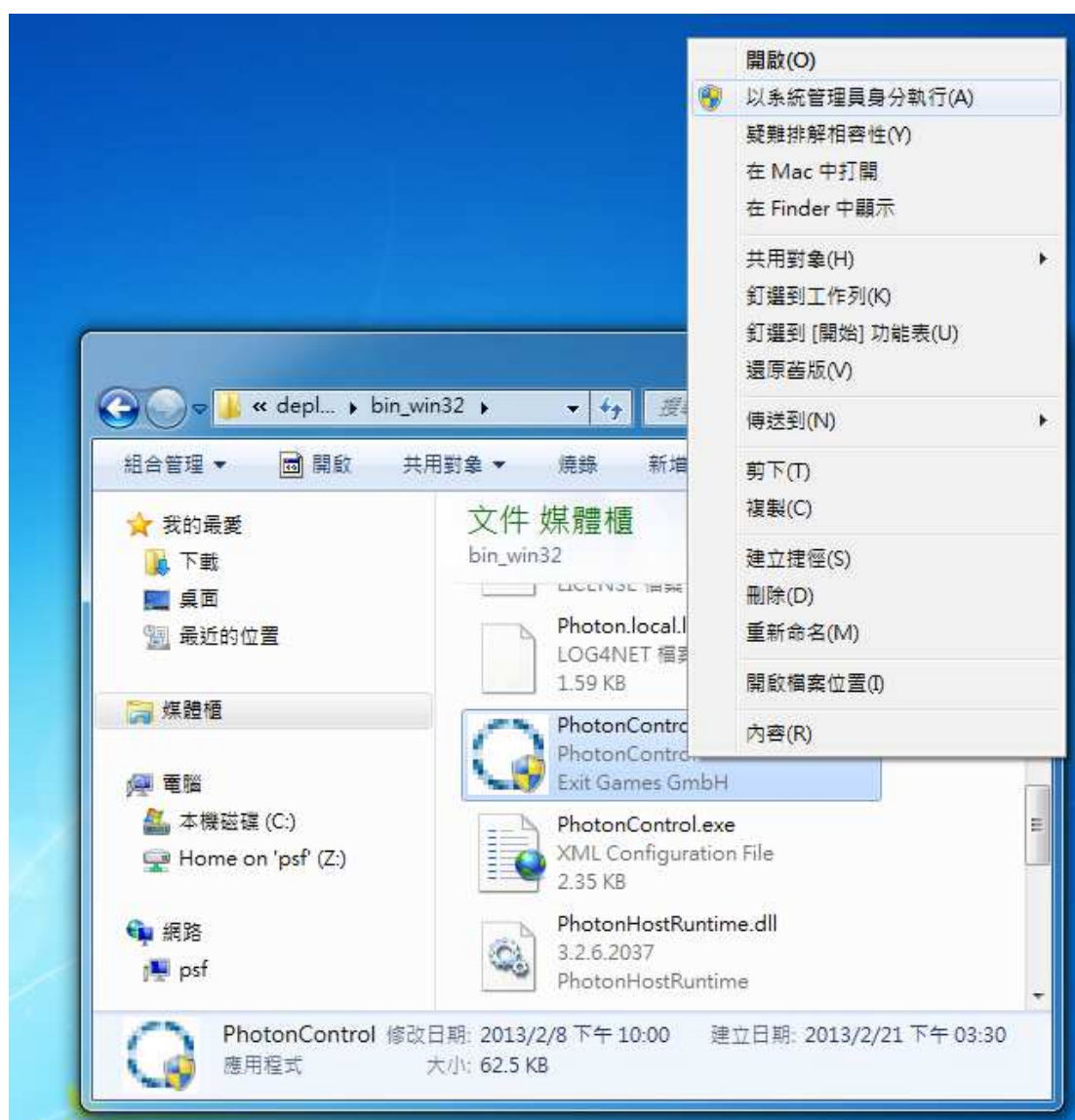
進入資料夾後，請將之前下載的 License 放到這個資料夾內，若裡面已有名為「30DayTemporary.license」的檔案請先刪除再放入自己的 License 檔。



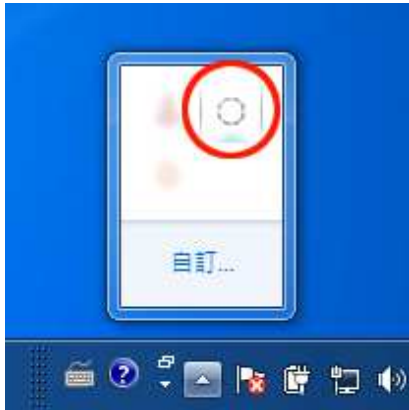
之後只要啟動 Server，Photon Server 便會自己抓到資料夾底下的授權檔。

## 4. 啟動 Server

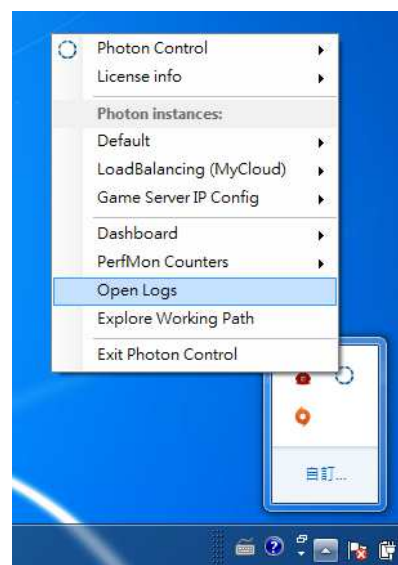
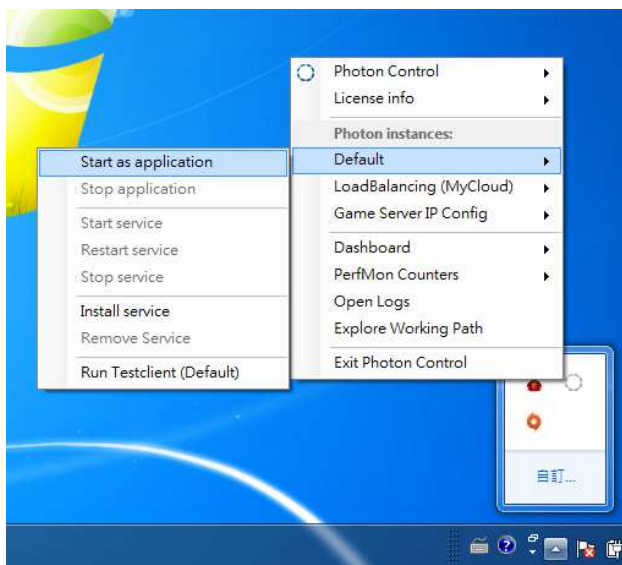
執行「PhotonControl.exe」，在工作列上會出現 Photon 的 icon，若執行發生問題就以系統管理員身分執行即可，因為會啟動 Windows Service 故有時必須以系統管理員身分執行。



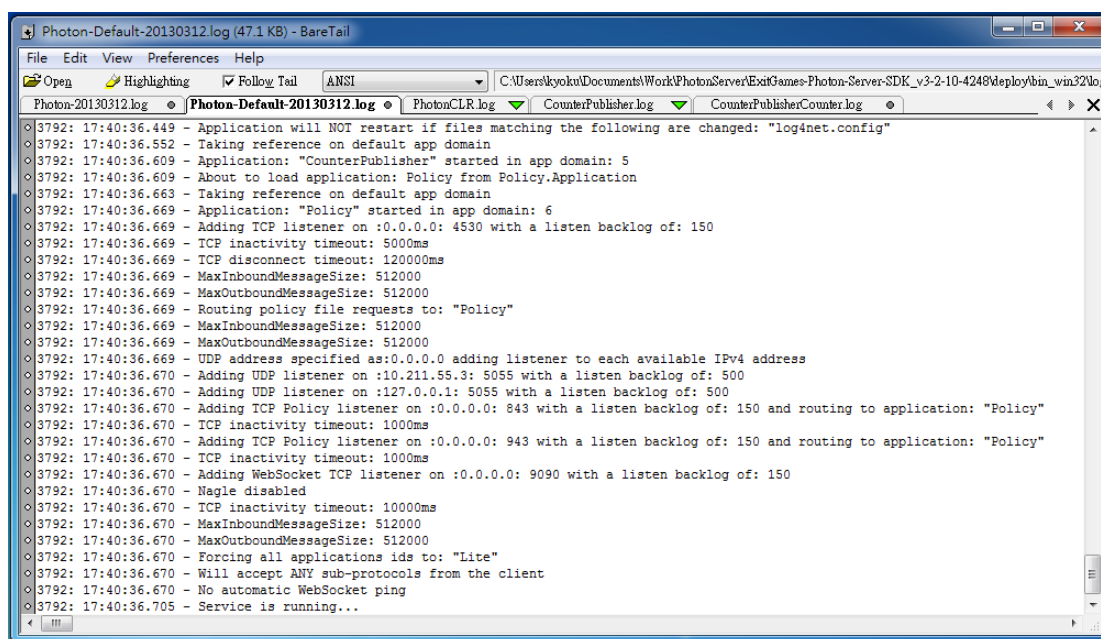
執行之後便會在工作列上出現 Photon Server 的 icon 了。



接下來在 Photon icon 上按滑鼠右鍵執行「Photon > Default > Start as application」，之後再執行「Open Logs」



等到 Log 畫面出現「Service is running...」就是啟動完成了。



```
Photon-20130312.log | Photon-Default-20130312.log | PhotonCLR.log | CounterPublisher.log | CounterPublisherCounter.log
3792: 17:40:36.449 - Application will NOT restart if files matching the following are changed: "log4net.config"
3792: 17:40:36.552 - Taking reference on default app domain
3792: 17:40:36.609 - Application: "CounterPublisher" started in app domain: 5
3792: 17:40:36.609 - About to load application: Policy from Policy.Application
3792: 17:40:36.663 - Taking reference on default app domain
3792: 17:40:36.669 - Application: "Policy" started in app domain: 6
3792: 17:40:36.669 - Adding TCP listener on :0.0.0.0: 4530 with a listen backlog of: 150
3792: 17:40:36.669 - TCP inactivity timeout: 5000ms
3792: 17:40:36.669 - TCP disconnect timeout: 120000ms
3792: 17:40:36.669 - MaxInboundMessageSize: 512000
3792: 17:40:36.669 - MaxOutboundMessageSize: 512000
3792: 17:40:36.669 - Routing policy file requests to: "Policy"
3792: 17:40:36.669 - MaxInboundMessageSize: 512000
3792: 17:40:36.669 - MaxOutboundMessageSize: 512000
3792: 17:40:36.669 - UDP address specified as:0.0.0.0 adding listener to each available IPv4 address
3792: 17:40:36.670 - Adding UDP listener on :10.211.55.3: 5055 with a listen backlog of: 500
3792: 17:40:36.670 - Adding UDP listener on :127.0.0.1: 5055 with a listen backlog of: 500
3792: 17:40:36.670 - Adding TCP Policy listener on :0.0.0.0: 843 with a listen backlog of: 150 and routing to application: "Policy"
3792: 17:40:36.670 - TCP inactivity timeout: 1000ms
3792: 17:40:36.670 - Adding TCP Policy listener on :0.0.0.0: 943 with a listen backlog of: 150 and routing to application: "Policy"
3792: 17:40:36.670 - TCP inactivity timeout: 1000ms
3792: 17:40:36.670 - Adding WebSocket TCP listener on :0.0.0.0: 9090 with a listen backlog of: 150
3792: 17:40:36.670 - Nagle disabled
3792: 17:40:36.670 - TCP inactivity timeout: 1000ms
3792: 17:40:36.670 - MaxInboundMessageSize: 512000
3792: 17:40:36.670 - MaxOutboundMessageSize: 512000
3792: 17:40:36.670 - Forcing all applications ids to: "Lite"
3792: 17:40:36.670 - Will accept ANY sub-protocols from the client
3792: 17:40:36.670 - No automatic WebSocket ping
3792: 17:40:36.705 - Service is running...
```

## 三 Photon 入門教學

### 1. 建立 Photon 專案

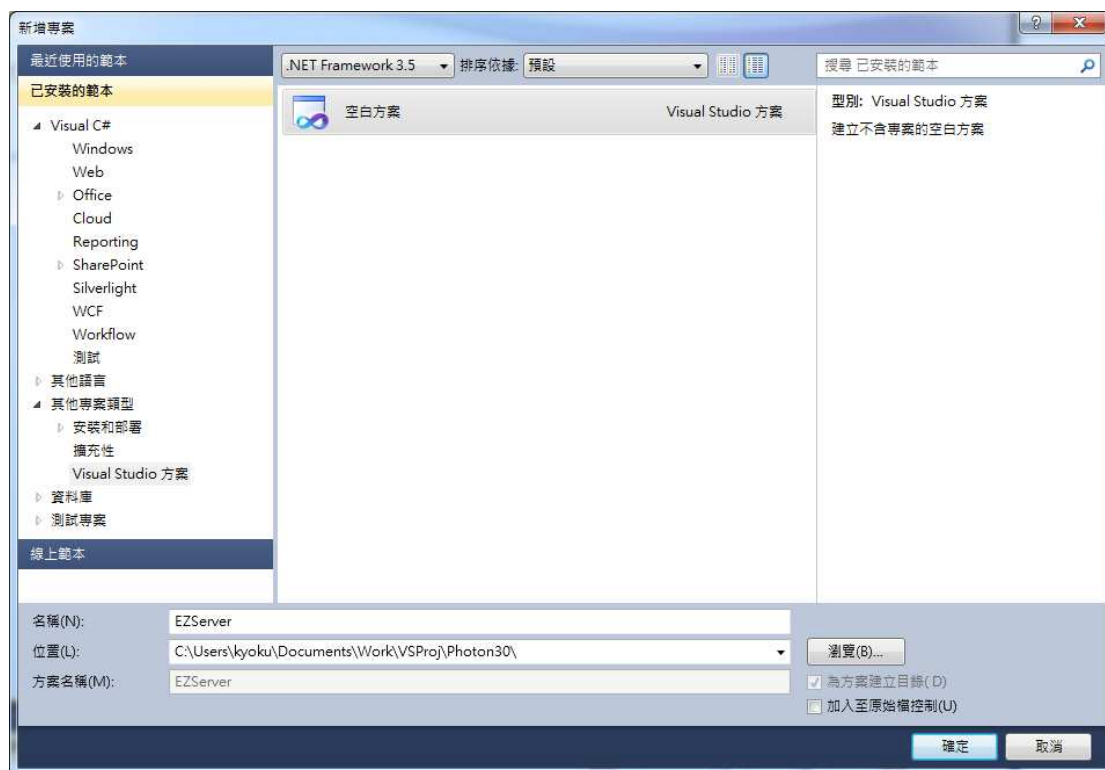
先前我們已經安裝好 Photon 了，現在我們開始建立 Photon 的基本專案，Photon 專案必需使用 .net Framework 3.5 及 Visual Studio 2008 以上的 Visual Studio 開發環境，因此對大多的 windows 開發者來說在開發環境上會很順手。

雖然 Photon 核心是 C++ 開發的，但 Photon 的 Server 端 SDK 是使用 C#，必需使用 C# 建立好遊戲所需 DLL 然後再掛載到 Photon 底下執行。

現在，請開啟 Visual Studio，本書的範例都是 VS 2010 開發的，您可以自行選用習慣的 VS 版本，但至少需要 VS 2008 以上。

## (1). 建立 Visual Studio 方案

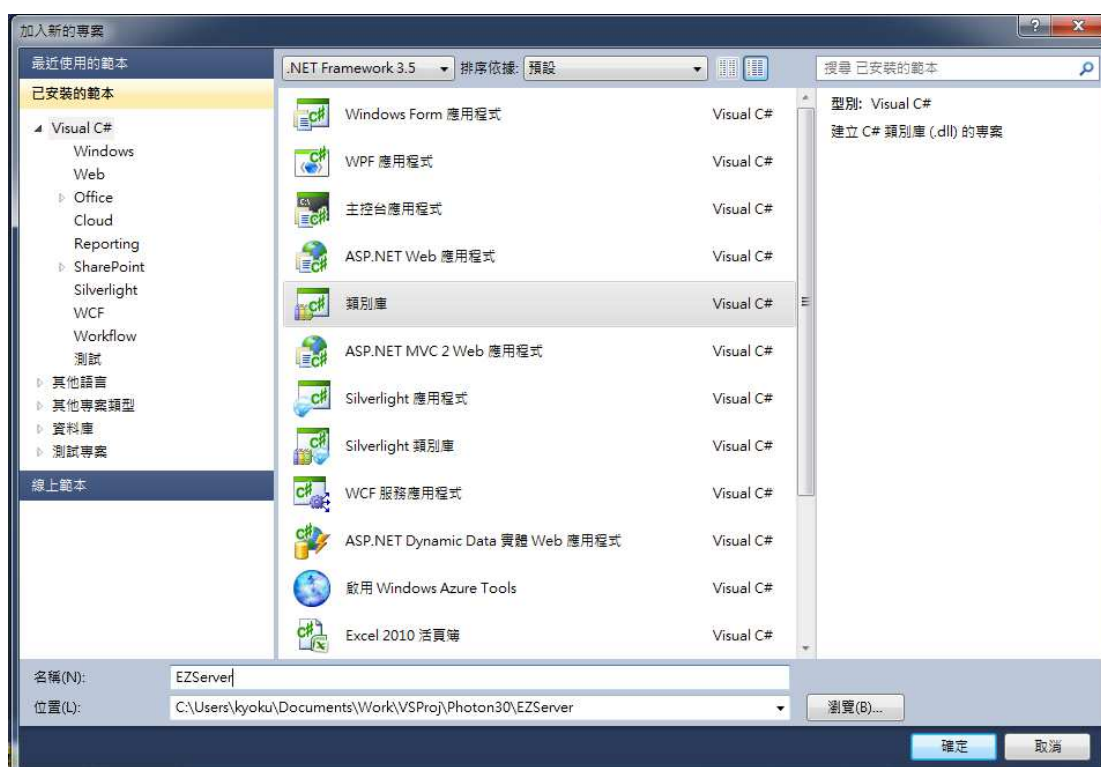
從 VS 建立一個空白方案，等一下會在此方案內建立 Server 專案及 Client 的專案，在 .Net Framework 版本選擇 3.5 或以上，.Net Framework 3.5 是 Photon 預設的 framework 版本，方案名稱這裡取為「EZServer」。



## (2). 建立 Server 專案

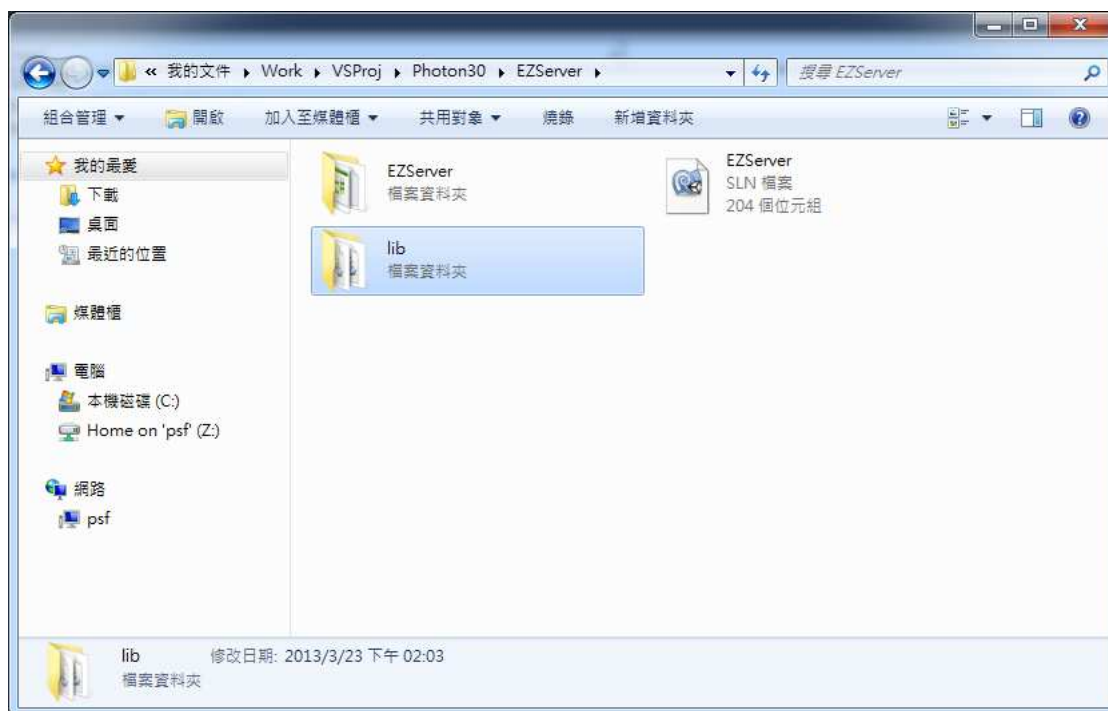
在建好的方案上按下滑鼠右鍵選加入一個新增專案，專案類型選「類別庫」，名稱取跟方案相同的「EZServer」，上面的 Framework 要選「.Net Framework 3.5」。



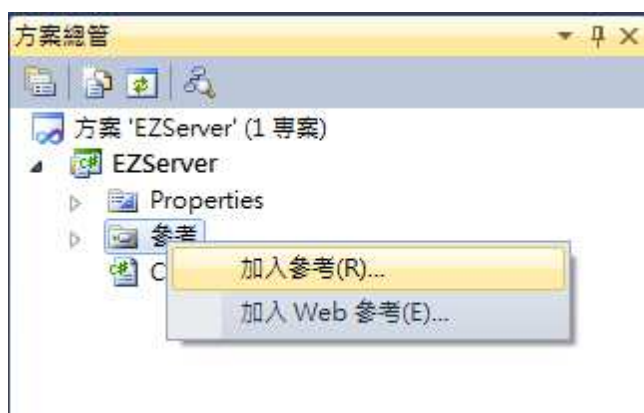


### (3). 加入參考

接下來將 Photon 安裝目錄底下的「lib」資料夾複製到 EZServer 底下以便我們加入參考。



回到 Visual Studio，在方案總管下的 EZServer 加入參考



分別加入以下的檔案

**ExitGames.Logging.Log4Net.dll**

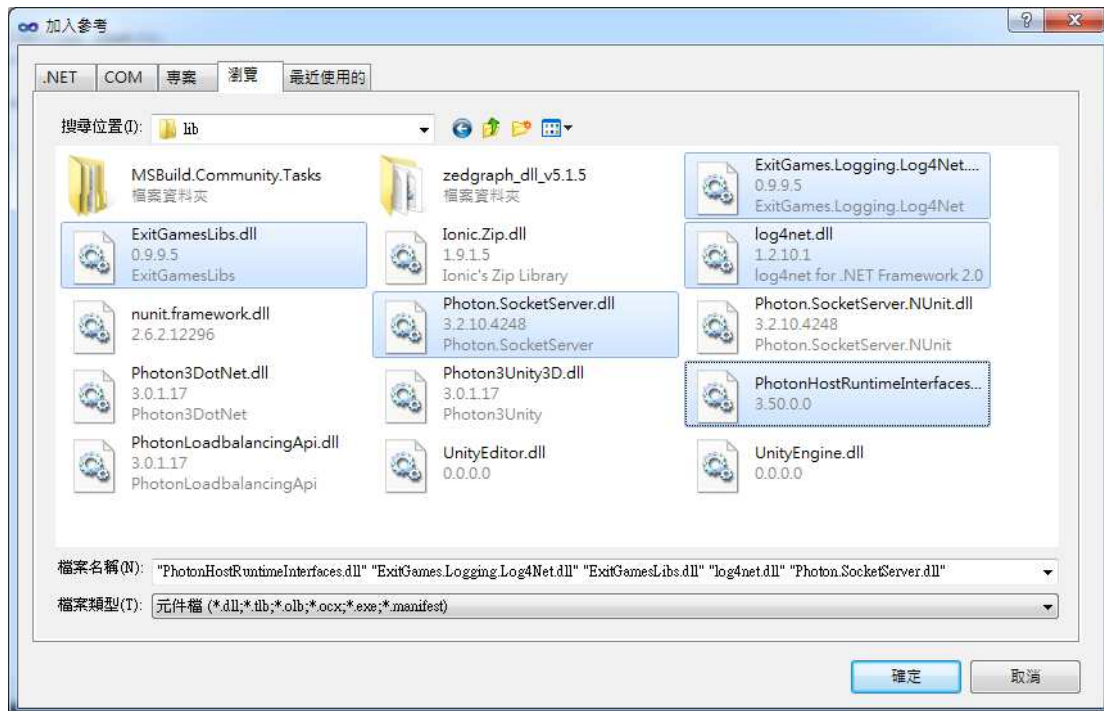
**ExitGamesLibs.dll**

**log4net.dll**

**Photon.SocketServer.dll**

**PhotonHostRuntimeInterfaces.dll**





雖然只有 PotonSocketServer 和 PhotonHostRuntimeInterfaces 是 Server 必需的，但其他的也提供了一些常用的好工具，因此我們直接匯入這些 dll。

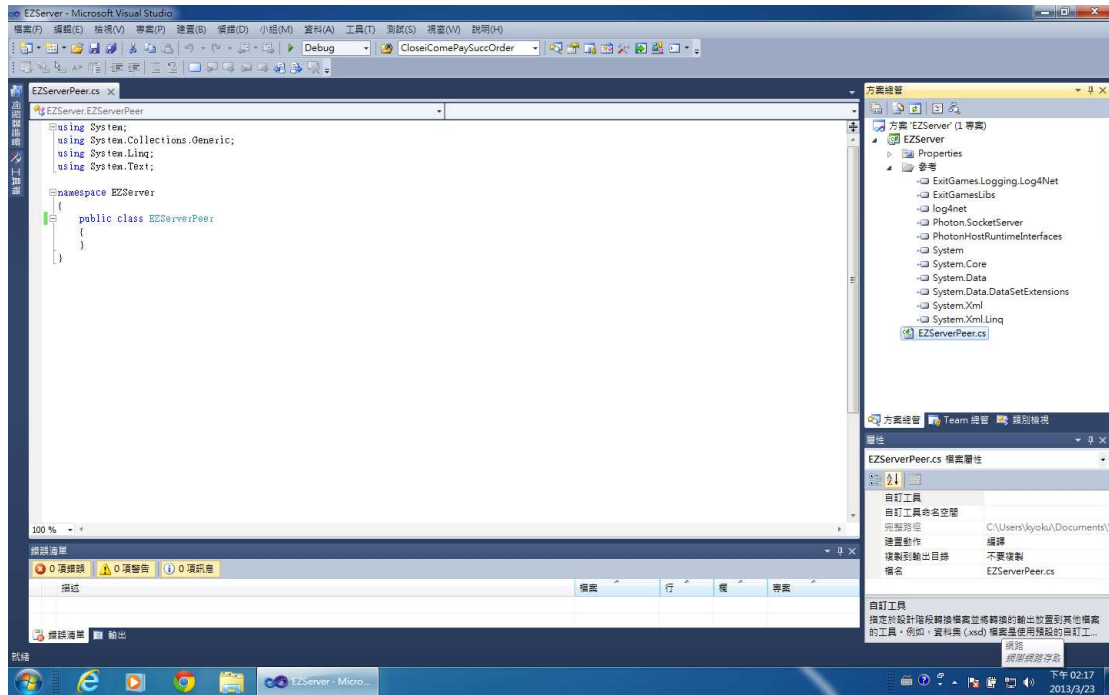
#### (4). 建立 Server 端框架程式

一個最精簡的 Photon 框架會含一個 Application 類別和一個 Peer 類別，作用分別如下：

**Application 類別：**整個 Server 的進入點

**Peer 類別：**處理和 Client 端的溝通

接著我們開始來建立這兩個類別，請將原本的 Class1.cs 更名為「EZServerPeer.cs」，VS 會自動幫我們將 Class 名稱也改成 EZServerPeer，當然也可以刪除 Class1.cs 檔另外用新增的方式建立一個 EZServerPeer.cs 類別。



建立好類別一開始會得到如下的程式碼。

```
using System;

using System.Collections.Generic;

using System.Linq;
using System.Text;

namespace EZServer
{

    public class EZServerPeer

    {

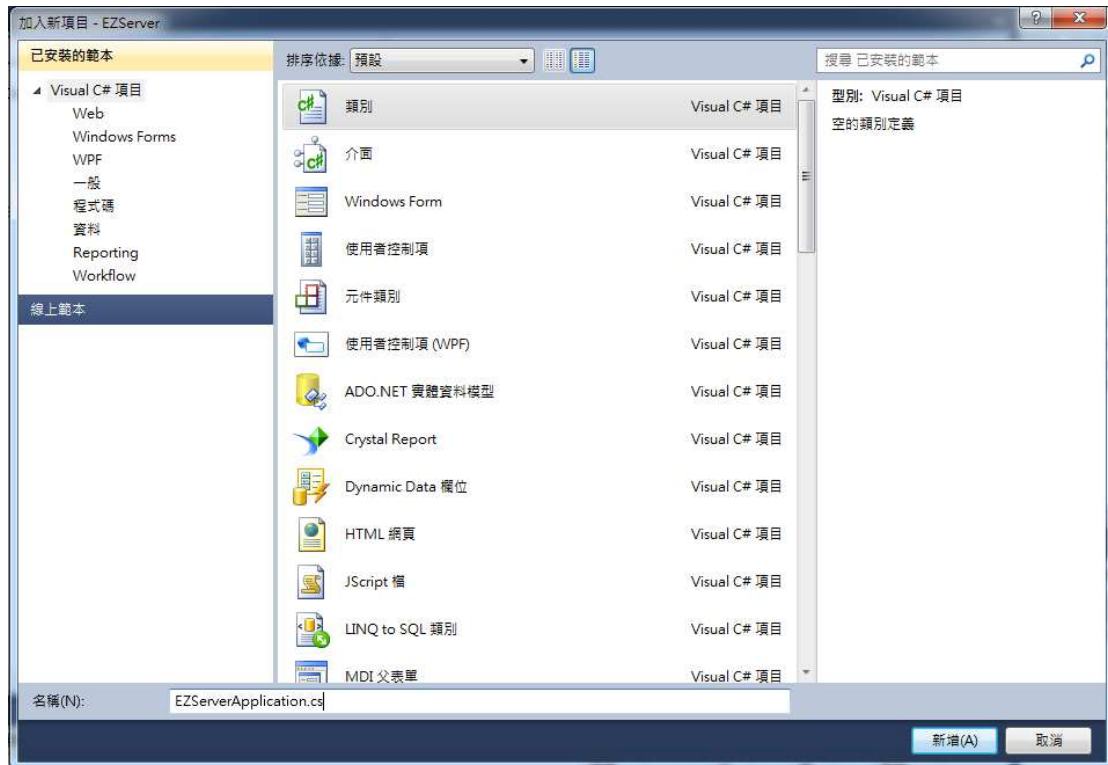
    }

}
```

這個類別是用來處理與 Client 溝通內容用的，接下來以新增項目加入一個

Application 類別來處理 Server 的進入點，以及後面用來管理線上所有連線和上線中玩家資料。

在專案裡面選新增類別，取名為「EZServerApplication.cs」。



新增 EZServerApplication.cs 好會得到如下的程式碼。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EZServer
{
    class EZServerApplication
    {
    }
}
```

然後來將 EZServerApplication 繼承自 ApplicationBase，並將 class 加入 public，接著上方加入「using Photon.SocketServer;」，如下。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Photon.SocketServer;

namespace EZServer
{
    public class EZServerApplication : ApplicationBase
    {
    }
}
```

接下來我們要加入 ApplicationBase 需要實作的抽象類別，Visual Studio 可以直接幫我們建立好所有需要實作的成員，我們不需要自己一個一個打，很方便，不過我們自行輸入結果也會是一樣的，若使用別的編輯器也可以自行輸入，不過既然我們使用 VS 就讓他動建立吧，將游標停在 ApplicationBase 字上面，一段時間會跳出選項，或是直接按下「Ctrl + .」跳出選項，如下圖，直接選擇「實作抽象類別」。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Photon.SocketServer;

namespace EZServer
{
    public class EZServerApplication : ApplicationBase
    {
    }
}
```

實作抽象類別 'ApplicationBase'(A)

除了 VS 以外，Unity3D 使用的 MonoDevelop 也有這樣的功能，我們在後面的章節建立 Unity 的 Client 端程式會看到如何使用，若您選用其他的編輯器無法自動產生需要實作的方法，您也可以將下面的程式碼自行複製到您的程式內。

完成後會得到如下的程式碼。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Photon.SocketServer;

namespace EZServer
{
    public class EZServerApplication : ApplicationBase
    {
        protected override PeerBase CreatePeer(InitRequest initRequest)
        {
            throw new NotImplementedException();
        }

        protected override void Setup()
        {
            throw new NotImplementedException();
        }
    }
}
```

```

    }

    protected override void TearDown()
    {
        throw new NotImplementedException();
    }
}
}

```

因為 EZServerApplication 在初始化時需要呼叫到 Peer，因此我們先處理一下 Peer 類別，請切換到 EZServerPeer，將 Peer 繼承自 PeerBase，PeerBase 是一個 interface，亦需要實作其相關成員，和 Application 一樣，我們也可以利用 Visual Studio 的「Ctrl + .」功能自動幫我們產生需要實作的成員，如下。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Photon.SocketServer;

namespace EZServer
{
    public class EZServerPeer : PeerBase
    {
        protected override void OnDisconnect(PhotonHostRuntimeInterfaces.DisconnectReason
reasonCode, string reasonDetail)
        {
            throw new NotImplementedException();
        }

        protected override void OnOperationRequest(OperationRequest operationRequest,
SendParameters sendParameters)

```

```

        {
            throw new NotImplementedException();
        }
    }
}

```

接著在 `EZServerPeer` 類加入建構式，建構式必需加入關鍵字「`: base(rpcProtocol, nativePeer)`」以確保父物件的建構式內容能先執行，上方加入「`using PhotonHostRuntimeInterfaces;`」，接著將原本實作方法內的 `Exception` 都刪除，我將每個實作方法的作用直接打上註解來說明每個方法的功用。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Photon.SocketServer;
using PhotonHostRuntimeInterfaces;

namespace EZServer
{
    public class EZServerPeer : PeerBase
    {
        #region 建構與解構式

        public EZServerPeer(IRpcProtocol rpcProtocol, IPhotonPeer nativePeer)
            : base(rpcProtocol, nativePeer)
        {
        }

        #endregion

        protected override void OnDisconnect(PhotonHostRuntimeInterfaces.DisconnectReason

```

```
reasonCode, string reasonDetail)
{
    // 失去連線時要處理的事項，例如釋放資源
}

protected override void OnOperationRequest(OperationRequest operationRequest,
SendParameters sendParameters)
{
    // 取得Client端傳過來的要求並加以處理
}
}
```

回到 Application，先將 Photon.SocketServer 建立 using 以縮短程式碼，然後修改 CreatePeer 方法建立一個 EZServerPeer 連線，其他的方法我也打上註解解釋該方法的作用。

```
namespace EZServer
{
    public class EZServerApplication : ApplicationBase
    {
        protected override PeerBase CreatePeer(InitRequest initRequest)
        {
            // 建立連線並回傳給Photon Server
            return new EZServerPeer(initRequest.Protocol, initRequest.PhotonPeer);
        }

        protected override void Setup()
        {
            // 初始化GameServer
        }
    }
}
```

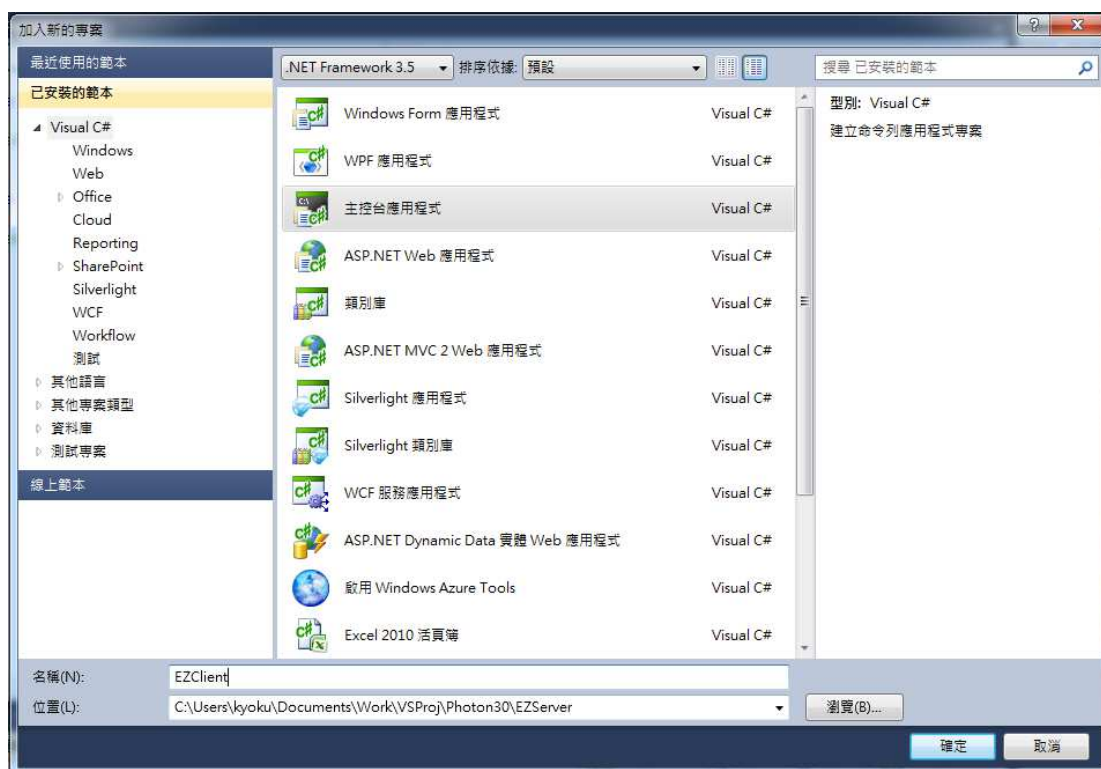


```
protected override void TearDown()
{
    // 關閉GameServer並釋放資源
}
}
```

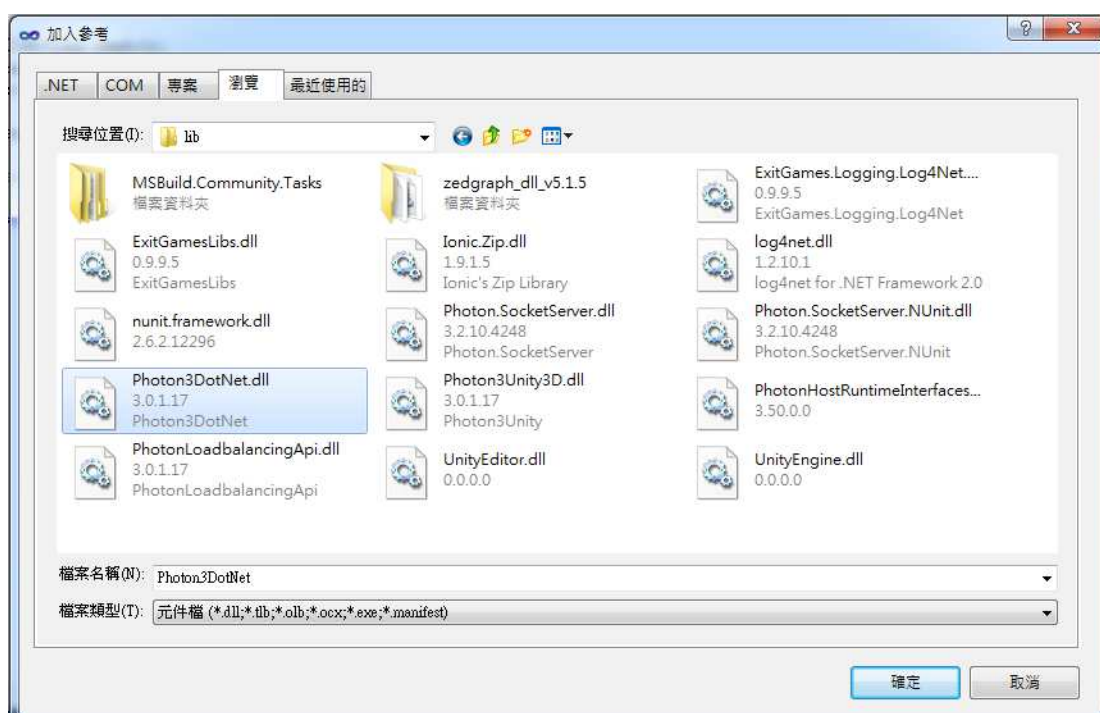
到這裡 Server 的基本框架便完成了，接下來我們要做一個測試用的 client 端及建立我們遊戲所需要的溝通。

## (5). 建立 Client 端測試程式

有了 Server 基本框架，我們現在要建立一個測試的 Client 端，目前別急著動到 Unity3D，我們直接使用 C#來建立 Client 的測試端比較方便也比較快速，在方案中加入一個新的主控台應用程式專案，目標.Net Framework 一樣要設為 3.5，取名為 EZClient。



加入 lib 目錄中的 Photon3DotNet.dll 參考(請參照前面 server 的加入參考方式)。



目前我們只建立一個簡單的溝通，因此只要加入這個 Photon3DotNet 參考即可，至於傳輸所需的 `protocol` 在最早的範例考慮的是讓整個架構越容易閱讀越好，因此在傳輸時直接由 `client` 端傳入命令代碼，`server` 端接收到之後取得數值直接判斷，之後的教學會將這些命令代碼建立成列舉的類別 `dll` 再分別由 `server` 端和 `client` 端讀入加以判讀命令內容。

打開 `Program.cs`，將 `class Program` 繼承自 `IPhotonPeerListener`，然後實作成員，會得到如下的程式碼。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ExitGames.Client.Photon;

namespace EZClient
{
    class Program : IPPhotonPeerListener
    {
        static void Main(string[] args)
        {
        }

        public void DebugReturn(DebugLevel level, string message)
        {
            throw new NotImplementedException();
        }

        public void OnEvent(EventData eventData)
        {
            throw new NotImplementedException();
        }
    }
}
```

```

public void OnOperationResponse(OperationResponse operationResponse)
{
    throw new NotImplementedException();
}

public void OnStatusChanged(StatusCode statusCode)
{
    throw new NotImplementedException();
}
}
}

```

然後在 **Main** 裡建立好遊戲的主迴圈，這裡直接用一個 **do...while** 簡單的代  
表遊戲主迴圈，並將預設的 **Exception** 都拿掉，請參照以下程式碼。

```

namespace EZClient
{
    class Program : IPhotonPeerListener
    {
        static void Main(string[] args)
        {
            var listener = new Program();
            var peer = new PhotonPeer(this, ConnectionProtocol.Udp);

            if (peer.Connect("localhost:5055", "EZServer")) // 若成功連到Server
            {
                do
                {
                    Console.WriteLine("."); // 畫個點代表程式還活著

                    peer.Service();           // 對Server進行觸發動作，當有命令從
                } while (true);
            }
        }
    }
}

```

Server傳過來時依類型OnEvent或OnOperationResponse會被觸發

```
        System.Threading.Thread.Sleep(500); // 休息0.5秒以免鎖死電腦
    }
    while (true);
}
else
{
    Console.WriteLine("Unknown hostname!");
}

}

public void DebugReturn(DebugLevel level, string message)
{
    // 印出Debug的回傳字串
    Console.WriteLine("Debug message : " + message);
}

public void OnEvent(EventData eventData)
{
    // 取得Server傳過來的事件
}

public void OnOperationResponse(OperationResponse operationResponse)
{
    // 取得Server傳過來的命令回傳
}

public void OnStatusChanged(StatusCode statusCode)
{
    // 連線狀態變更的通知

    Console.WriteLine("PeerStatusCallback:" + statusCode.ToString());
}
```

```
switch (statusCode)
{
    case StatusCode.Connect:
        // 若已連線
        break;
}

}

}
```

當我們做好框架並不會主動發生任何的傳輸，必需在主迴圈裡加入 `peer.Service()` 以觸發調度和委派的作業。之後，當 **Server** 有傳訊息過來，我們可以在 `OnEvent`、`OnOperationResponse`、`OnStatusChanged` 等方法裡接收到訊息的內容。

**Peer** 物件在建立時，必需以 `ConnectionProtocol` 指定 UDP 或 TCP 的傳輸方式，**Photon** 的預設傳輸方式是 UDP，這也是大部份 MMO 採用的方式，速度比 TCP 快很多，但資料安全性較差，不過這部份 **Photon** 的可靠傳輸會幫我們處理好，可以放心的使用 UDP 進行傳輸。

當 **Photon** 連線後 `OnStatusChanged` 會接收到 `StatusCode.Connect` 這個命令碼，大多的網路遊戲都是要接收到連成功後才繼續做登入或選取伺服器等動作，可以在此處記錄目前伺服器的連線狀態。

## (6). Server 端和 Client 端的訊息傳輸

現在，我們來為 client 和 server 端建立一個簡單的傳輸，我們要模擬遊戲一開始的 Login 動作，在 client 端讀入帳號及密碼然後送到 server 端，server 端取得該命令之後回傳對應的訊息及結果到 client 端。

我們先處理 client，宣告一個 ServerConnected 的旗標，在完成連線之後我們將連線旗標設為 true 然後開始要求使用者(玩家)輸入帳號及密碼，我改寫了一下 main 並加入 Program 的建構式，並在 PeerStatusCallback 裡加入若是完成連線則將 ServerConnected 變數設為 true。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ExitGames.Client.Photon;

namespace EZClient
{
    class Program : IPhtonPeerListener
    {
        public bool ServerConnected;
        public bool OnLogin;
        PhotonPeer peer;

        public Program()
        {
            ServerConnected = false;
            OnLogin = false;
            this.peer = new PhotonPeer(this, ConnectionProtocol.Udp);
        }
    }
}
```

```

static void Main(string[] args)
{
    new Program().Run();
}

public void Run()
{
    if (peer.Connect("localhost:5055", "EZServer")) //若成功連到Server
    {
        do
        {
            peer.Service();           //在主迴圈取得Server傳過來的命令，當有
            //傳命令過來時依命令的類型，EventAction或OperationResult會被觸發

            if (ServerConnected && !OnLogin)
            {
                Console.WriteLine("\n請輸入會員帳號：");
                string memberID = Console.ReadLine();

                Console.WriteLine("\n請輸入會員密碼：");
                string memberPW = Console.ReadLine();

                var parameter = new Dictionary<byte, object> {
                    { (byte)1, memberID.Trim() },
                    { (byte)2, memberPW.Trim() } // parameter key memberID=1, memberPW=2
                };

                this.peer.OpCustom(5, parameter, true); //命令代碼為5
                OnLogin = true;
            }
        } while (true);
    }
}

```



```

        System.Threading.Thread.Sleep(500); //休息.5秒以免鎖死電腦

    }
    while (true);
}
else
{
    Console.WriteLine("Unknown hostname!");
}
}

```

.....  
 .....

```

public void OnStatusChanged(StatusCode statusCode)
{
    // 連線狀態變更的通知

    Console.WriteLine("PeerStatusCallback:" + statusCode.ToString());

    switch (statusCode)
    {
        case StatusCode.Connect:
            // 若已連線
            ServerConnected = true;
            break;

        case StatusCode.Disconnect:
            // 若已離線
            ServerConnected = false;
            break;
    }
}

```

```

    }
}
}

```

在主要迴圈若已完成連線則要求玩家輸入帳號及密碼，Photon 3.0 的傳輸是利用 Dictionary 陣列，因此格式很自由(在 Photon 2.X 是使用 Hashtable 傳輸，到 3.0 則統一使用較新的 Dictionary)，建立好 parameter 後呼叫 OpCustom 將命令傳到 Server 端，這裡設命令的代碼 5 代表是登入，這是這裡自己訂的，不是絕對的，可任意自訂，每個遊戲應該都會制定一套傳輸代碼的列表。

```

Console.WriteLine("\n請輸入會員帳號：");
string memberID = Console.ReadLine();

Console.WriteLine("\n請輸入會員密碼：");
string memberPW = Console.ReadLine();

var parameter = new Dictionary<byte, object> {
    { (byte)1, memberID.Trim() },
    { (byte)2, memberPW.Trim() }
    // parameter key memberID=1, memberPW=2
};

this.peer.OpCustom(5, parameter, true); //命令代碼為5

```

**OpCustom 的參數：**

OpCustom(

```

byte customOpCode,
Dictionary<byte,object> customOpParameters,
bool sendReliable,
byte channelId,
bool encrypt
)
customOpCode : 命令代碼
customOpParameters : 要傳的內容
sendReliable : true 代表可靠傳輸，false 代表不可靠傳輸
channelId : 頻道
encrypt: true 加密傳輸，false 不加密傳輸，預設為不加密傳輸

```

回到 Server 端 EZServerPeer.cs，改寫 OnOperationRequest 方法，取得傳入值 request 後利用 switch 解讀命令代碼，剛才我們已假設 Login 代碼為 5，所以這裡直接用 case 5: 去處理登入命令。

此處只先介紹架構不想增加額外的學習負擔，因此這裡不讀資料庫而是直接判斷帳號密碼是否為 test,1234，若是，則傳回成功及會員資料(這裡資料只有重新回傳帳號密碼及一個 Ret 和暱稱)，否則傳回錯誤。

```

protected override void OnOperationRequest(OperationRequest
operationRequest, SendParameters sendParameters)
{
    //取得Client端傳過來的要求並加以處理

    switch (operationRequest.OperationCode)
    {
        case 5:
        {
            if (operationRequest.Parameters.Count < 2) //若參數小於2
            則返回錯誤
            {

```

```

//返回登入錯誤
OperationResponse response = new
OperationResponse(operationRequest.OperationCode) { ReturnCode = (short)2,
DebugMessage = "Login Fail" };
SendOperationResponse(response, new SendParameters());
}
else
{
var memberID = (string)operationRequest.Parameters[1];
var memberPW = (string)operationRequest.Parameters[2];

if (memberID == "test" && memberPW == "1234")
{
int Ret = 1;
var parameter = new Dictionary<byte, object> {
{ 80, Ret }, {1, memberID}, {2,
memberPW},
{3, "Kyoku"} // 80代表回傳值, 3代表暱
稱
};

OperationResponse response = new
OperationResponse(operationRequest.OperationCode,
parameter
) { ReturnCode = (short)0, DebugMessage = "" };
SendOperationResponse(response, new
SendParameters());

}
else
{
OperationResponse response = new
OperationResponse(operationRequest.OperationCode) { ReturnCode = (short)1,

```

```

DebugMessage = "Wrong id or password" };

        SendOperationResponse(response, new
SendParameters());
    }

}

break;
}

}

}

```

OnOperationRequest 是接收 Client 傳過來的要求的實作覆載方法，接收到後再以 switch ... case ... 去判斷內容並回傳適當的值給 Client 端，回傳的方式是透過「SendOperationResponse」這個方法。

#### SendOperationResponse 參數說明：

```

public SendResult SendOperationResponse(
    OperationResponse operationResponse,
    SendParameters sendParameters
)

```

operationResponse：要回傳的內容

sendParameters：回傳的規格

SendOperationResponse 必需傳入兩個物件，分別說明如下。

#### OperationResponse 的成員說明：

byte OperationCode：命令代碼，通常和要求時的命令代表相同，代表該要求的回傳值

Dictionary<byte, Object> Parameters : 回傳的內容  
 short ReturnCode : 回傳狀態的值，通常 0 代表成功，此值可以自訂其意義  
 string DebugMessage : 回傳的說明

通常 Parameters 會放入從資料庫取回的內容，然後將 ReturnCode 填入成功與否的代碼後回傳給 Client。

### SendParameters 的成員說明：

byte ChannelId : 頻道，通常都是用第 0 的頻道，除非特別的需求否則不建議更改  
 bool Encrypted : 是否加密，true 為加密，預設是 false  
 bool Flush : 立即傳輸，會讓 queue 尚未傳送的往後延，預設為 false  
 bool Unreliable : 可靠傳輸，預設為 true

再回到 client 端，我們來接收並處理 Server 端的回傳值，打開 client 端 Program.cs 改寫 OnOperationResponse 如下。

```
public void OnOperationResponse(OperationResponse operationResponse)
{
    // 取得Server傳過來的命令回傳

    switch (operationResponse.OperationCode)
    {
        case (byte)5: // 在此範例是訂5為Login要求
        {
            switch (operationResponse.ReturnCode)
            {
                case (short)0: // ReturnCode 0 代表成功
                {
                    int Ret =
```

```

Convert.ToInt16(operationResponse.Parameters[(byte)80]);

        string memberID =
Convert.ToString(operationResponse.Parameters[(byte)1]);

        string memberPW =
Convert.ToString(operationResponse.Parameters[(byte)2]);

        string Nickname =
Convert.ToString(operationResponse.Parameters[(byte)3]);

        Console.WriteLine("Login Success \nRet={0}
\nMemberID={1} \nMemberPW={2} \nNickname={3} \n", Ret, memberID, memberPW, Nickname);

        break;
    }
    case (short)1: // 帳號或密碼錯誤
    {
        Console.WriteLine(operationResponse.DebugMessage);
        break;
    }
    case (short)2: // 傳入的參數錯誤
    {
        Console.WriteLine(operationResponse.DebugMessage);
        break;
    }
    default:
    {
        Console.WriteLine(String.Format("不明的ReturnCode :
{0}", operationResponse.ReturnCode));

        break;
    }
}

break;
}

default:
{

```

```
        Console.WriteLine(String.Format("不明的OperationCode : {0}",  
operationResponse.OperationCode));  
        break;  
    }  
}  
}
```

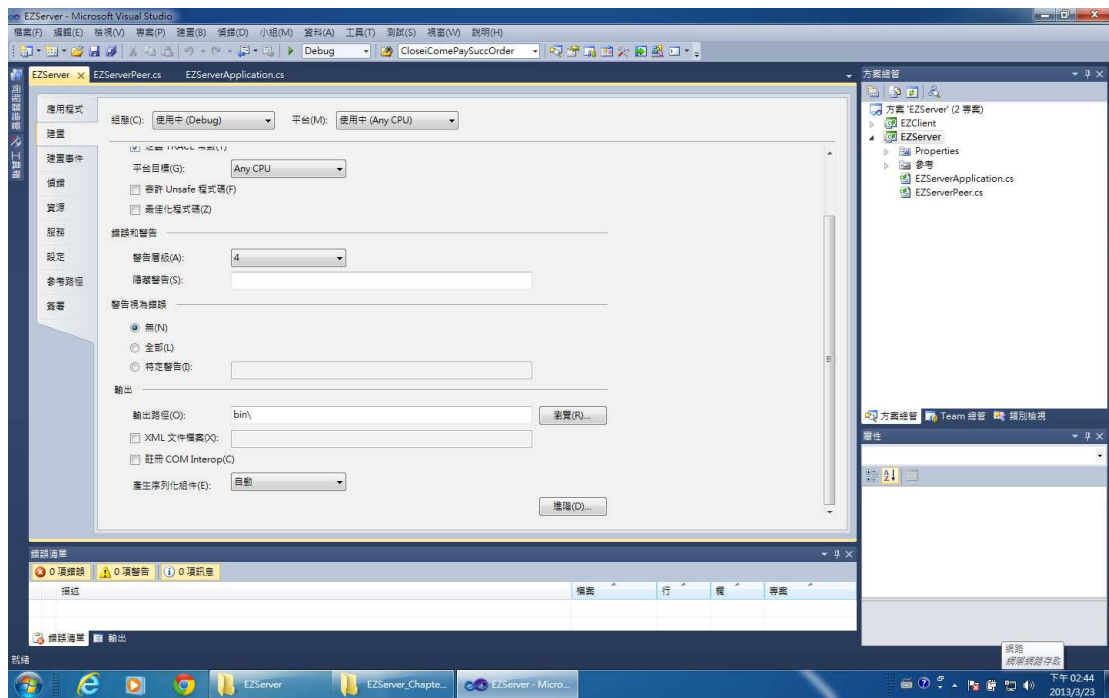
當我們取得由 Server 回傳的內容後，先判斷 OperationCode 是哪個要求的回傳值，若是 Login 的回傳，再去判斷 ReturnCode 是否為成功，若成功則從 Dictionary 取得回傳的內容並列印出來，若失敗則印出 Server 傳過來的 DebugMessage。

下一節我們要開始來測試我們的 Svrer 了。

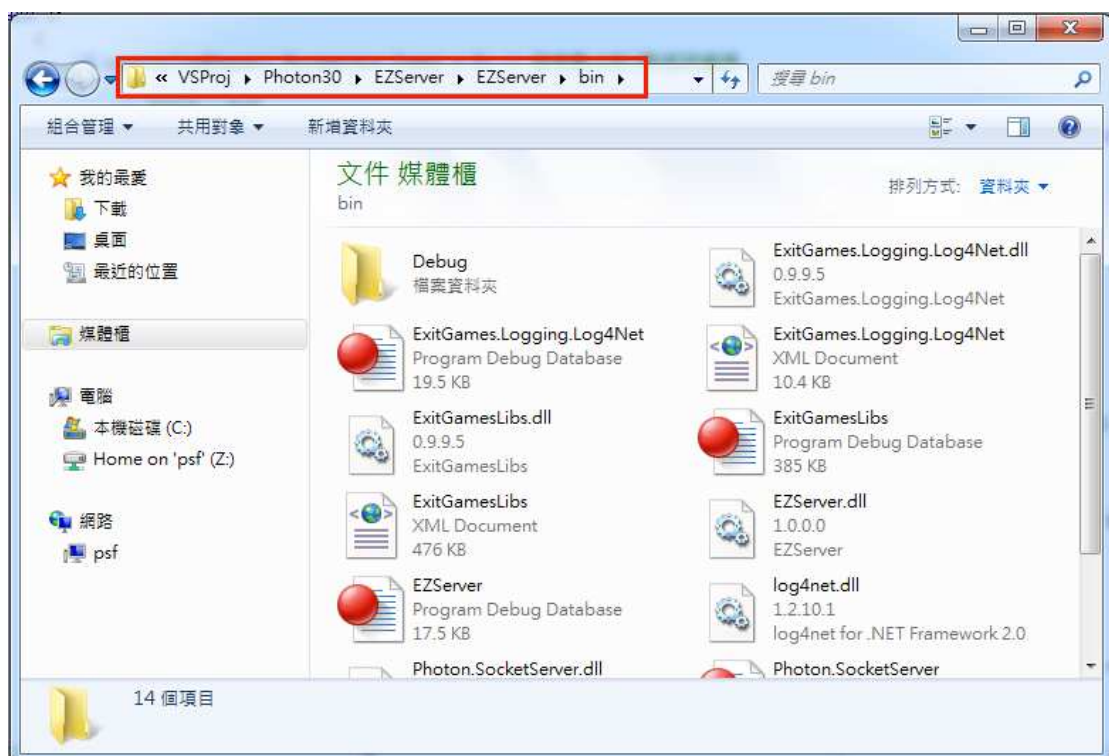
## 2. 測試

現在要開始來測試我們所寫的 Server，請先開啟 Server 的專案屬性，切到「建置」並將輸出路徑改成「bin\」。

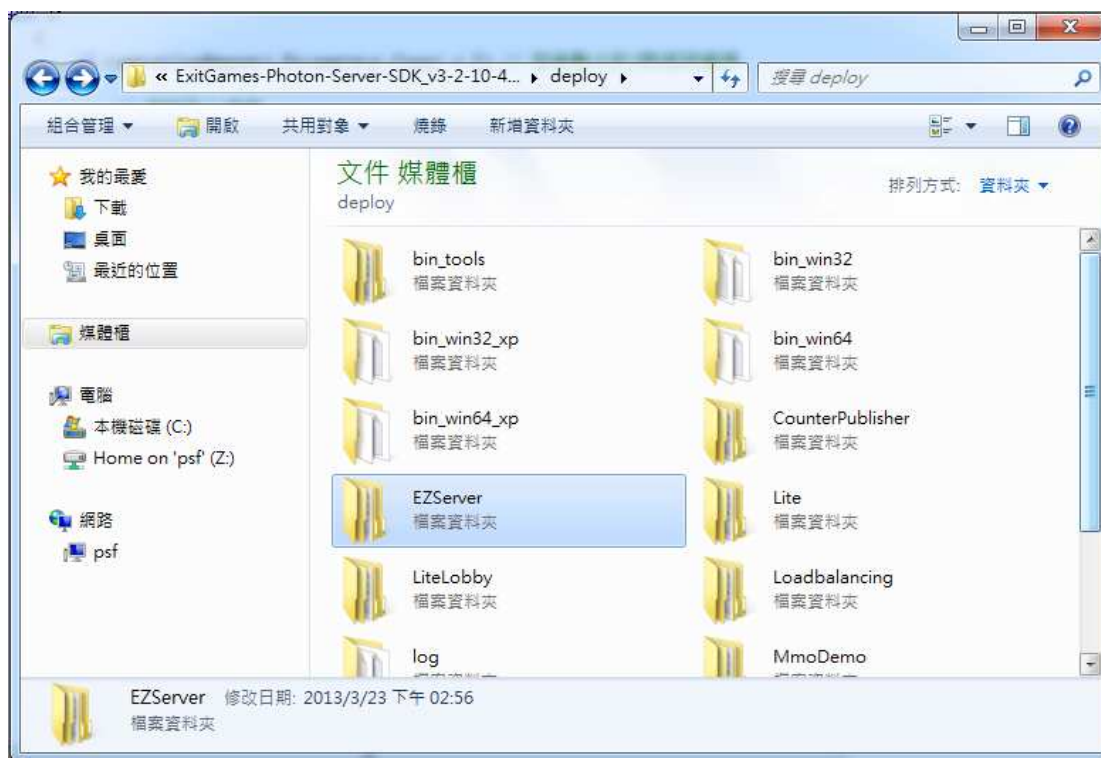




然後建置專案，建置後可以看到編譯好的 dll 放在「EZServer\bin」底下，Photon 會去掛載 bin 底下的 dll，所以一定要以這種資料存放方式，當然，你也可以編譯好之後自己建一個資料夾並在底下建立好 bin 的子資料夾然後將所有編好的檔案複製過去。



然後將整個 EZServer 專案 copy 到 Photon 所在的資料夾，copy 過來後你可以將資料夾底下的 code 或其他檔案都刪除，只留下 bin 這個資料夾即可，Photon 只需要編輯好的 bin 便可以運作。



之後開啟你執行 Photon 的那個目錄底下(視 win xp 或 win server 而定)，若您用的是 Windows XP/Windows Server 2003 請開啟「bin\_win32\_xp」的資料夾，若您用的是 Windows7 之後的作業系統請開啟「bin\_win32」的資料夾，您也可以使用 64 的資料夾，但若使用 64 則在建立專案時就要建立為 64 位元的專案，大部份開發自己的東西不會有問題，但若使用第三廠商開發的元件就必須有提供 64 位元的元件才能正確運行，為了相容性，本書都使用 32 bits 進行開發，讀者若有需求可以自行更改，開啟好資料夾後底下有一個名為「PhotonServer.config」的檔案，直接用記事本開啟。

找到 Application 節點，加入 EZServer 的節點。

```
<?xml version="1.0" encoding="Windows-1252"?>
```

```

<!--
(c) 2010 by Exit Games GmbH, http://www.exitgames.com

Photon server configuration file.

For details see the photon-config.pdf.

This file contains two configurations:

    "Instance1"
        Default. Various applications and demos.
        Starts the apps: Lite, LiteLobby, MmoDemo, CounterPublisher and Policy
        Listens: udp-port 5055, tcp-port: 4530, 843 and 943
    "InstanceLoadBalancing"
        Loadbalanced setup for local development: A Master-server and two
game-servers.
        Starts the apps: Game1, Game2, Master, CounterPublisher and Policy
        Listens: udp-port 5055, tcp-port: 4530, 843 and 943
-->

<Configuration>
    <!-- Multiple instances are supported. Each instance has its own node in the config file. -->
    <!-- PhotonControl will currently only start "Instance1" but the .cmd files could be modified to
start other instances. -->

    <!-- Instance settings -->
    <Instance1
        MaxMessageSize="512000"
        MaxQueuedDataPerPeer="512000"
        PerPeerMaxReliableDataInTransit="51200"
        PerPeerTransmitRateLimitKBSec="256"
        PerPeerTransmitRatePeriodMilliseconds="200"
        MinimumTimeout="5000"
        MaximumTimeout="30000">

    <!-- 0.0.0.0 opens listeners on all available IPs. Machines with multiple IPs should define

```

the correct one here. -->

```
<!-- Port 5055 is Photon's default for UDP connections. -->
```

```
<UDPListeners>
```

```
  <UDPListener
```

```
    IPAddress="0.0.0.0"
```

```
    Port="5055">
```

```
  </UDPListener>
```

```
</UDPListeners>
```

<!-- 0.0.0.0 opens listeners on all available IPs. Machines with multiple IPs should define the correct one here. -->

```
<!-- Port 5055 is Photon's default for TCP connections. -->
```

```
<TCPListeners>
```

```
  <TCPListener
```

```
    IPAddress="0.0.0.0"
```

```
    Port="4530">
```

```
  </TCPListener>
```

```
</TCPListeners>
```

```
<!-- Policy request listener for Unity and Flash (port 843) and Silverlight (port 943) -->
```

```
<TCPPolicyListeners>
```

```
  <!-- multiple Listeners allowed for different ports -->
```

```
  <TCPPolicyListener
```

```
    IPAddress="0.0.0.0"
```

```
    Port="843"
```

```
    Application="Policy">
```

```
  </TCPPolicyListener>
```

```
  <TCPPolicyListener
```

```
    IPAddress="0.0.0.0"
```

```
    Port="943"
```

```
    Application="Policy">
```

```
  </TCPPolicyListener>
```

```
</TCPPolicyListeners>
```

```
<!-- WebSocket (and Flash-Fallback) compatible listener -->
```

```
<WebSocketListeners>
```

```
  <WebSocketListener
```

```
    IPAddress="0.0.0.0"
```

```
    Port="9090"
```

```
    DisableNagle="true"
```

```
    InactivityTimeout="10000"
```

```
    OverrideApplication="Lite">
```

```
  </WebSocketListener>
```

```
</WebSocketListeners>
```

```
<!-- Defines the Photon Runtime Assembly to use. -->
```

```
<Runtime
```

```
  Assembly="PhotonHostRuntime, Culture=neutral"
```

```
  Type="PhotonHostRuntime.PhotonDomainManager"
```

```
  UnhandledExceptionPolicy="Ignore">
```

```
</Runtime>
```

```
<!-- Defines which applications are loaded on start and which of them is used by default.
```

```
Make sure the default application is defined. -->
```

```
<!-- Application-folders must be located in the same folder as the bin_win32 folders. The
```

```
BaseDirectory must include a "bin" folder. -->
```

```
<Applications Default="Lite">
```

```
  <!-- Lite Application -->
```

```
  <Application
```

```
    Name="Lite"
```

```
    BaseDirectory="Lite"
```

```
    Assembly="Lite"
```

```
    Type="Lite.LiteApplication"
```

```
    EnableAutoRestart="true"
```

```
    WatchFiles="dll;config"
```

```
    ExcludeFiles="log4net.config">
```

```
  </Application>
```

```

<!-- LiteLobby Application -->
<Application
    Name="LiteLobby"
    BaseDirectory="LiteLobby"
    Assembly="LiteLobby"
    Type="LiteLobby.LiteLobbyApplication"
    EnableAutoRestart="true"
    WatchFiles="dll;config"
    ExcludeFiles="log4net.config">

</Application>

<!-- MMO Demo Application -->
<Application
    Name="MmoDemo"
    BaseDirectory="MmoDemo"
    Assembly="Photon.MmoDemo.Server"
    Type="Photon.MmoDemo.Server.PhotonApplication"
    EnableAutoRestart="true"
    WatchFiles="dll;config"
    ExcludeFiles="log4net.config">

</Application>

<!-- EZServer Application -->
<Application
    Name="EZServer"
    BaseDirectory="EZServer"
    Assembly="EZServer"
    Type="EZServer.EZServerApplication"
    EnableAutoRestart="true"
    WatchFiles="dll;config"
    ExcludeFiles="log4net.config">

</Application>

```

```

<!-- CounterPublisher Application -->
<Application
  Name="CounterPublisher"
  BaseDirectory="CounterPublisher"
  Assembly="CounterPublisher"
  Type="Photon.CounterPublisher.Application"
  EnableAutoRestart="true"
  WatchFiles="dll;config"
  ExcludeFiles="log4net.config">
</Application>

<!-- Flash & Silverlight Policy Server -->
<Application
  Name="Policy"
  BaseDirectory="Policy"
  Assembly="Policy.Application"
  Type="Exitgames.Realtime.Policy.Application.Policy">
</Application>
</Applications>
</Instance1>

<InstanceLoadBalancing
  MaxMessageSize="512000"
  MaxQueuedDataPerPeer="512000"
  PerPeerMaxReliableDataInTransit="51200"
  PerPeerTransmitRateLimitKBSec="256"
  PerPeerTransmitRatePeriodMilliseconds="200"
  MinimumTimeout="5000"
  MaximumTimeout="30000">

  <!-- 0.0.0.0 opens listeners on all available IPs. Machines with multiple IPs should define
the correct one here. -->

  <!-- Port 5055 is Photon's default for UDP connections. -->
<UDPListeners>

```

```

<UDPListener
    IPAddress="0.0.0.0"
    Port="5055"
    OverrideApplication="Master">
</UDPListener>

<UDPListener
    IPAddress="0.0.0.0"
    Port="5056"
    OverrideApplication="Game1">
</UDPListener>

<UDPListener
    IPAddress="0.0.0.0"
    Port="5057"
    OverrideApplication="Game2">
</UDPListener>
</UDPListeners>

<!-- 0.0.0.0 opens listeners on all available IPs. Machines with multiple IPs should define
the correct one here. -->

<!-- Port 5055 is Photon's default for TCP connections. -->
<TCPListeners>
    <!-- TCP listener for Game clients on Master application -->
    <TCPListener
        IPAddress="0.0.0.0"
        Port="4530"
        OverrideApplication="Master">
    </TCPListener>

    <TCPListener
        IPAddress="0.0.0.0"
        Port="4531"
        OverrideApplication="Game1">
    </TCPListener>

```



```

<TCPListener
    IPAddress="0.0.0.0"
    Port="4532"
    OverrideApplication="Game2">
</TCPListener>

<!-- DON'T EDIT THIS. TCP listener for GameServers on Master application -->
<TCPListener
    IPAddress="0.0.0.0"
    Port="4520">
</TCPListener>
</TCPListeners>

<!-- Policy request listener for Unity and Flash (port 843) and Silverlight (port 943) -->
<TCPPolicyListeners>
    <!-- multiple Listeners allowed for different ports -->
    <TCPPolicyListener
        IPAddress="0.0.0.0"
        Port="843"
        Application="Policy">
    </TCPPolicyListener>
</TCPPolicyListeners>

<!-- Defines the Photon Runtime Assembly to use. -->
<Runtime
    Assembly="PhotonHostRuntime, Culture=neutral"
    Type="PhotonHostRuntime.PhotonDomainManager"
    UnhandledExceptionPolicy="Ignore">
</Runtime>

<!-- Defines which applications are loaded on start and which of them is used by default.
Make sure the default application is defined. -->

    <!-- Application-folders must be located in the same folder as the bin_win32 folders. The
BaseDirectory must include a "bin" folder. -->

```

```

<Applications Default="Master">
  <Application
    Name="Master"
    BaseDirectory="LoadBalancing\Master"
    Assembly="Photon.LoadBalancing"
    Type="Photon.LoadBalancing.MasterServer.MasterApplication"
    EnableAutoRestart="true"
    WatchFiles="dll;config"
    ExcludeFiles="log4net.config"
  >
</Application>
<Application
  Name="Game1"
  BaseDirectory="LoadBalancing\GameServer1"
  Assembly="Photon.LoadBalancing"
  Type="Photon.LoadBalancing.GameServer.GameApplication"
  EnableAutoRestart="true"
  WatchFiles="dll;config"
  ExcludeFiles="log4net.config">
</Application>
<Application
  Name="Game2"
  BaseDirectory="LoadBalancing\GameServer2"
  Assembly="Photon.LoadBalancing"
  Type="Photon.LoadBalancing.GameServer.GameApplication"
  EnableAutoRestart="true"
  WatchFiles="dll;config"
  ExcludeFiles="log4net.config">
</Application>
<Application
  Name="Policy"
  BaseDirectory="Policy"
  Assembly="Policy.Application"
  Type="Exitgames.Realtime.Policy.Application.Policy"

```

```

        EnableAutoRestart="true"
        WatchFiles="dll;config.xml"
        ExcludeFiles="log4net.config">
    </Application>

    <!-- CounterPublisher Application -->
    <Application
        Name="CounterPublisher"
        BaseDirectory="CounterPublisher"
        Assembly="CounterPublisher"
        Type="Photon.CounterPublisher.Application"
        EnableAutoRestart="true"
        WatchFiles="dll;config"
        ExcludeFiles="log4net.config">
    </Application>
</Applications>
</InstanceLoadBalancing>
</Configuration>

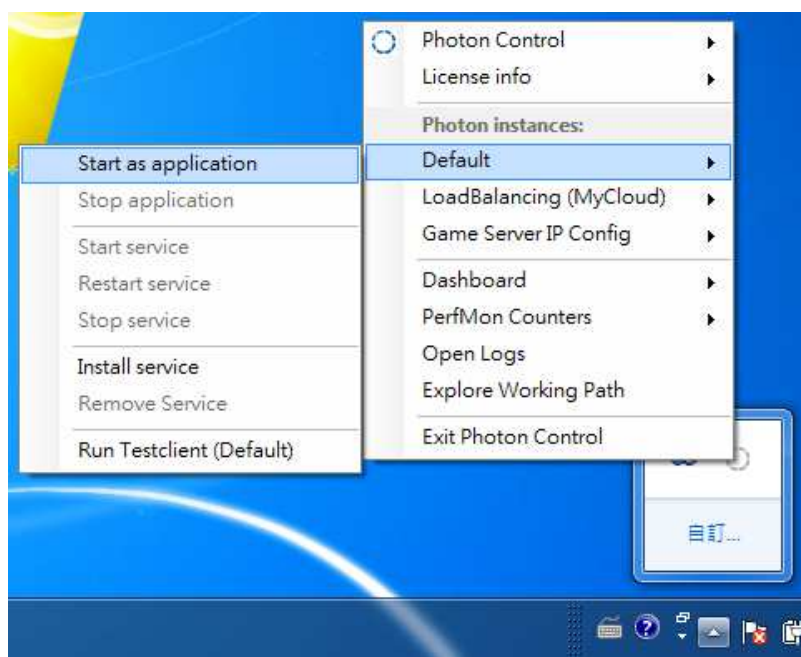
```

其中這個 Name="EZServer" 就是我們 Client 端的指令：

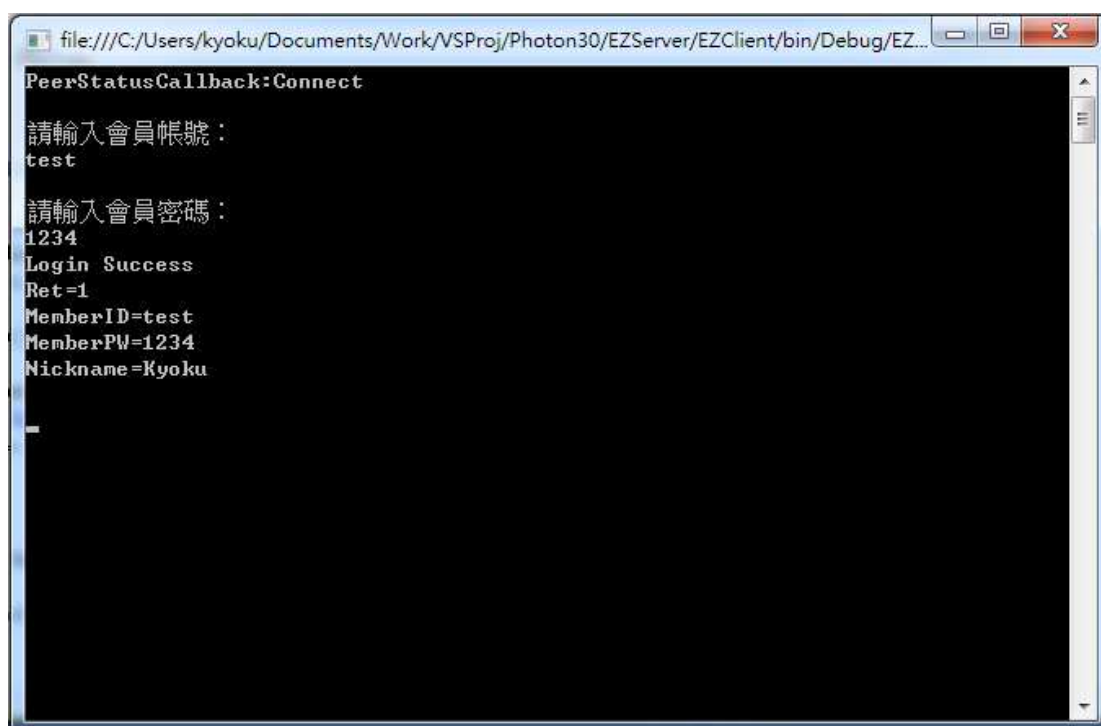
```
peer.Connect("localhost:5055", "EZServer")
```

裡面的 EZServer，若 config 文字檔裡面有修改 client 端程式碼也要跟著改才行。

存好檔請重新啟動 Server，若已啟動 Photon Server 就先 Stop application 再重新 Start as application，若還沒啟動 Photon 請執行先執行「PhotonControl.exe」，在工作列上會出現 Photon 的 icon，接下來在 Photon icon 上按滑鼠右鍵執行「Photon > Default > Start as application」便完成啟動 Photon Server。

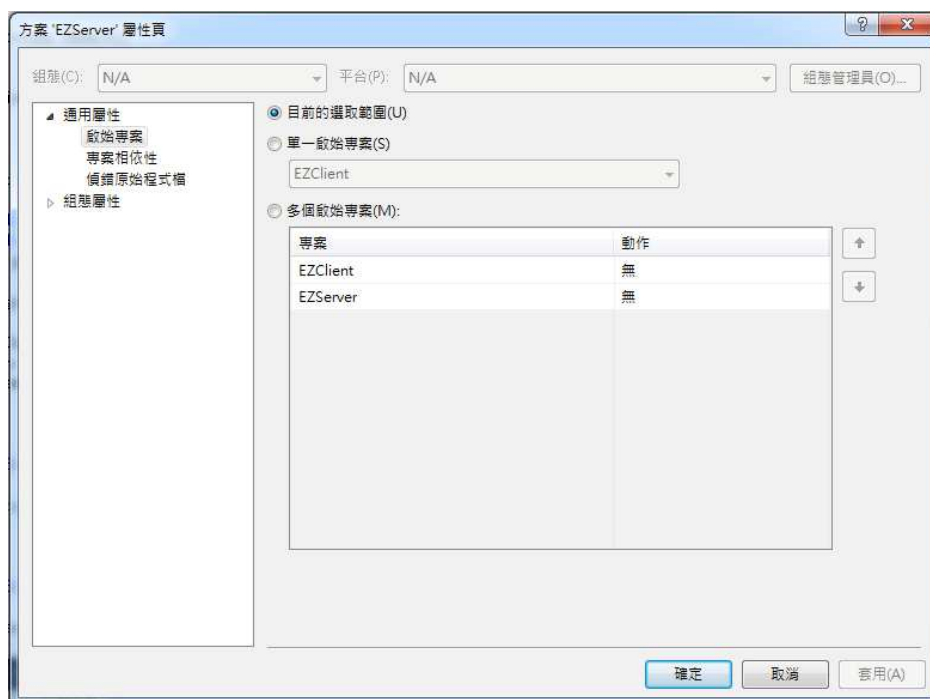
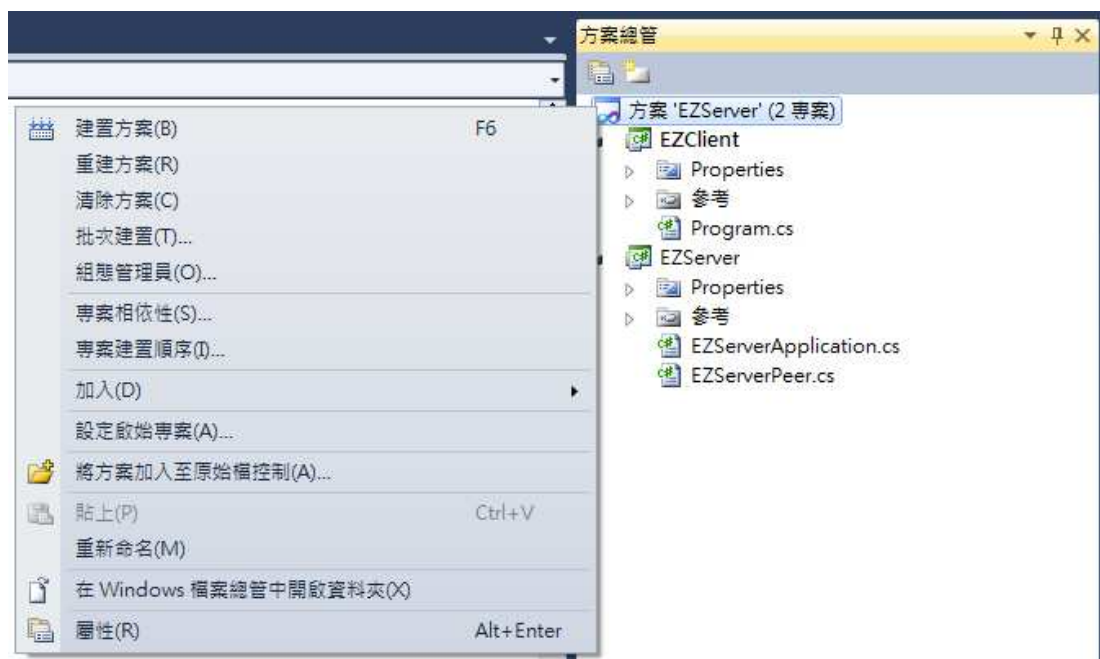


完成後就可以測試 Client 了，直接編譯執行後可以看到這樣的結果。



若是您執行後無法順利的執行 EZClient，而是一直執行到 EZServer 專案而發生錯誤，您可以在 VS 的方案總管上選 EZServer 方案按下滑鼠右鍵選最下面

的屬性，然後將方案的屬性改成「目前的選取範圍」，這樣就能開哪個檔案就執行哪個檔案的專案了。

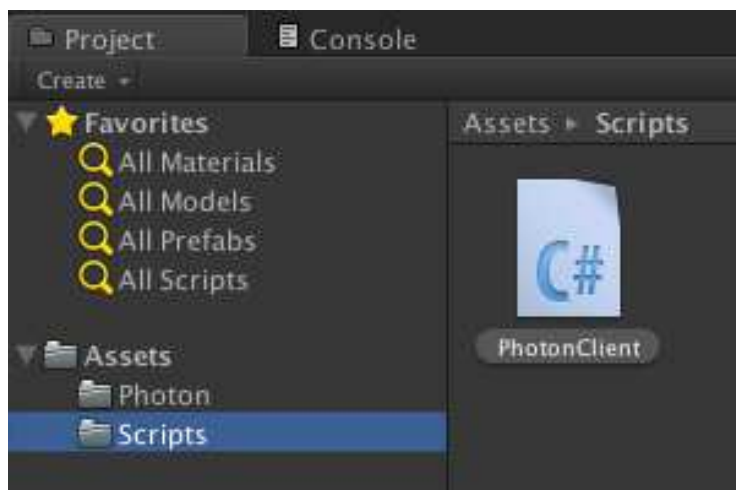
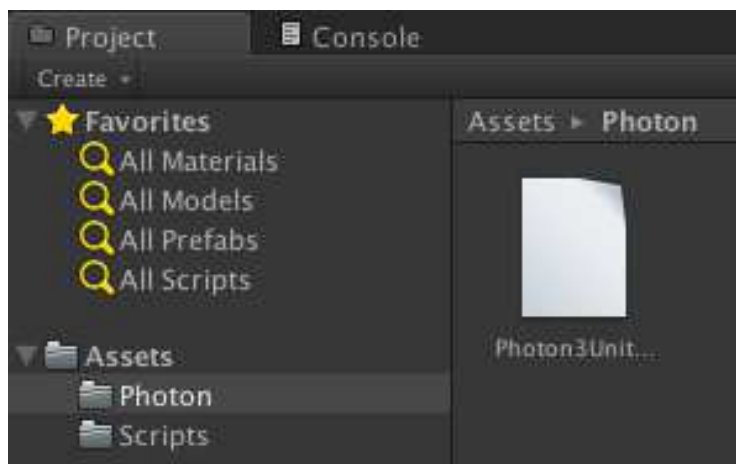


## 四 Unity3D 與 Photon 的連接

### 1. 建立專案

Unity3D 與 Photon 的連結和 C#主控台的指令差不多，請先建立一個新的 Unity 專案，這裡取名為「EZClientUnity」。

接下來將 Photon 底下「lib」資料夾的「PhotonUnity3D.dll」這個檔案拉到 Unity 專案裡面，然後建立一個 C#的 Script，並將這個 script 取名為「PhotonClient.cs」，因為 Unity3D 可以自由的建立子資料夾，因此我習慣用資料夾將 DLL 和 script 整理好，將 Photon3Unity3D.dll 放在 Photon 資料夾底下，然後將自己建立的 PhotonClient.cs 放在 Scripts 資料夾底下。

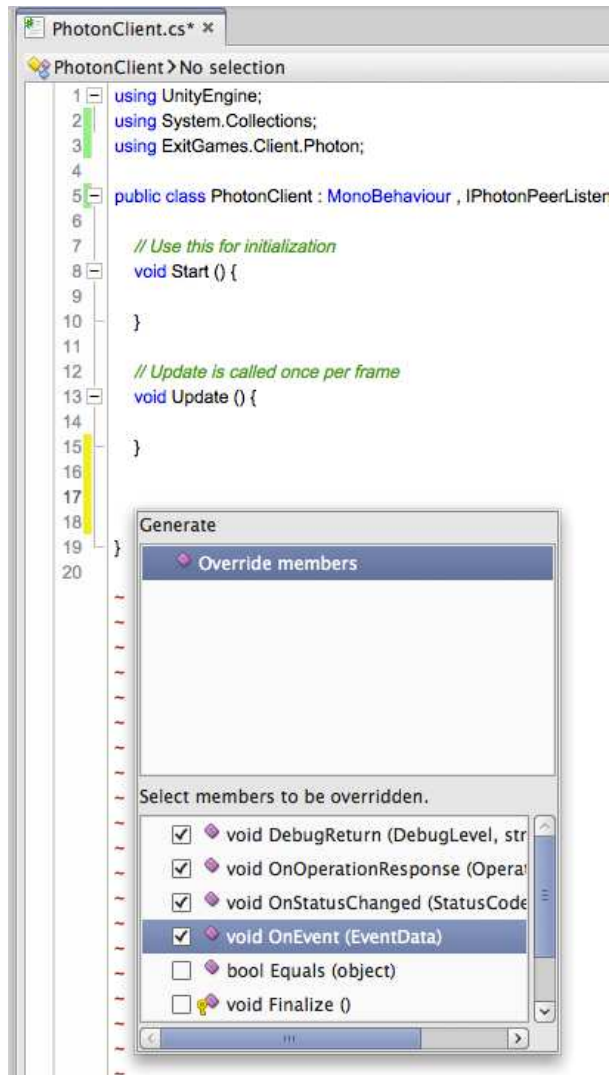


## 2. 建立 Unity for Photon 基本架構

打開 PhotonClient.cs 檔，若 class 名稱為「NewBehaviourScript」請將他更名為「PhotonClient」，Unity 的 Script 裡面第一個類別名稱必需與檔名相同，然後將此 class 加入 IPhotonPeerListener 的繼承，並在最上方加入「using ExitGames.Client.Photon;」。

繼承自 IPhotonPeerListener 之後，因為 IPhotonPeerListener 是一個 Interface 因此必需實作裡面的方法才行，所以要在 PhotonClient.cs 裡加入 4 個方法，DebugReturn, OnOperationResponse, OnStatusChanged 和 OnEvent。

若是使用 MonoDevelop 這個工具撰寫 script，這也是 Unity 預設的程式碼編輯器，裡面有工具可以自動生成員框架，先加入繼承「，IPhotonPeerListener」，再執行「Edit > Show Code Generation Window」可以叫出自動產生成員的視窗，選定要產生的方法成員後按下 enter 自動建立程式碼。



```
using UnityEngine;
using System.Collections;
using ExitGames.Client.Photon;

public class PhotonClient : MonoBehaviour, IPhotonPeerListener {

    // Use this for initialization
    void Start () {

    }

}
```



```

// Update is called once per frame
void Update () {

}

public void DebugReturn (DebugLevel level, string message)
{
    throw new System.NotImplementedException ();
}

public void OnOperationResponse (OperationResponse operationResponse)
{
    throw new System.NotImplementedException ();
}

public void OnStatusChanged (StatusCode statusCode)
{
    throw new System.NotImplementedException ();
}

public void OnEvent (EventData eventData)
{
    throw new System.NotImplementedException ();
}

}

```

這裡補充一點，為何這裡的 **PhotonClient** 類別可以同時繼承自 **MonoBehaviour** 和 **IPhotonPeerListener**，在 C#裡是不能多重繼承的，其實是因為 **IPhotonPeerListener** 是一個 **Interface**，C#雖然不能同時繼承兩個類別，但可以繼承一個類別加上多個 **Interface**。

### 3. 連線到 Server

現在可以看到整個 **Client** 程式框架和之前做的主控台的程式是差不多的，因此我們只要實作出這個框架裡面的訊息處理就可以了，在此之前我們先來建立一個 **GUI** 顯示連線狀態，並在 **Start** 裡面建立與 **Server** 的連線。

```
using UnityEngine;
using System.Collections;
using System.Security;
using ExitGames.Client.Photon;

public class PhotonClient : MonoBehaviour, IPhotonPeerListener {

    public string ServerAddress = "localhost:5055";
    protected string ServerApplication = "EZServer";

    protected PhotonPeer peer;
    public bool ServerConnected;

    // Use this for initialization
    void Start () {

        this.ServerConnected = false;

        this.peer = new PhotonPeer(this, ConnectionProtocol.Udp);
        this.Connect();

    }

    internal virtual void Connect()
    {
```

```

try
{
    this.peer.Connect(this.ServerAddress, this.ServerApplication);
}
catch (SecurityException se)
{
    this.DebugReturn(0, "Connection Failed. " + se.ToString());
}
}

```

// Update is called once per frame

```
void Update () {
```

```
    this.peer.Service();
```

```
}
```

```
public void DebugReturn (DebugLevel level, string message)
```

```
{
```

```
    Debug.Log(message);
```

```
}
```

```
public void OnOperationResponse (OperationResponse operationResponse)
```

```
{
```

```
    throw new System.NotImplementedException ();
```

```
}
```

```
public void OnStatusChanged (StatusCode statusCode)
```

```
{
```

```
    this.DebugReturn(0, string.Format("PeerStatusCallback: {0}", statusCode));
```

```
    switch (statusCode)
```

```
{
```

```
    case StatusCode.Connect:
```

```
        this.ServerConnected = true;
```

```
        break;
```

```

        case StatusCode.Disconnect:
            this.ServerConnected = false;
            break;
    }

}

public void OnEvent (EventData eventData)
{
    throw new System.NotImplementedException ();
}

void OnGUI()
{
    GUILayout.Label(new Rect(30,10,400, 20), "EZServer Unity3D Test");

    if( this.ServerConnected )
    {
        GUILayout.Label(new Rect(30,30,400, 20), "Connected");
    }
    else
    {
        GUILayout.Label(new Rect(30,30,400, 20), "Disconnect");
    }
}

}

```

首先宣告一個 **peer** 並建立連線，例外處理需要再 **using System.Security**; 進來，建立 **peer** 連線方法和之前主控台一模一樣，差別在於我們將

「this.peer.Service();」移到 Update()裡面，因為這個就是 Unity 最主要的迴圈，接下來我們實作出 DebugReturn 方法，直接將 debug 訊息印出來。

然後在 OnStatusChanged 的實作，取得連線狀態變更的事件，若連線完成則將 ServerConnected 變數改為 true。

最下面加入 OnGUI()可以將訊息顯示到遊戲裡畫面上。

若是讀者 Server 和 Client 放在同一台電腦基本上目前這樣就可以進行測試了，但若讀者的 Server 和 Client 放在不同台電腦，那就不能將 Server 的 IP 指向本機的「localhost」，必須將「public string ServerAddress = "localhost:5055";」這一行改成是「public string ServerAddress = "Server 的 IP:5055";」，以筆者開發環境而言，我的 Client 端是使用 Mac 的電腦開發，Server 端是利用 Parallels 安裝 Windows7，因此必需先取得 Server 端的 IP，先到 Windows 7 底下，執行「開始 > 附屬應用程式 > 命令提示字元」，打開後執行「ipconfig /all」接下 enter 便可顯示出 Server 所在的 IP，如下圖可以看到 Server 的 IP 是「10.211.55.3」。

```

Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\kyoku>ipconfig /all

Windows IP 設定

主機名稱 . . . . . : 8B0A
主要 DNS 尾碼 . . . . . :
節點類型 . . . . . : 混合式
IP 路由啟用 . . . . . : 否
WINS Proxy 啟用 . . . . . : 否
DNS 尾碼搜尋清單 . . . . . : localdomain

乙太網路卡 區域連線:

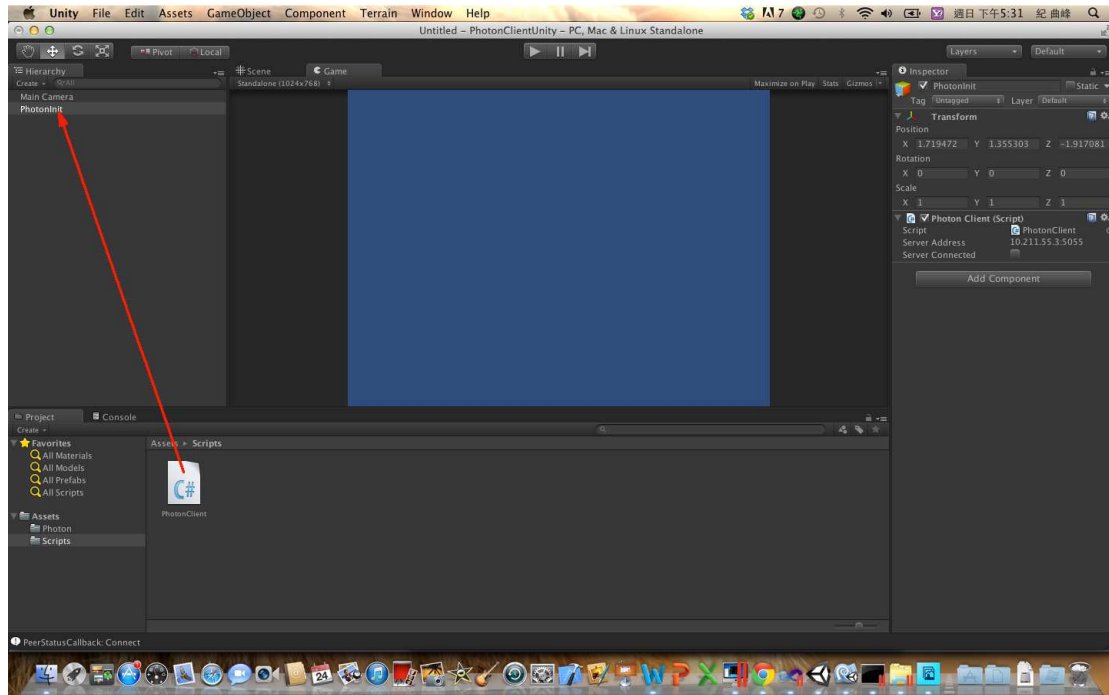
連線特定 DNS 尾碼 . . . . . : localdomain
描述 . . . . . : Intel(R) PRO/1000 MT Network Connection
實體位址 . . . . . : 00-1C-42-D5-53-1D
DHCP 已啟用 . . . . . : 是
自動設定啟用 . . . . . : 是
IPv6 位址 . . . . . : fdb2:2c26:f4e4:0:f4f8:e2ff:3da1:7b5d< 偏好
選項>
臨時 IPv6 位址 . . . . . : fdb2:2c26:f4e4:0:646f:cb22:ea5e:a8f2< 偏好
選項>
連結-本機 IPv6 位址 . . . . . : fe80::f4f8:e2ff:3da1:7b5d< 偏好選項>
IPv4 位址 . . . . . : 10.211.55.3< 偏好選項>
子網路遮罩 . . . . . : 255.255.255.0
租用取得 . . . . . : 2013年3月24日 下午 12:52:52
租用到期 . . . . . : 2013年3月24日 下午 05:22:55
預設閘道 . . . . . : 10.211.55.1
DHCP 伺服器 . . . . . : 10.211.55.1
DNS 伺服器 . . . . . : 10.211.55.1
NetBIOS over Tcpip . . . . . : 啟用
  
```

接下來修改程式碼為如下，或是之後從 Inspector 上修改，因為是 public 變

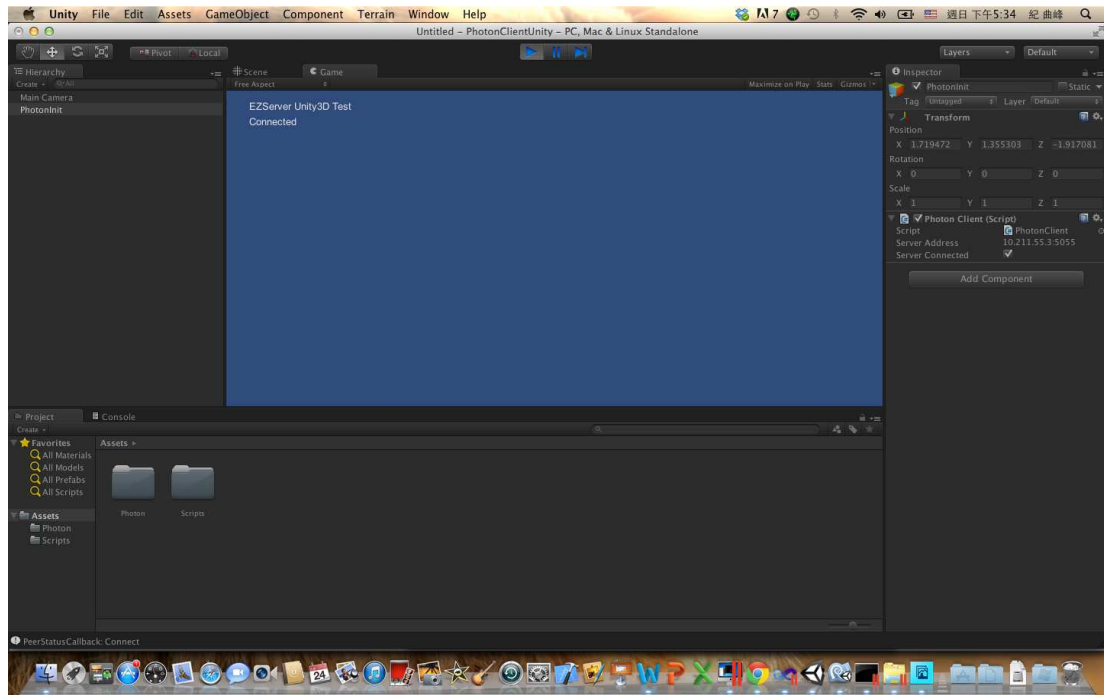
數

```
public string ServerAddress = "10.211.55.3:5055";
```

完成後建立一個 EmptyObject 並更名為 PhotonInit，把 PhotonClient 拉到該物件上。(也可以直接拉到 Camera 或其他任意物件上)



執行後結果如下圖



## 4. Unity 與 Server 的資料傳輸

為了測試，這裡建立兩個 TextField 讓使用者輸入來模擬會員登入，然後實作出 OnOperationResponse，登入完成就顯示會員的資訊，若登入失敗則顯示「Please Login」並顯示出錯誤的訊息，所有的命令代碼都和前一個主控台範例一樣，因此不需修改 Server 端。

```
using UnityEngine;
using System.Collections;
using System.Security;
using System;
using System.Collections.Generic;
using ExitGames.Client.Photon;

public class PhotonClient : MonoBehaviour, IPhotonPeerListener {
```

```

public string ServerAddress = "localhost:5055";
protected string ServerApplication = "EZServer";

protected PhotonPeer peer;
public bool ServerConnected;

public string memberID = "";
public string memberPW = "";

public bool LoginStatus;
public string getMemberID = "";
public string getMemberPW = "";
public string getNickname = "";
public int getRet = 0;

public string LoginResult = "";

// Use this for initialization
void Start () {

    this.ServerConnected = false;
    this.LoginStatus = false;

    this.peer = new PhotonPeer(this, ConnectionProtocol.Udp);
    this.Connect();

}

internal virtual void Connect()
{
    try
    {
        this.peer.Connect(this.ServerAddress, this.ServerApplication);
    }

```



```

        catch (SecurityException se)
        {
            this.DebugReturn(0, "Connection Failed. " + se.ToString());
        }
    }

    // Update is called once per frame
    void Update () {
        this.peer.Service();
    }

    public void DebugReturn (DebugLevel level, string message)
    {
        Debug.Log(message);
    }

    public void OnOperationResponse (OperationResponse operationResponse)
    {
        // display operationCode
        this.DebugReturn(0, string.Format("OperationResult:" +
operationResponse.OperationCode.ToString()));

        switch (operationResponse.OperationCode)
        {
            case 5:
            {
                if( operationResponse.ReturnCode == 0) // if success
                {
                    getRet = Convert.ToInt32(operationResponse.Parameters[80]);
                    getMemberID =
Convert.ToString(operationResponse.Parameters[1]);
                    getMemberPW=
Convert.ToString(operationResponse.Parameters[2]);

```

```

        getNickname= Convert.ToString(operationResponse.Parameters[3]);
        LoginStatus = true;

    }
    else
    {
        LoginResult = operationResponse.DebugMessage;
        LoginStatus = false;
    }
    break;
}

}

}

public void OnStatusChanged (StatusCode statusCode)
{
    this.DebugReturn(0, string.Format("PeerStatusCallback: {0}", statusCode));
    switch (statusCode)
    {
        case StatusCode.Connect:
            this.ServerConnected = true;
            break;
        case StatusCode.Disconnect:
            this.ServerConnected = false;
            break;
    }
}

}

public void OnEvent (EventData eventData)
{
    throw new System.NotImplementedException ();
}

```

```

void OnGUI()
{
    GUI.Label(new Rect(30,10,400, 20), "EZServer Unity3D Test");

    if( this.ServerConnected )
    {
        GUI.Label(new Rect(30,30,400, 20), "Connected");

        GUI.Label(new Rect(30,60,80, 20), "MemberID:");
        memberID = GUI.TextField(new Rect(110, 60, 100, 20), memberID, 10);

        GUI.Label(new Rect(30,90,80, 20), "MemberPW:");
        memberPW = GUI.TextField(new Rect(110, 90, 100, 20), memberPW, 10);

        if( GUI.Button( new Rect(30,120,100,24 ), "Login" ) )
        {
            var parameter = new Dictionary<byte, object> {
                { (byte)1, memberID },    { (byte)2, memberPW } // parameter
key memberID=1, memberPW=2
            };

            this.peer.OpCustom(5, parameter, true); // operationCode is 5

        }

        if( LoginStatus )
        {
            GUI.Label(new Rect(30,150,400, 20), "Your MemberID : " + getMemberID);
            GUI.Label(new Rect(30,170,400, 20), "Your Password : " + getMemberPW);
            GUI.Label(new Rect(30,190,400, 20), "Your Nickname : " + getNickname);
            GUI.Label(new Rect(30,210,400, 20), "Ret : " + getRet.ToString());

        }
    }
}

```

```
        else
        {
            GUI.Label(new Rect(30,150,400, 20), "Please Login");
            GUI.Label(new Rect(30,170,400, 20), LoginResult);
        }

    }
    else
    {
        GUI.Label(new Rect(30,30,400, 20), "Disconnect");
    }
}

}
```

除了公用變數和最後面 GUI 處理外其他都和前面主控台程式差不多，我們在 `OnOperationResponse` 取得登入的回傳資料，先判斷 `ReturnCode` 是否為 0，若是 0 表示登入成功，成功時將資料後存到變數裡，然後再由 GUI 印出其結果，若失敗則顯示 Server 回傳的 `DebugMessage`，最後的執行結果如下圖。

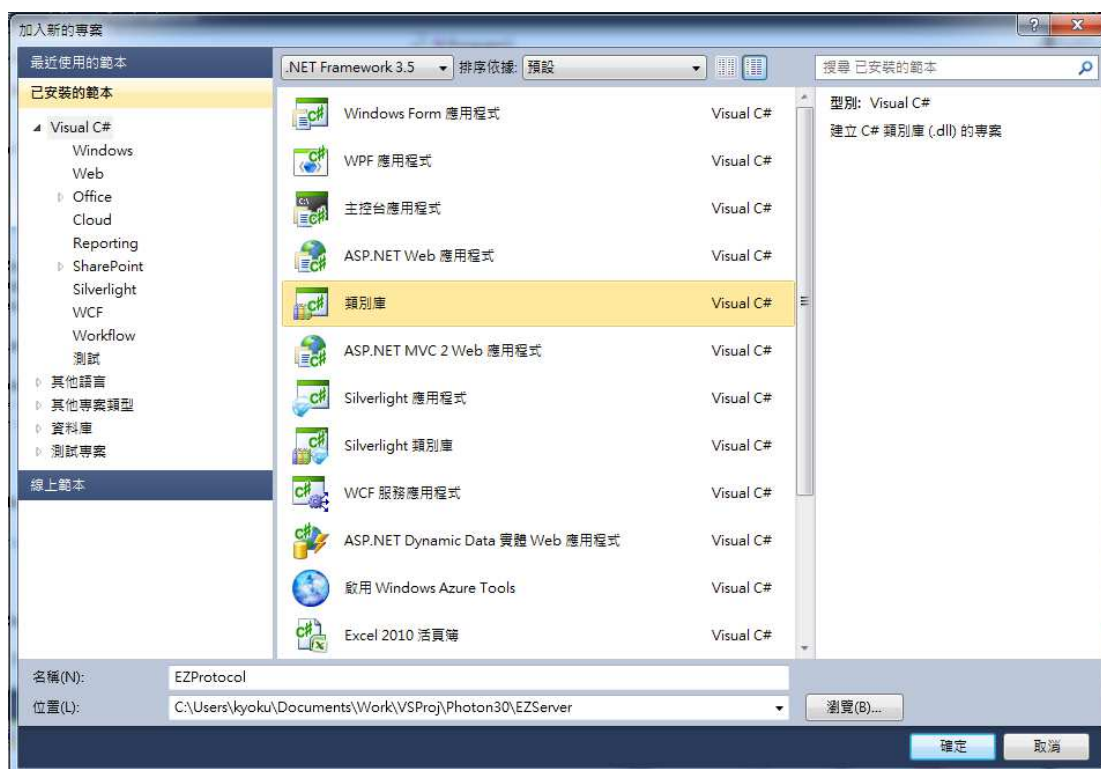


## 五 將命令代碼變成看得懂的東西吧

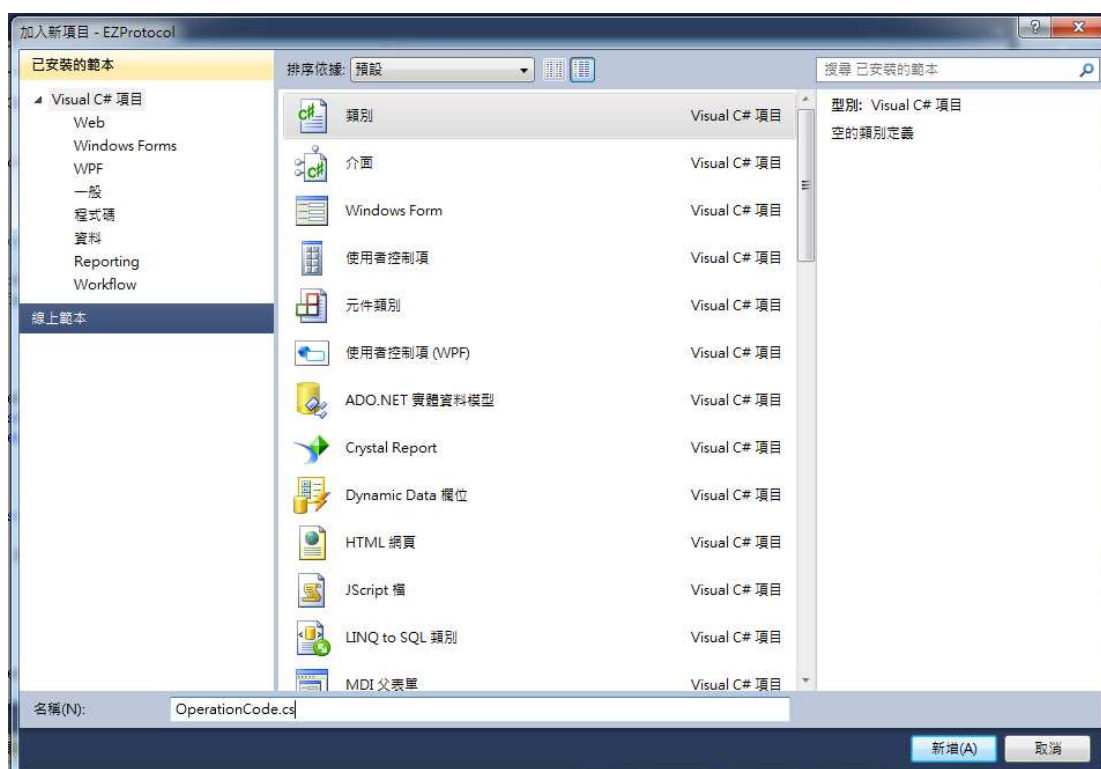
到目前為止我們的命令代碼都是一個數字，不但不易閱讀而且萬一打錯數字要找出錯誤也非常困難，所以我們要將命令代碼轉換為統一的傳輸協議，讓 Server 和 Client 可以用指令的方式去閱讀這些代碼。

### 1. 建立列舉命令類別

回到 Visual Studio，在方案下新增一個類別專案「EZProtocol」。



把原來的 Class1.cs 刪除，加入一個新的類別「OperationCode」。



OperationCode 類別一開始的內容如下。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EZProtocol
{
    class OperationCode
    {
    }
}
```

我們把他改成列舉，上面的 **using** 可以全部砍掉，在這個命令列舉裡不會用這些。

```
namespace EZProtocol
{
    public enum OperationCode
    {
    }
}
```

最後加入列舉的內容

```
namespace EZProtocol
{
    public enum OperationCode
    {
        Login=5,
    }
}
```

```
}
```

用相同的方法再建立一個 **ErrorCode.cs** 的類別

```
namespace EZProtocol
{
    public enum ErrorCode
    {
        Ok=0,
        InvalidOperation,
        InvalidParameter
    }
}
```

接下來再建立一個讀取參數用的列舉，檔名為 **ParameterCode.cs**，裡面放入兩個列舉類別「**LoginParameterCode**」和「**LoginResponseCode**」。

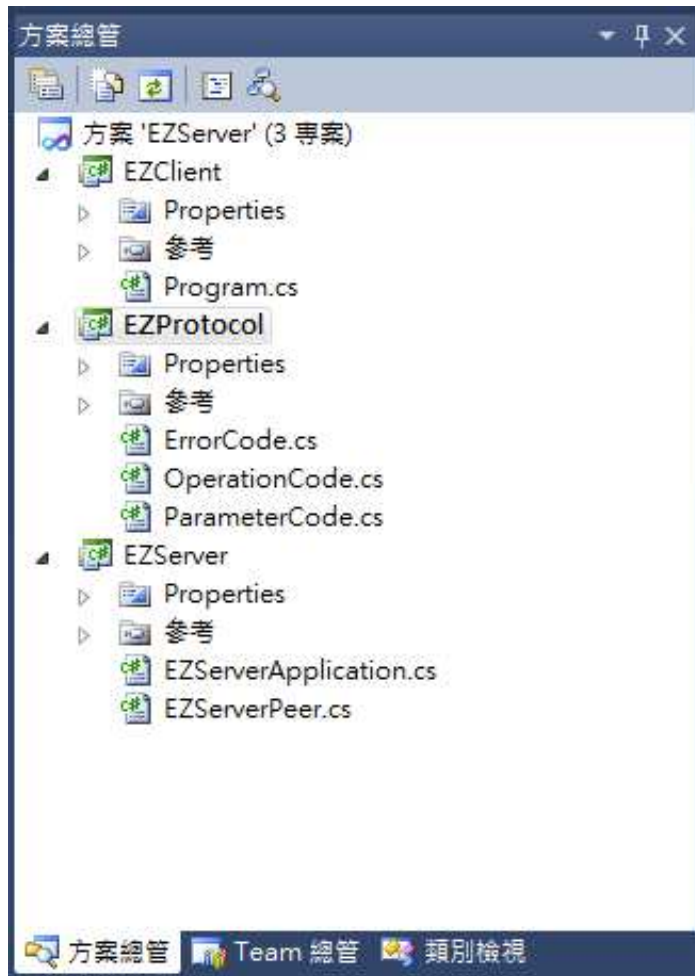
```
namespace EZProtocol
{
    public enum LoginParameterCode
    {
        MemberID = 1,
        MemberPW,
    }

    public enum LoginResponseCode
    {
        MemberID = 1,
        MemberPW,
        Nickname,
    }
}
```

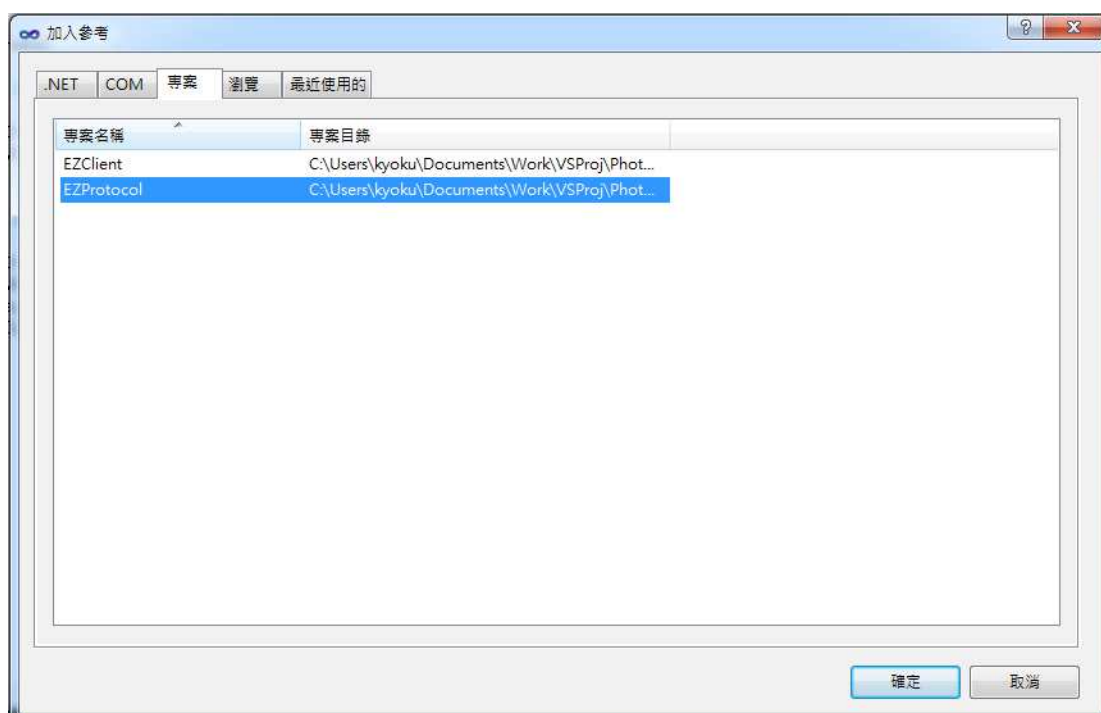
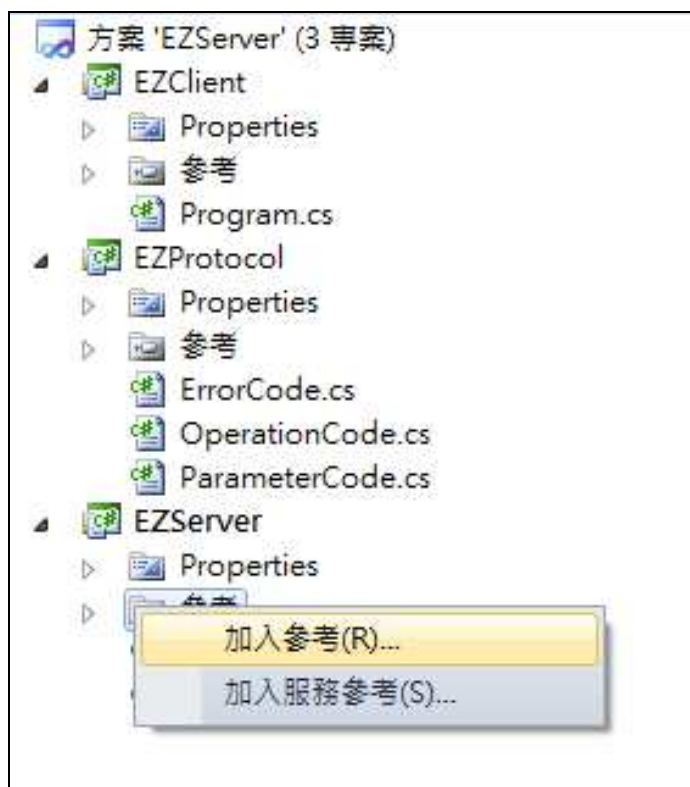


```
        Ret = 80,  
    }  
}
```

到此會看到專案的結構如下



將 EZServer 和 EZClient 都加入 EZProtocol 的參考。



## 2. 將 Server 端的命令碼換成列舉的類別

打開 EZServerPeer.cs 檔，將原本寫死的那幾個數字換成列舉的類別。

```
.....

using EZProtocol;

namespace EZServer
{
    public class EZServerPeer : PeerBase
    {
        .....

        protected override void OnOperationRequest(OperationRequest operationRequest,
        SendParameters sendParameters)
        {
            // //取得Client端傳過來的要求並加以處理

            switch (operationRequest.OperationCode)
            {
                case (byte)OperationCode.Login:
                {
                    if (operationRequest.Parameters.Count < 2) // 若參數小於2則返回錯誤
                    {
                        // 返回登入錯誤

                        OperationResponse response = new
                        OperationResponse(operationRequest.OperationCode) { ReturnCode = (short)ErrorCode.InvalidParameter,
                        DebugMessage = "Login Fail" };

                        SendOperationResponse(response, new SendParameters());
                    }
                }
            }
        }
    }
}
```

```

else
{
    var memberID =
(string)operationRequest.Parameters[(byte>LoginParameterCode.MemberID];
    var memberPW =
(string)operationRequest.Parameters[(byte>LoginParameterCode.MemberPW];

    if (memberID == "test" && memberPW == "1234")
    {
        int Ret = 1;
        var parameter = new Dictionary<byte, object> {
            { (byte>LoginResponseCode.Ret, Ret },
            {(byte>LoginResponseCode.MemberID, memberID}, {(byte>LoginResponseCode.MemberPW,
memberPW}, {(byte>LoginResponseCode.Nickname, "Kyoku"}
        };

        OperationResponse response = new
OperationResponse(operationRequest.OperationCode, parameter) { ReturnCode = (short>ErrorCode.Ok,
DebugMessage = "" };

        SendOperationResponse(response, new SendParameters());

    }
    else
    {
        OperationResponse response = new
OperationResponse(operationRequest.OperationCode) { ReturnCode = (short>ErrorCode.InvalidOperation,
DebugMessage = "Wrong id or password" };

        SendOperationResponse(response, new SendParameters());

    }

}

break;

```

```

        }
    }
}
}
}

```

### 3. 將 Client 端的命令換成列舉類別

開啟 client 端的 Program.cs，參考 Server 的做法將那些寫死的數字換成易閱讀的列舉類別

```

.....

using EZProtocol;

namespace EZClient
{
    class Program : IPhotonPeerListener
    {
        .....

        public void Run()
        {
            if (peer.Connect("localhost:5055", "EZServer")) //若成功連到Server
            {

```

```

do
{
    peer.Service();           // 在主迴圈取得Server傳過來的命令，當有傳命令過來時依命令的類型，EventAction或OperationResult會被觸發

    if (ServerConnected && !OnLogin)
    {
        Console.WriteLine("\n請輸入會員帳號：");
        string memberID = Console.ReadLine();

        Console.WriteLine("\n請輸入會員密碼：");
        string memberPW = Console.ReadLine();

        var parameter = new Dictionary<byte, object> {
            { (byte)LoginParameterCode.MemberID, memberID.Trim() }, { (byte)LoginParameterCode.MemberPW, memberPW.Trim() } // parameter key memberID=1, memberPW=2
        };

        this.peer.OpCustom((byte)OperationCode.Login, parameter, true); //命令
    }

    OnLogin = true;

    System.Threading.Thread.Sleep(500); // 休息.5秒以免鎖死電腦

}
while (true);
}
else
{
    Console.WriteLine("Unknown hostname!");
}

```

代碼為5

```

    }
}

.....

public void OnOperationResponse(OperationResponse operationResponse)
{
    // 取得Server傳過來的命令回傳

    switch (operationResponse.OperationCode)
    {
        case (byte)OperationCode.Login: // 在此範例是訂5為Login要求
        {
            switch (operationResponse.ReturnCode)
            {
                case (short)ErrorCode.Ok: // ReturnCode0代表成功
                {
                    int Ret =
Convert.ToInt16(operationResponse.Parameters[(byte>LoginResponseCode.Ret]);
                    string memberID =
Convert.ToString(operationResponse.Parameters[(byte>LoginResponseCode.MemberID]);
                    string memberPW =
Convert.ToString(operationResponse.Parameters[(byte>LoginResponseCode.MemberPW]);
                    string Nickname =
Convert.ToString(operationResponse.Parameters[(byte>LoginResponseCode.Nickname]);

                    Console.WriteLine("Login Success \nRet={0}
\nMemberID={1} \nMemberPW={2} \nNickname={3} \n", Ret, memberID, memberPW, Nickname);

                    break;
                }
            }
        case (short)ErrorCode.InvalidOperation: // 帳號或密碼錯誤
        {
            Console.WriteLine(operationResponse.DebugMessage);

```

```

        break;
    }
    case (short)ErrorCode.InvalidParameter: // 傳入的參數錯誤
    {
        Console.WriteLine(operationResponse.DebugMessage);
        break;
    }
    default:
    {
        Console.WriteLine(String.Format("不明的ReturnCode :
{0}", operationResponse.ReturnCode));

        break;
    }
}
break;
}
default:
{
    Console.WriteLine(String.Format("不明的OperationCode : {0}",
operationResponse.OperationCode));

    break;
}
}
}

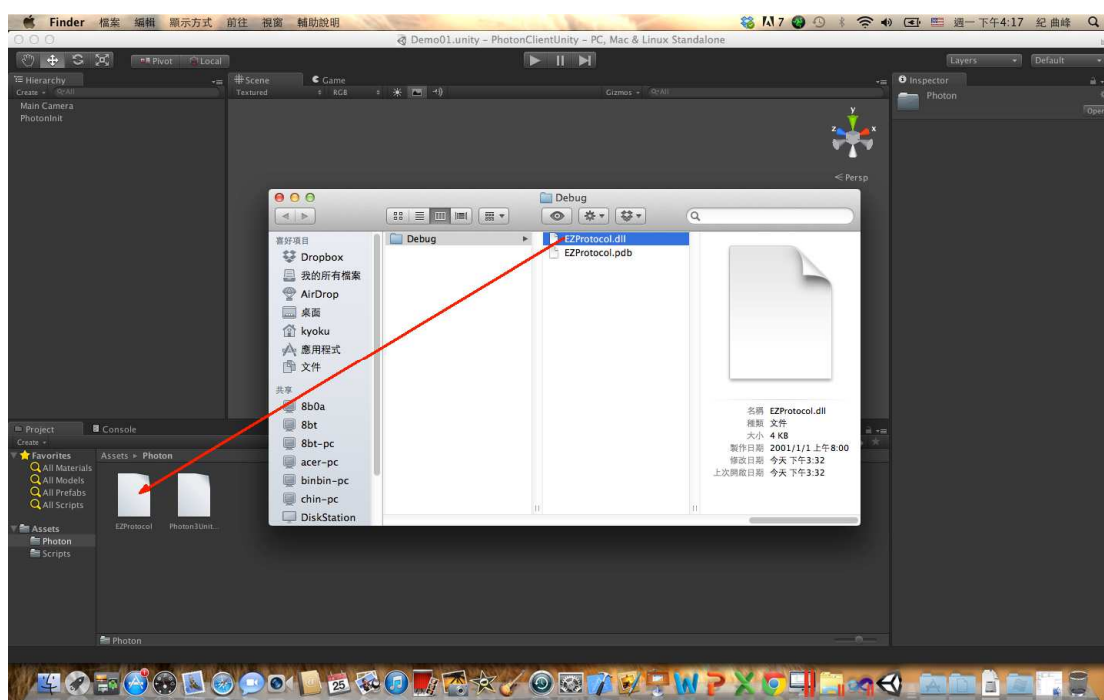
```

.....



## 4. 將 Unity 的命令代碼換成列舉類別

要讓 Unity 認得我們的列舉類別成員我們必需把列舉類別的 dll 加到 Unity 裡面，若 Photon 專案內沒有這個 dll，先選取專案然後在讓 VS 重新編譯專案便可產生此檔，有了 dll 後請開啟 Unity 專案，將 EZProtocol 編譯好的 dll 直接拉到 Unity 專案內。



之後修改 Unity 的 PhotonClient.cs 程式碼

```

.....
using EZProtocol;

public class PhotonClient : MonoBehaviour, IPhotonPeerListener {

.....

    public void OnOperationResponse (OperationResponse operationResponse)
    {

        // display operationCode

```

```

        this.DebugReturn(0, string.Format("OperationResult:"
operationResponse.OperationCode.ToString()));

        switch (operationResponse.OperationCode)
        {
            case (byte)OperationCode.Login:
            {
                if( operationResponse.ReturnCode == (short)ErrorCode.Ok) // if success
                {
                    getRet
                    Convert.ToInt32(operationResponse.Parameters[(byte>LoginResponseCode.Ret]);
                    getMemberID
                    Convert.ToString(operationResponse.Parameters[(byte>LoginResponseCode.MemberID]);
                    getMemberPW=
                    Convert.ToString(operationResponse.Parameters[(byte>LoginResponseCode.MemberPW]);
                    getNickname=
                    Convert.ToString(operationResponse.Parameters[(byte>LoginResponseCode.Nickname]);
                    LoginStatus = true;

                }
                else
                {
                    LoginResult = operationResponse.DebugMessage;
                    LoginStatus = false;
                }
                break;
            }
        }
    }
}
.....

```

```

void OnGUI()
{
    GUI.Label(new Rect(30,10,400, 20), "EZServer Unity3D Test");

    if( this.ServerConnected )
    {
        GUI.Label(new Rect(30,30,400, 20), "Connected");

        GUI.Label(new Rect(30,60,80, 20), "MemberID:");
        memberID = GUI.TextField(new Rect(110, 60, 100, 20), memberID, 10);

        GUI.Label(new Rect(30,90,80, 20), "MemberPW:");
        memberPW = GUI.TextField(new Rect(110, 90, 100, 20), memberPW, 10);

        if( GUI.Button( new Rect(30,120,100,24 ), "Login" ) )
        {
            var parameter = new Dictionary<byte, object> {
                { (byte>LoginParameterCode.MemberID, memberID },
                { (byte>LoginParameterCode.MemberPW, memberPW } // parameter key memberID=1, memberPW=2
            };

            this.peer.OpCustom((byte)OperationCode.Login, parameter, true); // operationCode
is 5

        }

        if( LoginStatus )
        {
            GUI.Label(new Rect(30,150,400, 20), "Your MemberID : " + getMemberID);
            GUI.Label(new Rect(30,170,400, 20), "Your Password : " + getMemberPW);
            GUI.Label(new Rect(30,190,400, 20), "Your Nickname : " + getNickname);
            GUI.Label(new Rect(30,210,400, 20), "Ret : " + getRet.ToString());
        }
        else
    
```

```
        {  
            GUI.Label(new Rect(30,150,400, 20), "Please Login");  
            GUI.Label(new Rect(30,170,400, 20), LoginResult);  
        }  
  
    }  
    else  
    {  
        GUI.Label(new Rect(30,30,400, 20), "Disconnect");  
    }  
}  
  
}
```

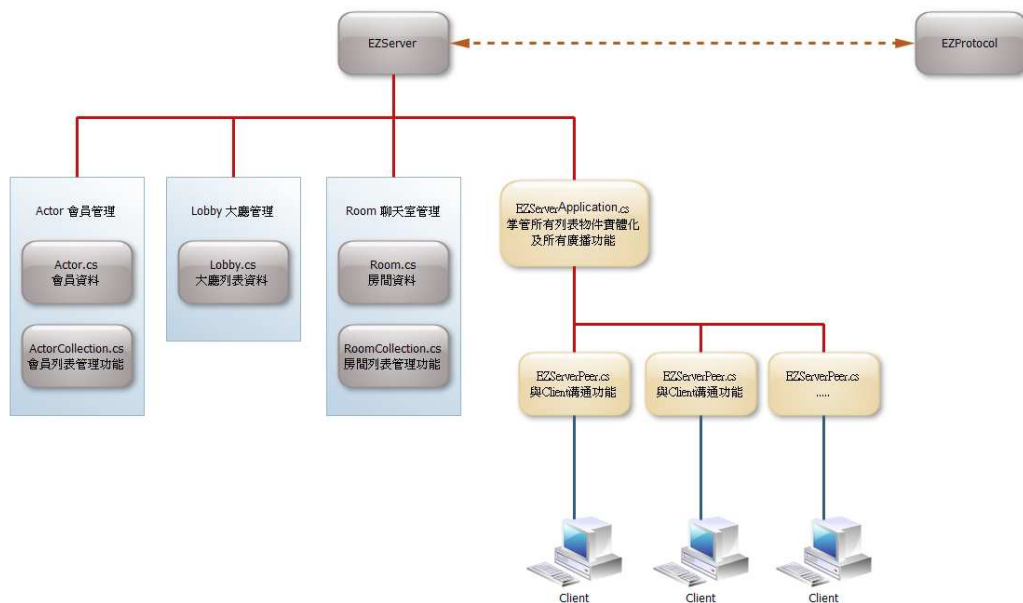
## 六 柚子星球多人聊天室原始碼說明

本教學提供了柚子星球線上聊天室原始碼，可從下列網址下載：

<http://www.digiart.com.tw/EZServerSrc.zip>

下載後您可自行修改內容以符合您的需求，以下對這個 **Server** 的架構內容做一個簡單的說明以方便理解其內容。

## 1. Server 架構圖



## 2. 廣播的原理

建立 Server 的廣播系統必須建立一個獨立的執行緒，再由執行緒去主動發送訊號，執行緒可以看成是程式中的程式，在 .net framework 中建立執行緒有幾種方法，較傳統的方法是自己建立執行緒然後自己管理所有執行緒的列隊，不過 .net 後來提供了 **ThreadPool** 來管理執行緒集合，使用起來非常方便，指令也很簡單，現在來看一下如何使用 **ThreadPool**。

```

.....
using System.Threading;

namespace EZServer
{
    public class EZServerApplication : ApplicationBase
    {
        .....

        private bool broadcastActive = false;

        .....

        protected override void Setup()
        {
            .....

            // create broadcast threading
            broadcastActive = true;
            ThreadPool.QueueUserWorkItem(this.LobbyBroadcast);
        }
    }
}

.....

protected override void TearDown()
{
    // 關閉GameServer並釋放資源
    broadcastActive = false;
}

private void LobbyBroadcast(object state)
{
    while (broadcastActive)

```

```

    {
        Thread.Sleep(500);
    }
}

```

利用 ThreadPool 建立執行緒只須一行指令

ThreadPool.QueueUserWorkItem( 要執行的方法 );

然後我們建立一個執行緒的方法，取名為「LobbyBroadcast(object state)」，傳入值的格式是固定的，我們不須更動，接著我們將這個名稱填入到 ThreadPool，於是變成「ThreadPool.QueueUserWorkItem( this.LobbyBroadcast );」，這樣執行緒便建立完成了。但是，這樣建立的執行緒只會執行一次，因為執行緒就是程式裡面的程式，所以可以把他當作是一支獨立的程式來看，一般的程式都是執行到尾就結束了，若想要讓此程式永遠執行下去就要加一個 while( ) 迴圈，但一旦執行緒建立了 While 迴圈就必須在和式結束時將他停掉，否則這支獨立的程式將永遠佔住一個記憶區塊和 cpu 自顧自的執行下去，因此在 LobbyBroadcast 裡面 while 迴圈要判斷若程式結速就停掉迴圈，如「while (broadcastActive) { ... }」，當程式要結束時將 broadcastActive 設為 false 迴圈就自然停掉了。

最後還一點要注意的，每個迴圈都要讓執行緒休息一下，不然這個迴圈將會佔住 100% 的 cpu，因此筆者直接在這個框架裡加上「Thread.Sleep(500);」讓 while 每個迴圈處理完就釋放 0.5 秒的時間，除了釋放掉迴圈一段時間讓 cpu 去做別的事之外，我們原本就不希望廣播的機制去影響到正常遊戲的訊號存取，因此才不採用 Client 不斷向 Server 索取列表的方式，而是讓 Server 固定一段時間才向 Client 分批發佈列表內容，這麼一來不管 Client 上線再多，Server 在廣播的動作都是固定的，所耗的資源也是固定的，但 Client 會犧牲掉一些即時性。

建立好執行緒後，從執行緒裡發出對 Client 的 Event 便可完成廣播的動作。

### 3. Client 的委派與事件

什麼是委派，簡單說就是「我希望能幫我做某某事」用的，可是這不就和一般的函式(或稱方法)一樣了嗎？在概念可以說是有點像，但程式碼看起來就差很多，我們先來看看如何宣告委派。

```
public delegate void ConnectEventHandler(bool ConnectStatus);
```

這就是一個標準的委派宣告，delegate 關鍵字是委派的宣告字元，然後後面的整段字串「void ConnectEventHandler(bool ConnectStatus)」都是委派的宣告內容，委派的宣告和一般的函式宣告不同，委派只宣告外觀，不直接宣告函式名稱。

宣告完之後我們便可以另外宣告實際的實作內容讓自己或別的物件來使用這個委派。

先實作物件內容，這個實作物件的外觀必需跟之前宣告委派時一模一樣，依此例就是傳入一個布林值，然後做某些事，最後不回傳任何值，實作方法的名稱可以自訂。

```
static public void myHandlerMethod( bool status)
{
    if( status)
    {
        // 可以把想做的事放在這裡
    }
    else
    {
    }
}
```



```
}
```

然後宣告物件，並指定實作的方法為myHandlerMethod。

```
public ConnectEventHandler myHandler = new ConnectEventHandler(myHandlerMethod);
```

最後我們就可以直接使用這個委派，使用的方法是呼叫最後宣告的物件，也就是 myHandler，一樣必須和宣告委派時一樣的外觀才行。

```
myHandler(false);
```

相信讀者看到現在一定覺得委派這個東西窮極無聊，若只是想做一個讓人呼叫的方法用一般的函式就行了，何必寫了這麼長一串只是為了一個公用函式，在一般的委派的确感受不到他有什麼作用，其實委派的價值可不止這樣，當委派遇到了事件 Event 之後一切都不同了，事件 Event 才是委派的終極應用方式，因為他可以幫我們解決各種非同步問題，也可以讓我們開發的元件更具使用彈性，委派之所以能夠和事件那麼契合主要就在於委派所擁有的特性。

### 委派的特性：

1. 任何與委派外觀相符的方法都可指派給委派
2. 委派允許將方法當作參數傳遞
3. 委派可用於定義回呼方法
4. 委派可鏈結在一起，使單一事件上呼叫多個方法

基於這些特性讓委派與事件變得非常好用，可直接觀看下載專案，在 Unity 專案裡面的 PhotonService.cs，最下面的部份可看到 Event 的實作。

.....

```
public void OnEvent (EventData eventData)
{
    throw new System.NotImplementedException ();
}
```

```
/**
 * 事件委派宣告
 */
```

```
// 連線事件的委派
```

```
public delegate void ConnectEventHandler(bool ConnectStatus);
public event ConnectEventHandler ConnectEvent;
```

```
}
```

我們先宣告了一個連線用的委派，使用前面已經有說明，宣告委派只宣告外觀，至於實際的名稱可以由使用這個委派的物件自行定義，接下來還有一個特性是委派可以當做傳入值與傳回值，所以來看看事件的宣告。

```
public event ConnectEventHandler ConnectEvent;
```

首先使用關鍵字「event」宣告一個事件，事件名稱是「ConnectEvent」，回傳值

是名為「ConnectEventHandler」的委派，這樣事件的宣告便完成，現在來加入事件的觸發回傳。

```

.....

public void Connect(string IP, int Port, string ServerName)
{
    try
    {
        string ServerAddress = IP + "." + Port.ToString();
        this.peer = new PhotonPeer(this, ConnectionProtocol.Udp);
        if( !this.peer.Connect(ServerAddress, ServerName) )
        {
            ConnectEvent(false);
        }
    }
    catch (Exception EX)
    {
        ConnectEvent(false);
        throw EX;
    }
}

.....

public void OnStatusChanged (StatusCode statusCode)
{
    switch (statusCode)
    {
        case StatusCode.Connect:
            this.ServerConnected = true;
            ConnectEvent(true);
            break;
        case StatusCode.Disconnect:

```

```

        this.peer = null;

        this.ServerConnected = false;

        ConnectEvent(false);

        break;
    }
}

.....

```

和前面說明的委派使用範例一樣，事件的外觀必須對齊委派所宣告一樣的外觀才行，所以使用時直接使用事件名稱並傳入一個布林值「ConnectEvent(Boolean)」。

之後要再實作一個方法來接收此事件，實作的方式是使用運算符「+=」將實作的事件加進去，類似下面的做法，有時常在寫 **winform** 的人應該早就對這種寫法很熟悉了，**winform** 所有的按鈕或一些像 **timer** 之類的，都使用這種方式在做事件觸發。

```

PhotonService PS = new PhotonService();
PS.ConnectEvent += doConnectEvent;    // 加入事件

.....

// 事件觸發內容實作
private void doConnectEvent( bool status )
{
    if( status )
    {
    }
    else
    {
    }
}

```

```
}  
}
```

透過事件的用法，我們可以將 **Photon** 與遊戲邏輯程式碼進行完美的切割，然後不斷的擴充其功能便能完成一個大型的專案。

## 七 結語

Online Game 自從出現在市場之後，隨即成為了遊戲界的主流，現在，就算單機遊戲也都會提供或多或少的連線及雲端功能，因此在連線的技術已是開發遊戲者不能忽視的技術之一，幸而目前不管是 **Server** 或是 **Client** 都有各種套裝引擎可供選擇，因為遊戲引擎的出現及套裝 **Server** 的出現，遊戲及 **Server** 的開發門檻降低了很多，也讓許多的小型獨立團隊都得以開發出很受歡迎的社群遊戲，即便如此，要完成一個 OLG 絕不是單一個人有辦法獨立完成的，程式、企畫、美術缺一不可，為了讓一個 **Team** 能順利完成工作，除了基本技術的訓練外，遊戲內容的規畫及工作分配等等往往比技術來得重要，大多開發失敗的遊戲其主要原因都不是技術，在開發遊戲之前好好思考如何執行開發才是完成遊戲最重要的工作。

2013/5/5

紀曲峰