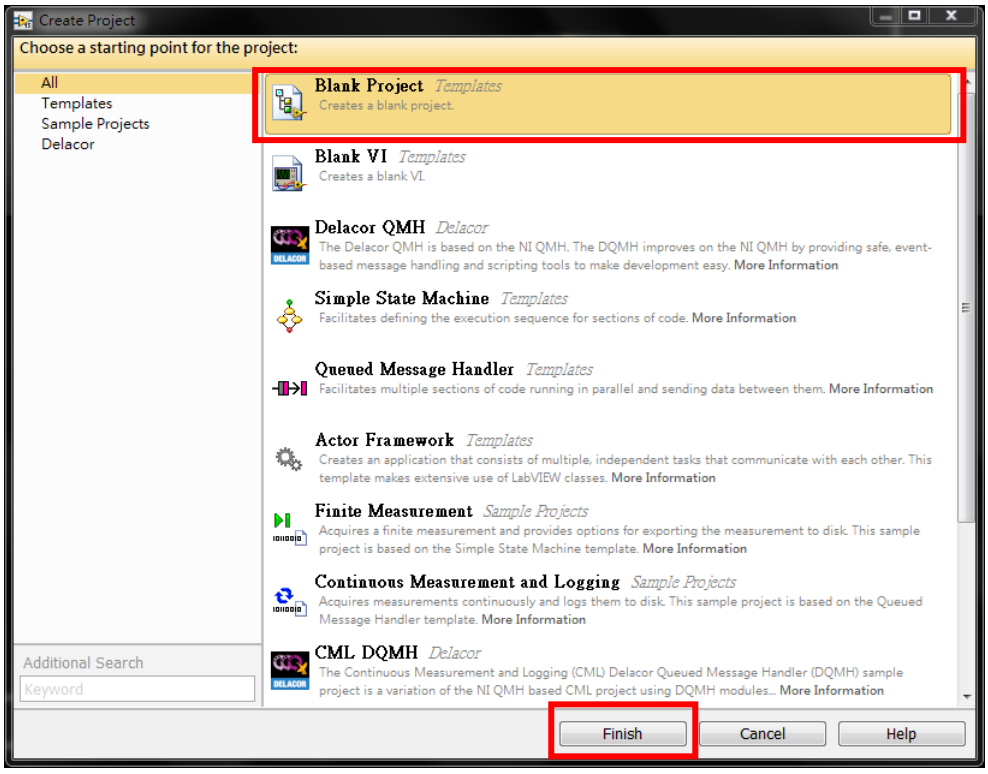
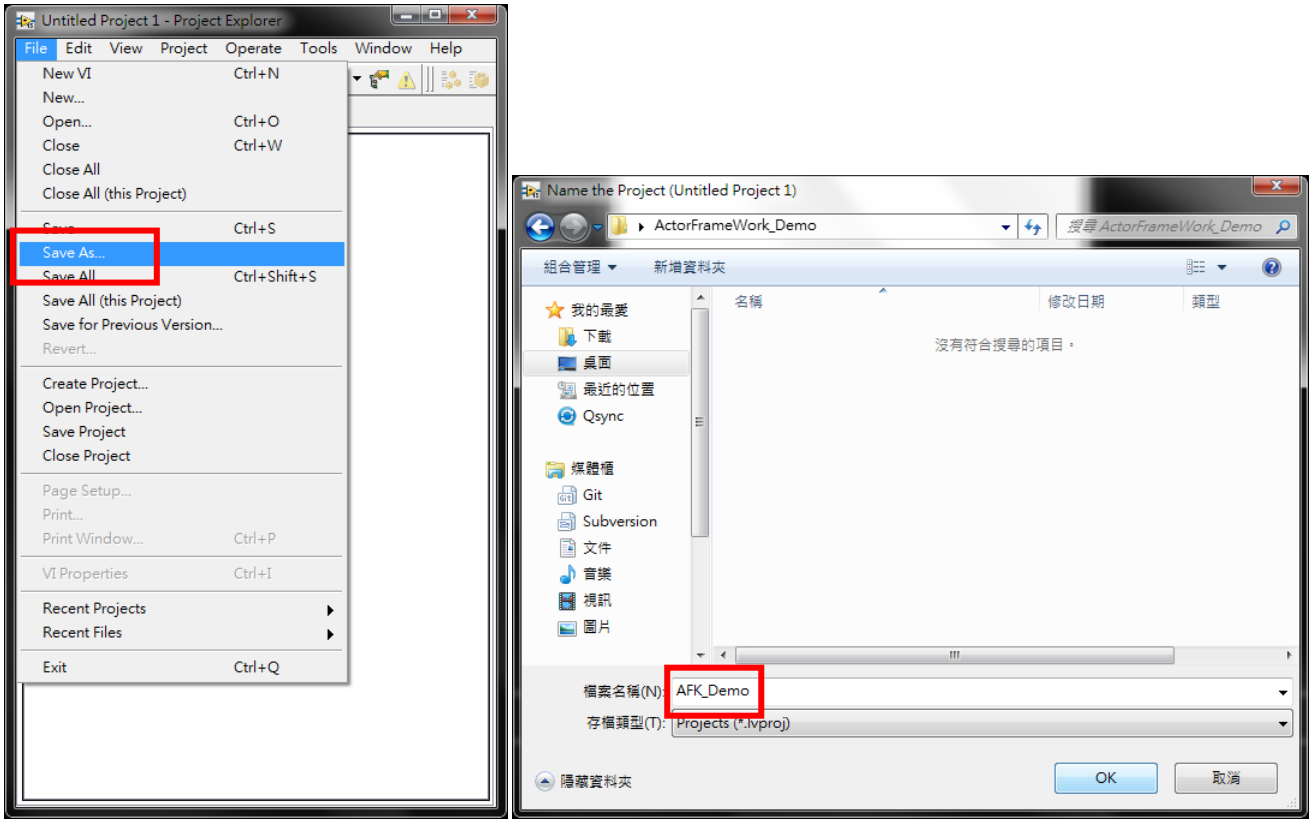
目的：建立一個能夠提供使用者操作的介面。

首先建立空白專案，用以設計 Actor Frame Work 專案。

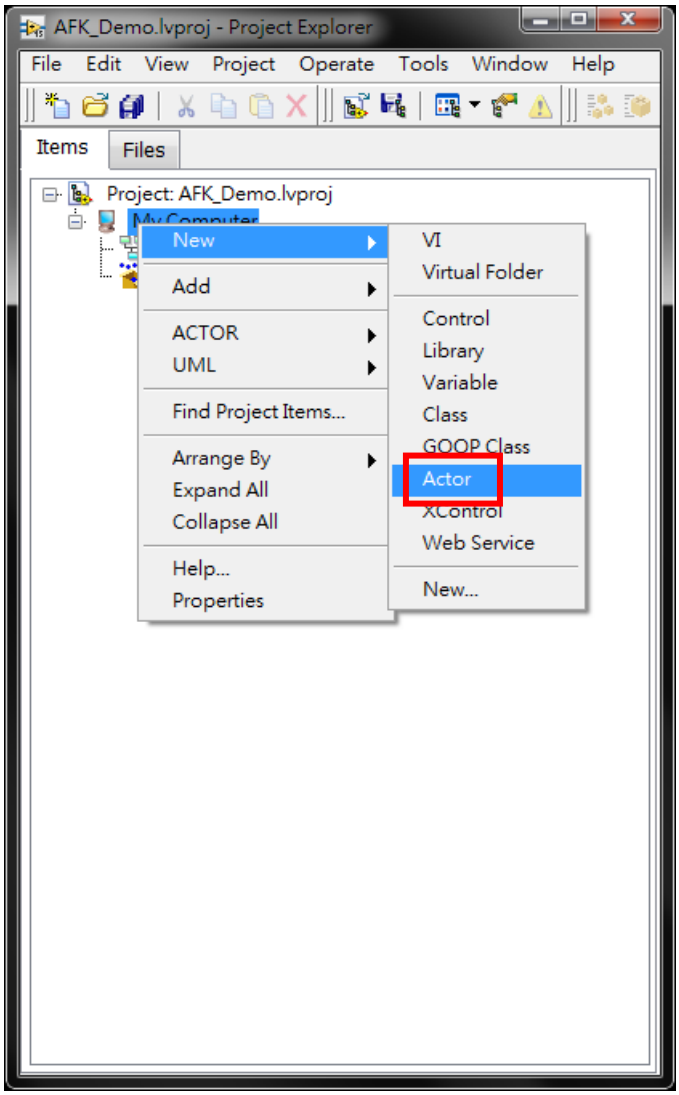


圖：建立空白專案。



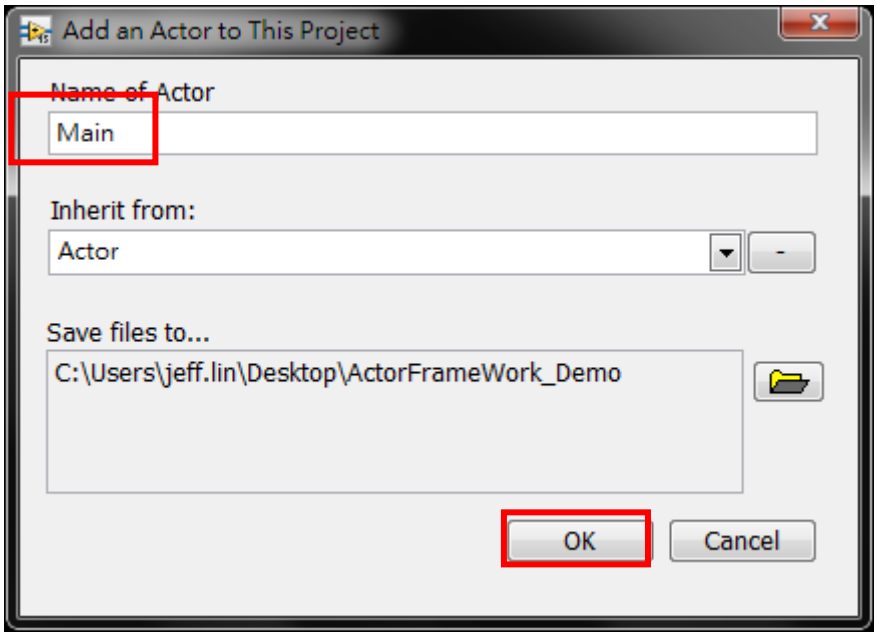
圖：儲存並命名此專案。

新增 Actor，同時命名為 Main，做為主要的(Root) Actor。



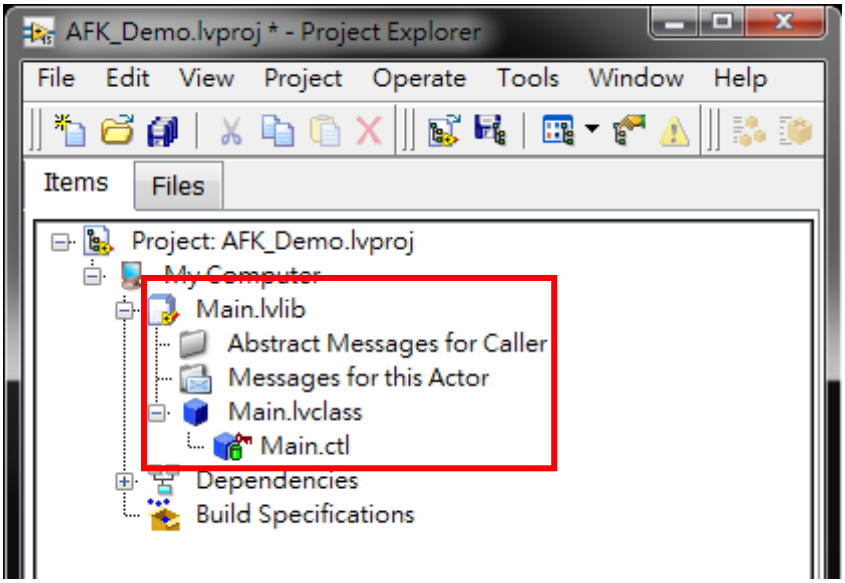
圖：新增 Actor。

(My Computer 右鍵→New→Actor)



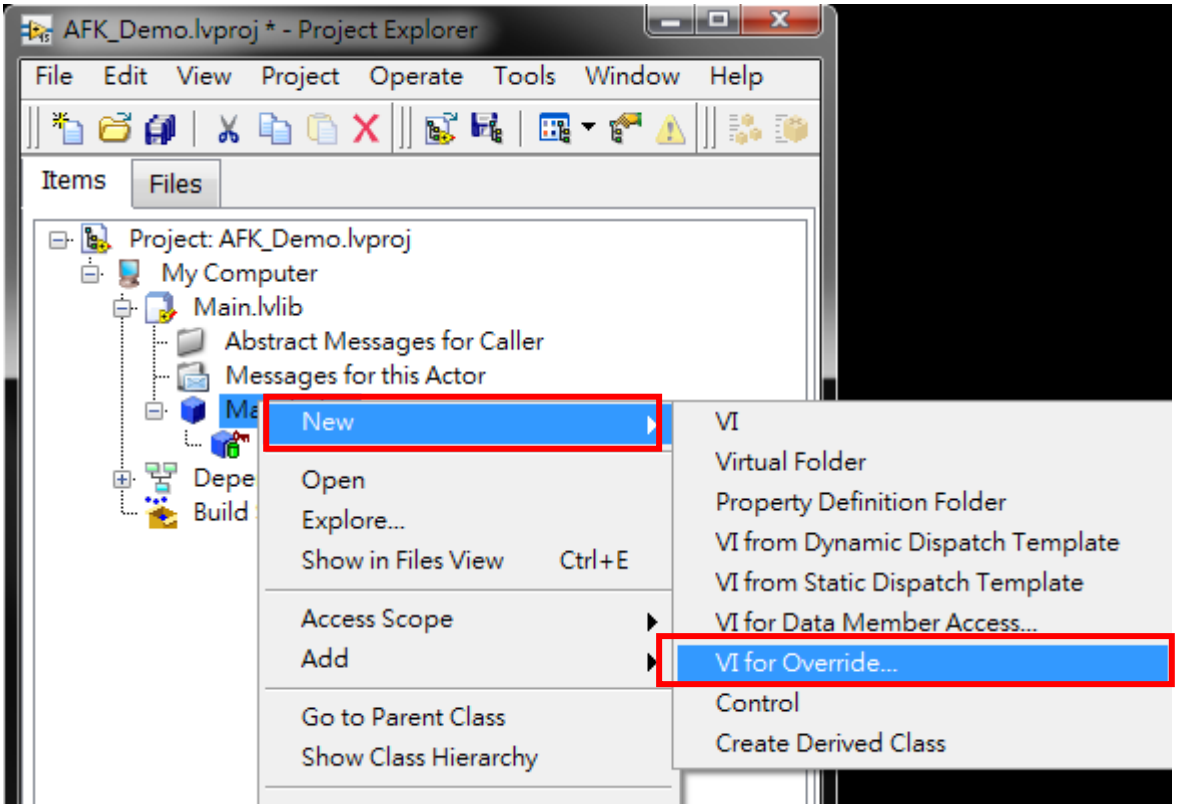
圖：將此 Actor 命名為 Main。

Main Actor 已經加入專案中，Main.lvlib 內部則是該 Actor 的相關檔案。



圖：Main Actor 已加入專案

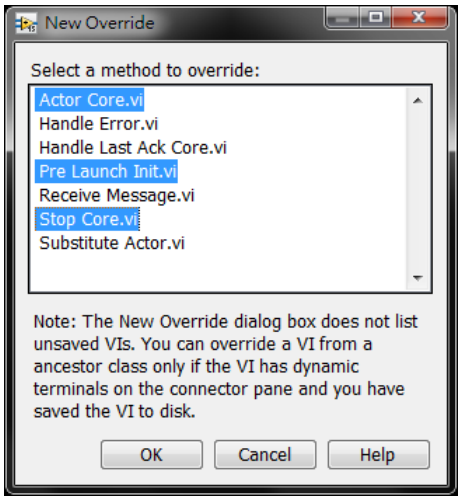
Actor 本身已經有設計一系列的 Method，為了擴充功能要 Override 原本的 Method。



圖：新增需要 Override 的 Method。

(Main.lvclass 上右鍵→New→VI for Override…)

因為需要一個可供使用者操作的介面，所以必須 Override Actor Core，並加入一個可接受 Stop 訊號的 User Event，其訊號分別由 Pre Launch Init 建立，由 Stop Core 釋放。



圖：選取需要改寫的 Method(Pre Launch Init.vi、Actor Core.vi、Stop Core.vi)。

Actor Core：運行核心。

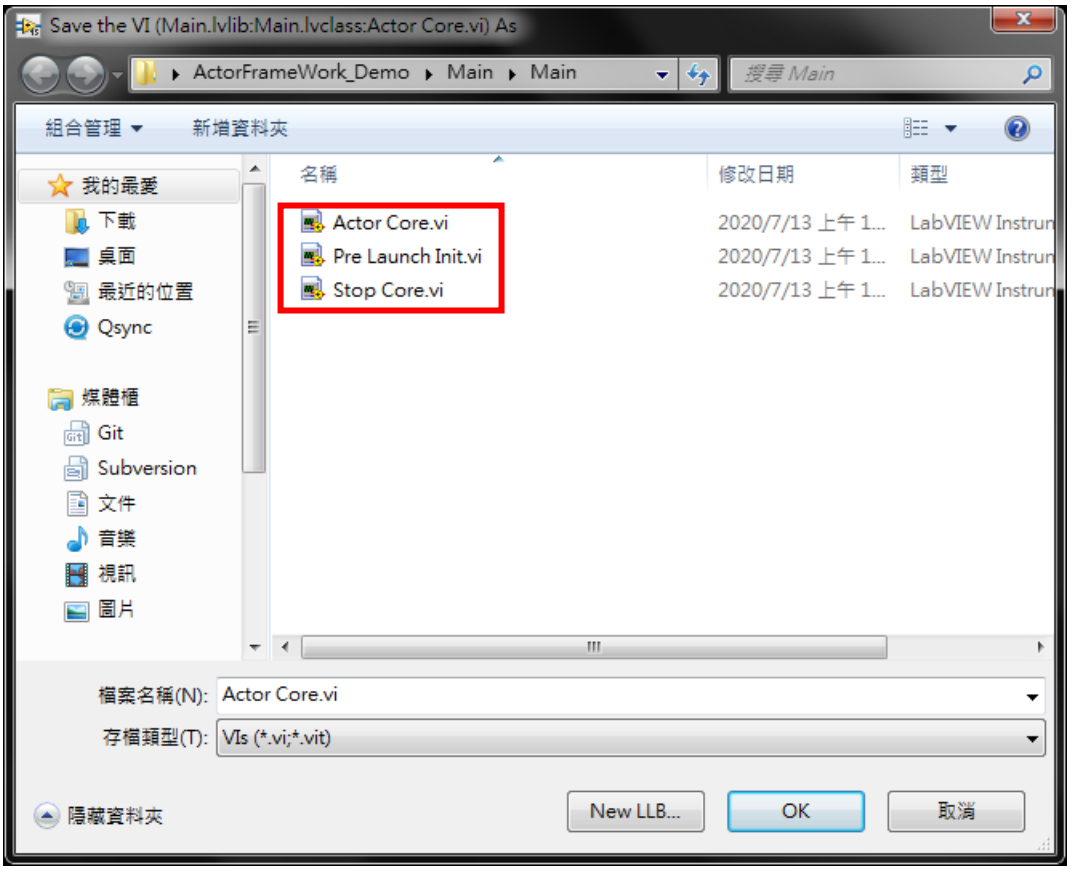
Handle Error：當 Actor Core 發生 Error 的處理方式。

Handle Last Ack Core：當 Actor Core 完成後，會做的最後一個動作。

Pre Launch Init：Actor Core 啟動前的初始化動作。

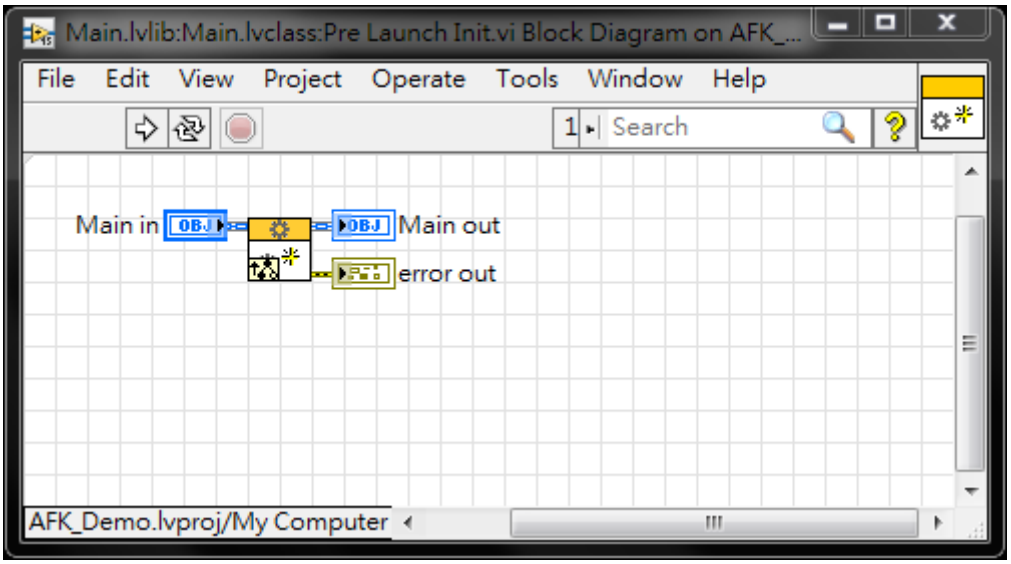
Receive Message：當 Actor 接收到 Message 時的動作。

Substitute Actor：替代 Actor 內部資料

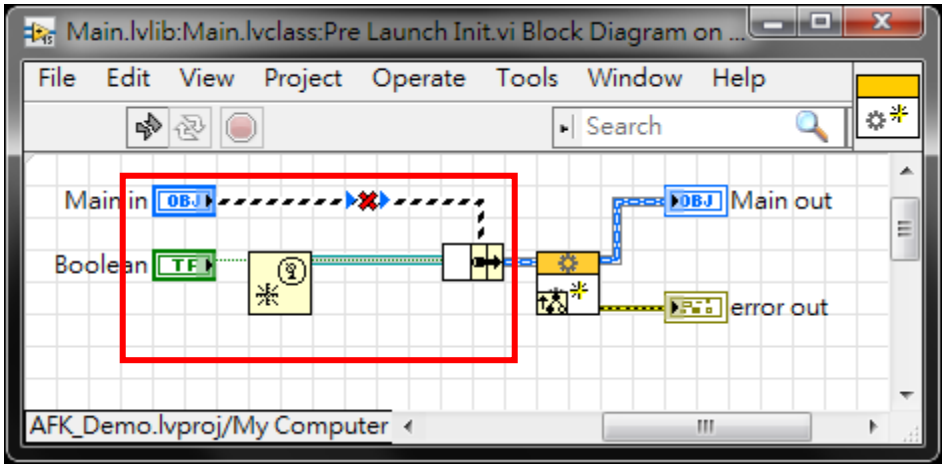


圖、儲存 Override 的 VI

首先修改 Pre Launch Init.vi，加入 User Event，供使用者發送 Stop 訊號



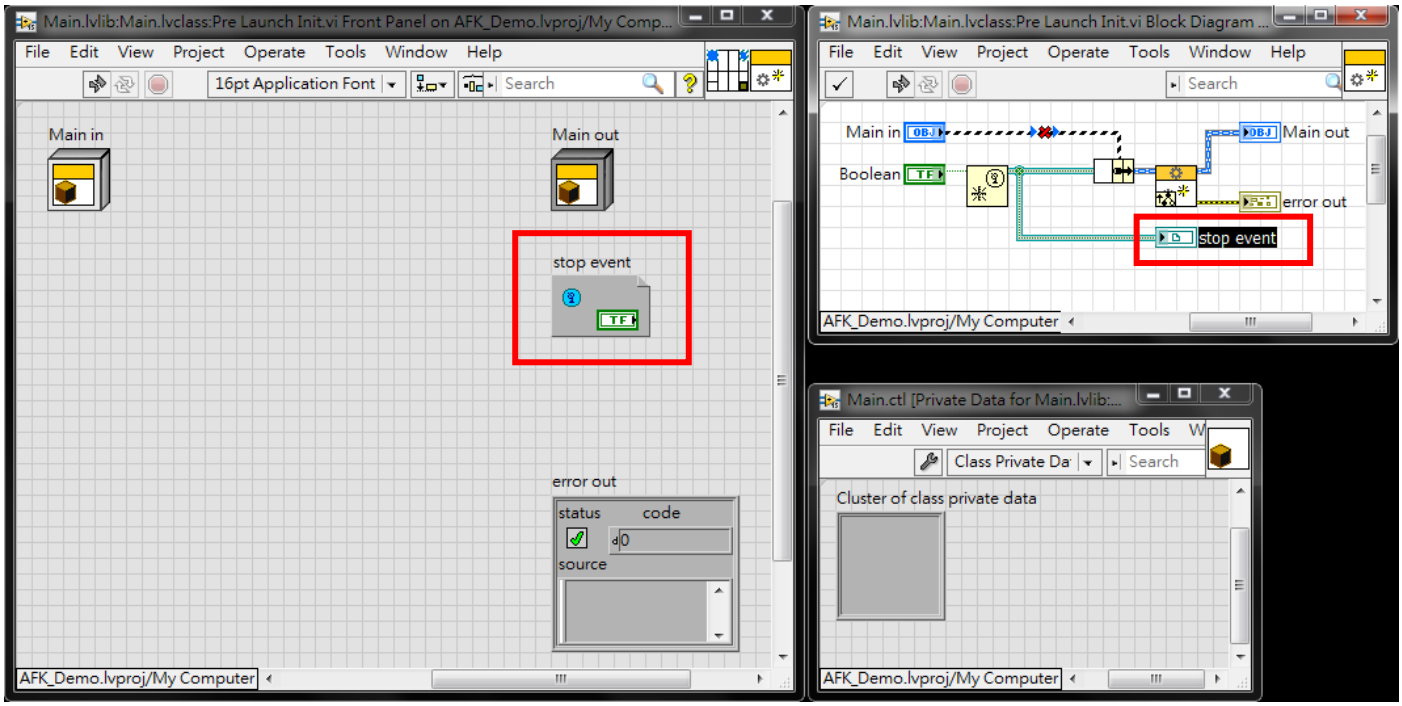
圖：改寫 Pre Launch Init.vi(原始)。



圖：改寫 Pre Launch Init.vi(修改後)。

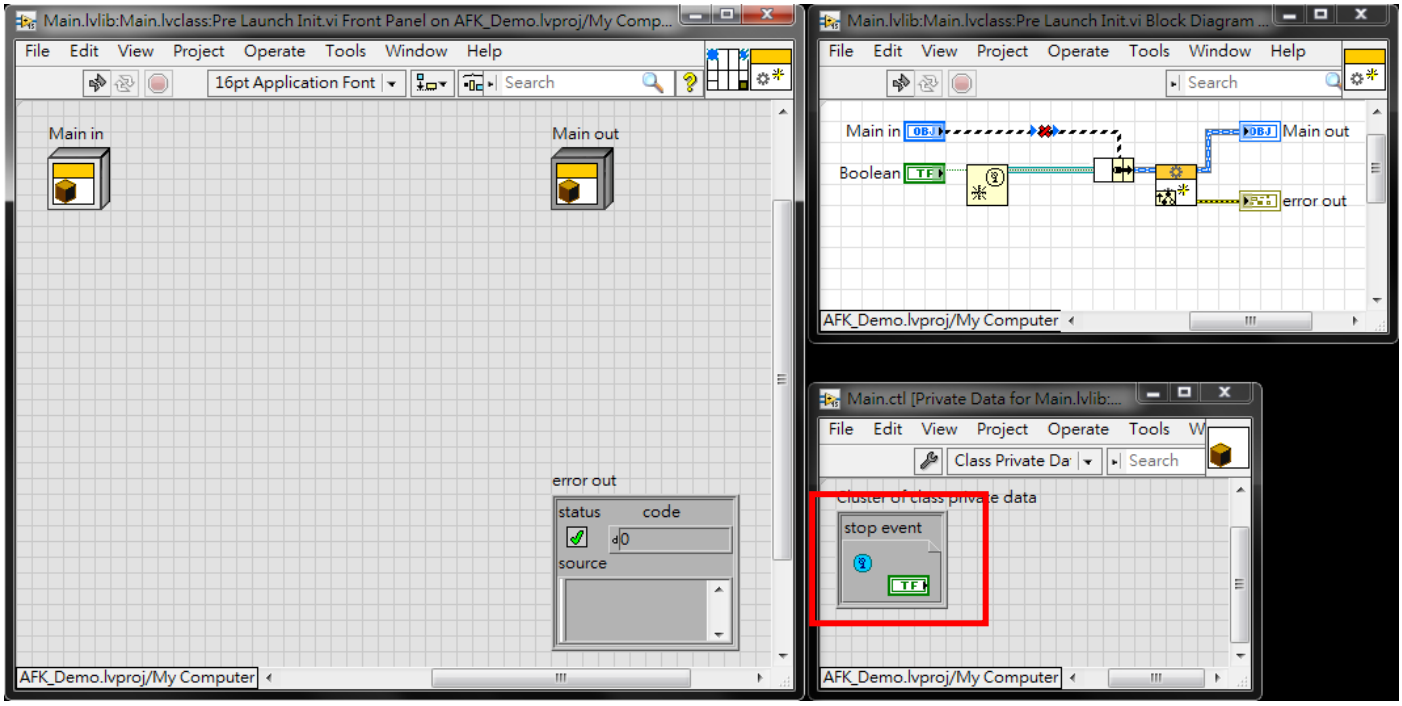
因為 Main.ctl 還沒加入此 User Event 所以目前還是斷線，

首先手動創造 Boolean User Event 的 Indicator 並更名為 Stop event。



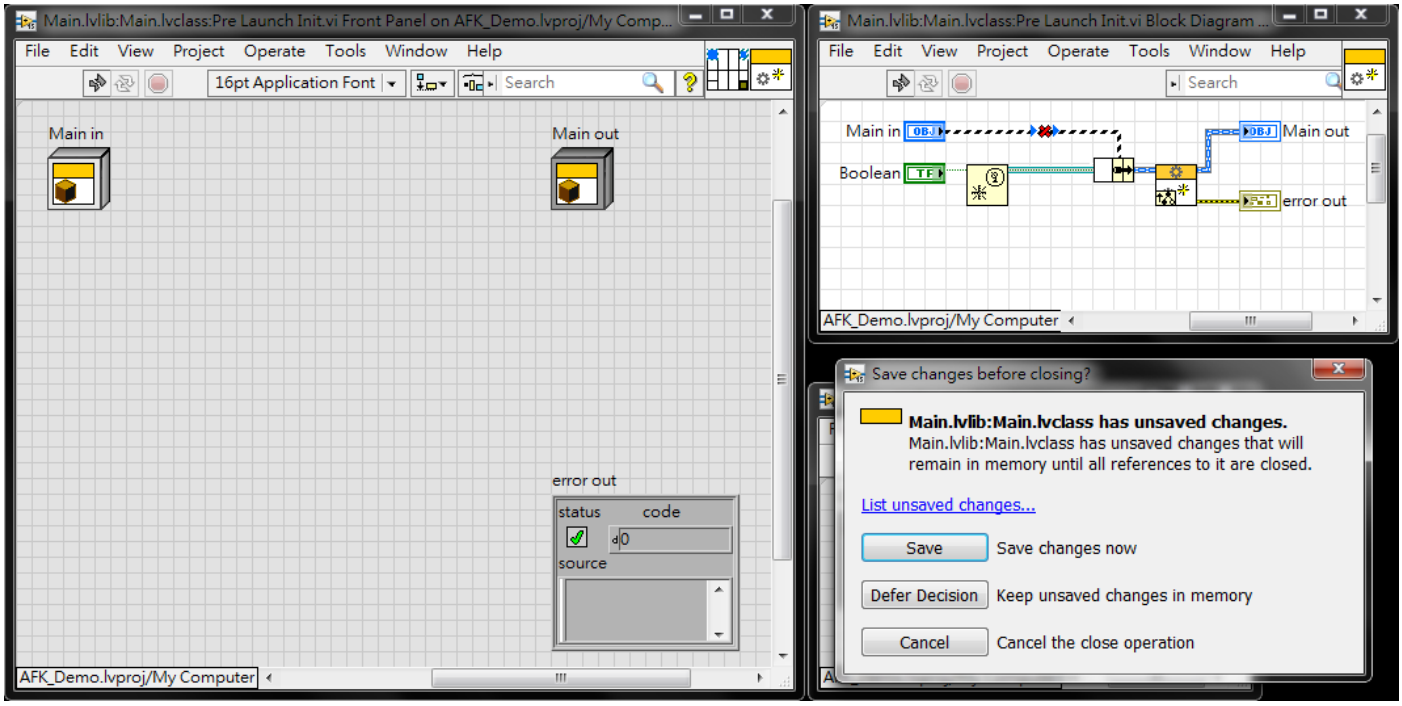
圖：手動創造 Stop Event 的 Indicator

將 Stop Event 的 Indicator，拉進 Main.ctl。



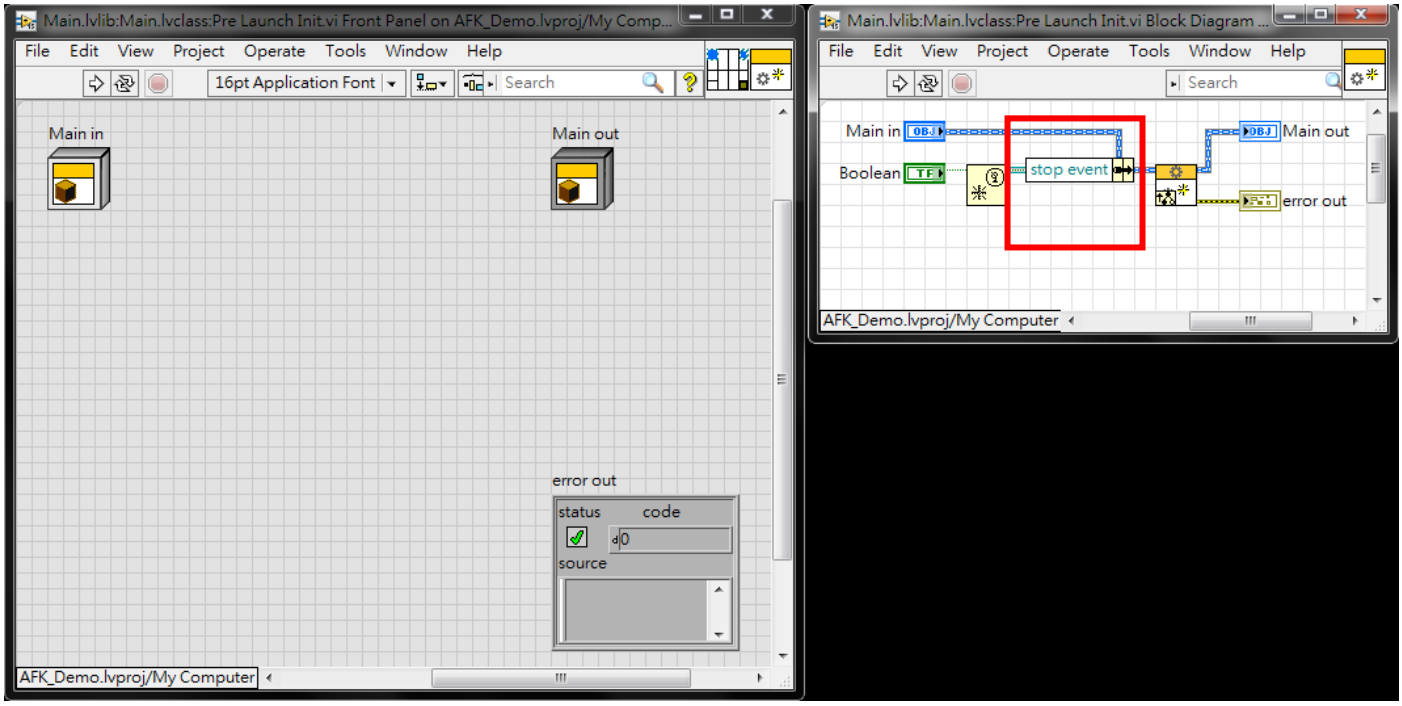
圖：將 Stop Event 加入 Main.ctl

修改完後儲存 Main.ctl。



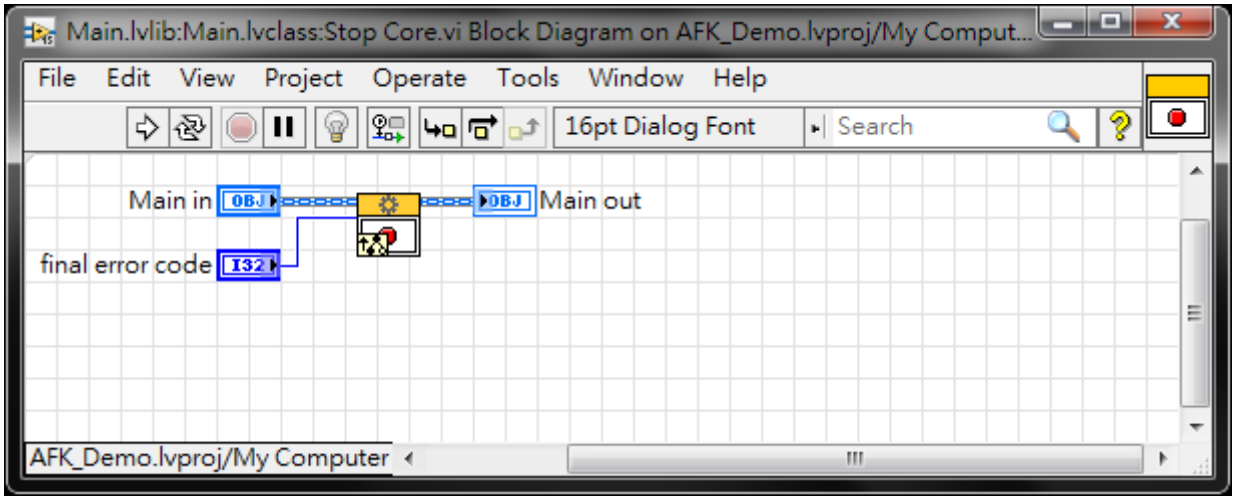
圖：儲存 Main.ctl

完成 Main.ctl 的修改後，可以發現因為 Step Event 已經加入 Actor，
設定 Stop Events 之後 Pre Launch Init.vi 已經沒有斷線了，此 Actor 便有了可以監控 Stop 訊號的機制。

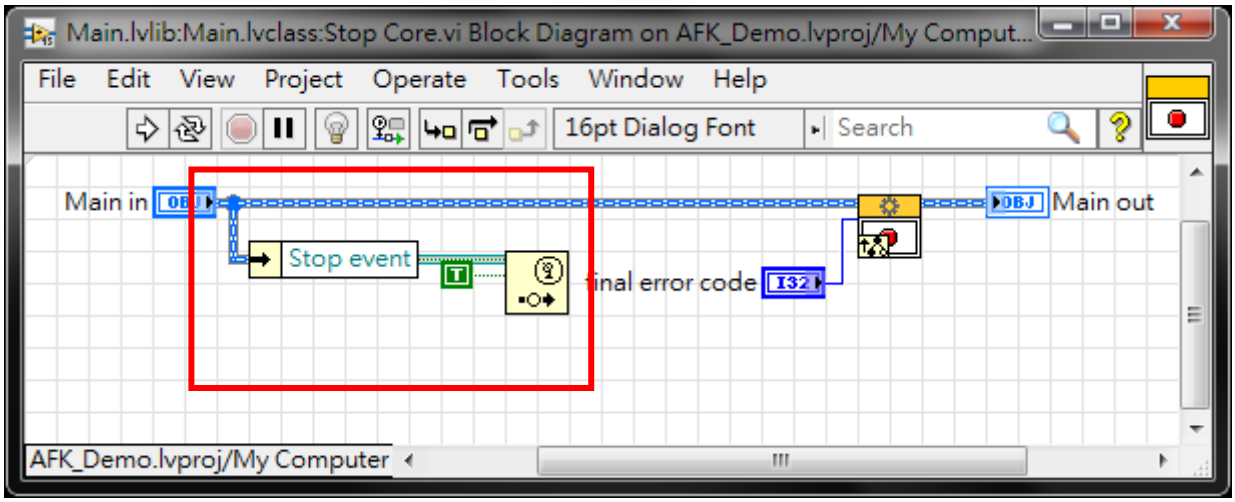


圖：完成 Pre Launch Init.vi

接著改寫 Stop Core，在 Actor 關閉之後，需要在此釋放管理 Stop 訊號的 User Events。

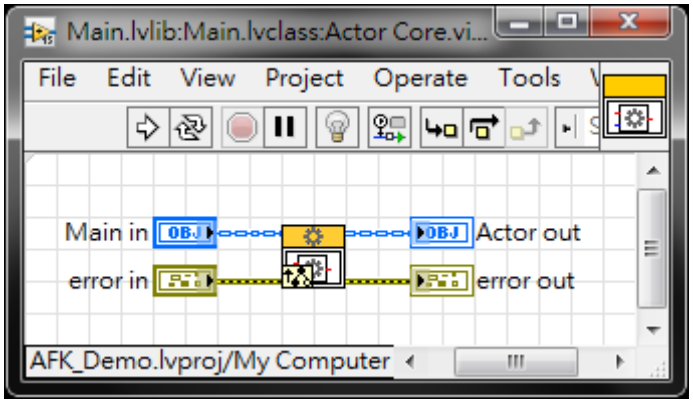


圖：改寫 Stop Core.vi(修改前)

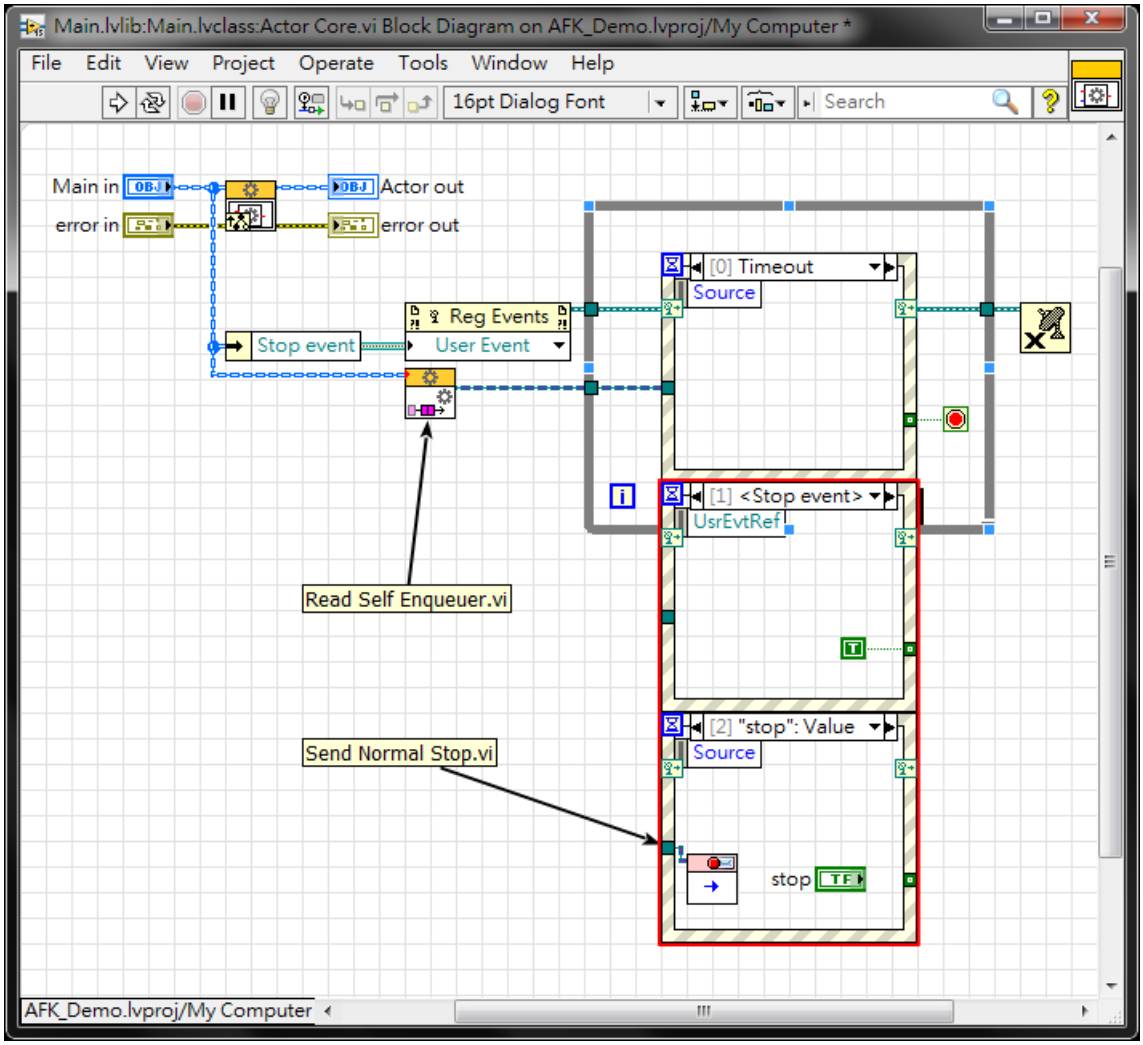


圖：改寫 Stop Core.vi(修改後)

最後改寫 Actor Core，此 Method 為實際運行 Actor 的核心，也需要並在此設計關閉機制。
確保在 Actor Core 停止後可以順利關閉整個 Actor。

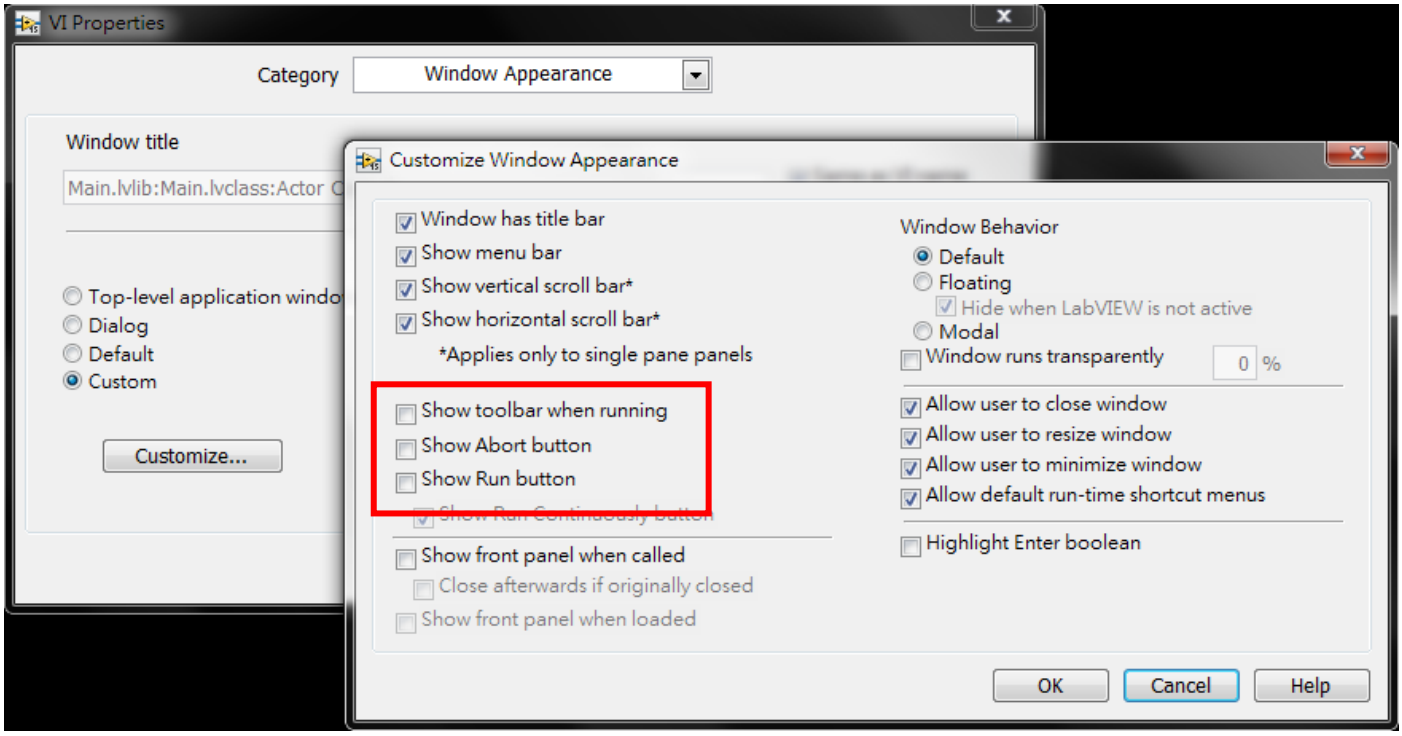


圖：改寫 Actor Core.vi(修改前)



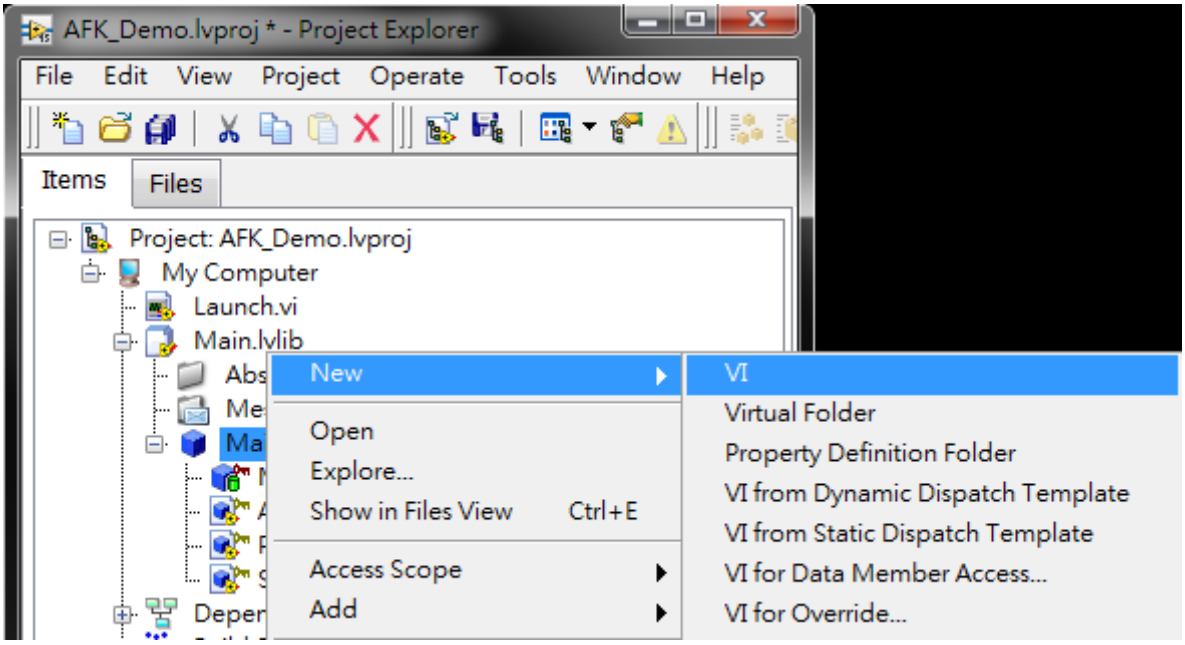
圖：改寫 Actor Core.vi(修改後)

接著修改 Actor Core 的 Customize Window Appearance，由於 Actor 的啟動原理是由底層載入，並利用動態呼叫的方式啟動，而不是從當下的 VI 執行，所以為了確保開發者不要誤觸 VI 本身的 Run/Abort，造成程式關不乾淨而引發的錯誤，建議關閉 Actor Core 的 Abort/Run 按鈕。



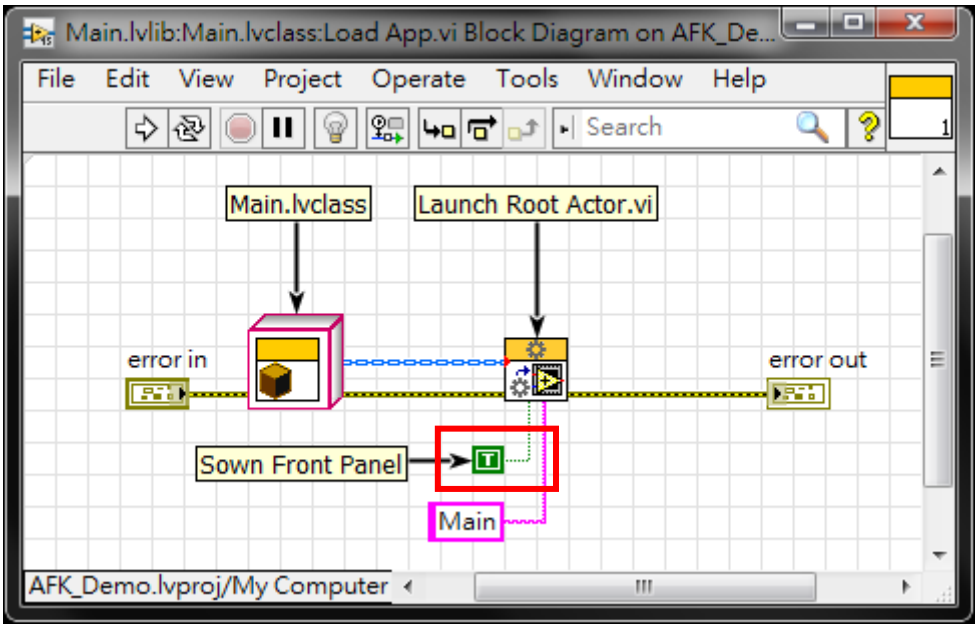
圖：修改 Actor Core.vi 中 VI Properties 內的 Window Appearance

到這邊已經完成 Actor Frame Work 的殼，接著需要設定 Main Actor 的進入點，設計一個 Load VI 作為載入 Actor 的程式，並取名為 Load App。



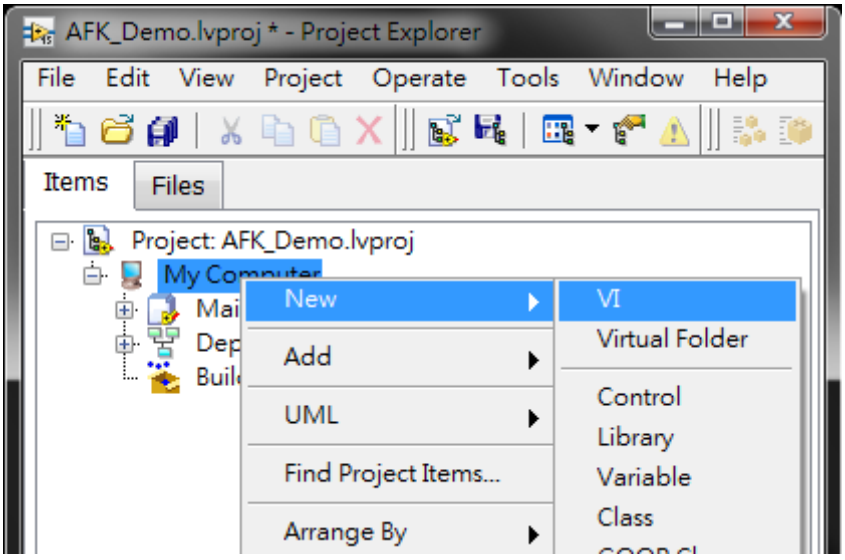
圖：新增一個 VI 並取名為 Load App。
(Main.lvclass 右鍵→ New→ VI)

設計 Load App 時，由於我們將 Main Actor 設定為 UI Actor，所以需要設定 Show Front Panel，用以確保啟動時會開啟介面。



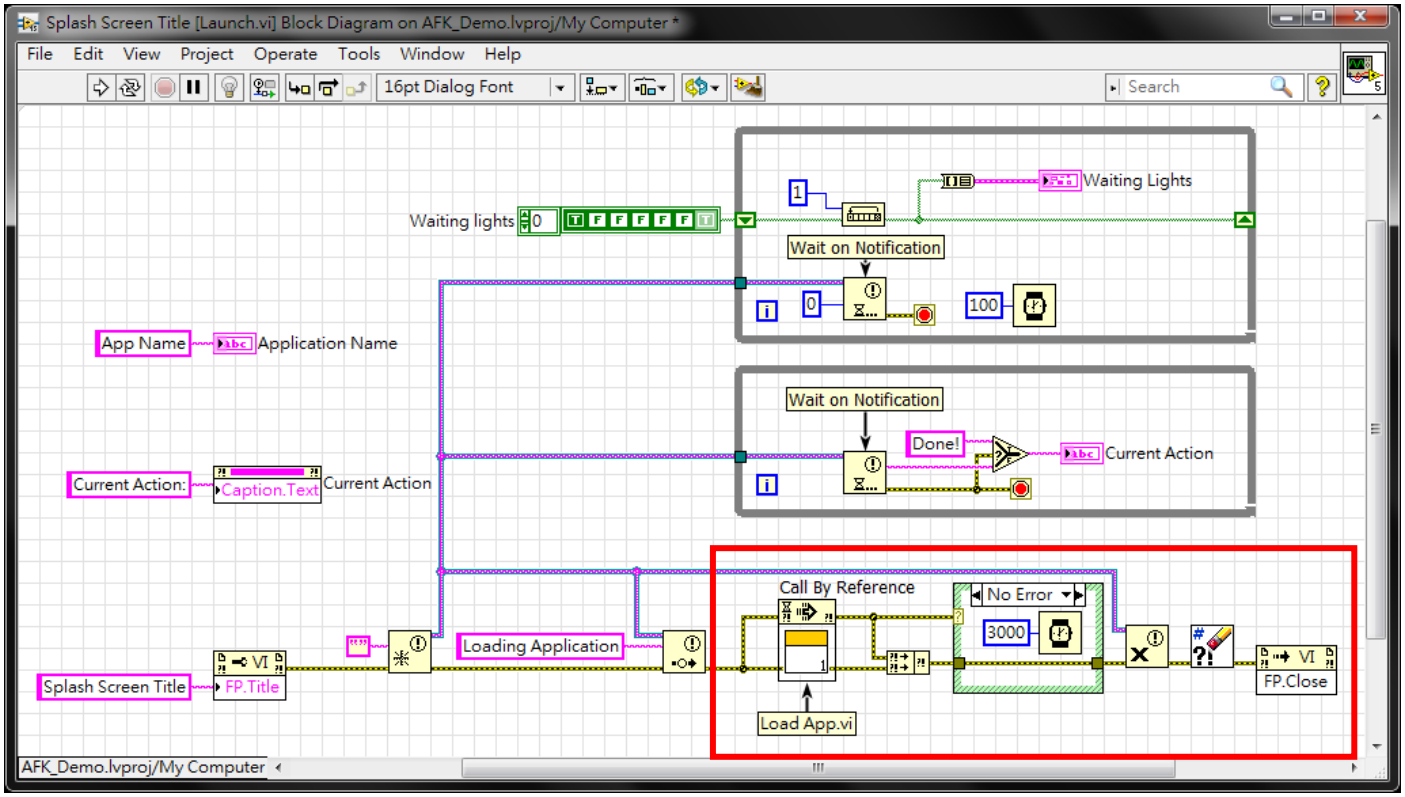
圖：編輯 Load App 的程式碼。

接著設計一個啟動器，作為啟動 Load App 的程式，並取名為 Launch.vi



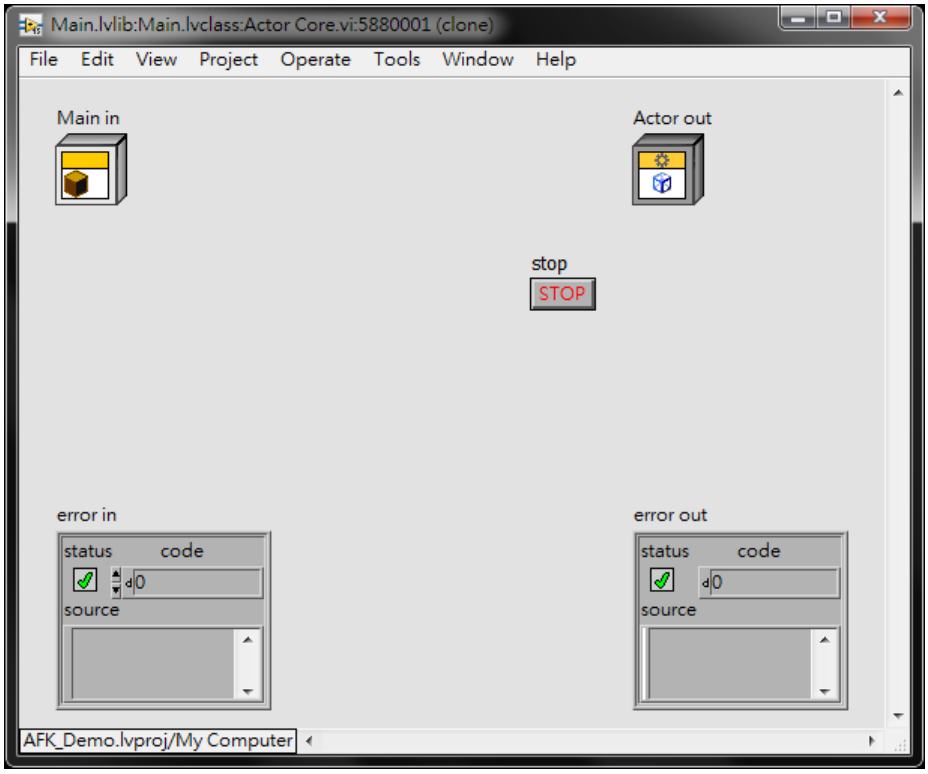
圖：新增一個 VI 並取名為 Launch.vi。
(My Computer 右鍵→ New→ VI)

設計啟動器時，利用 Call By Reference 的方式，讓程式執行時讓載入 Actor 的時候在背景作業。
(主要功能標示於紅框處，其餘部分程式碼只是修飾啟動畫面。)



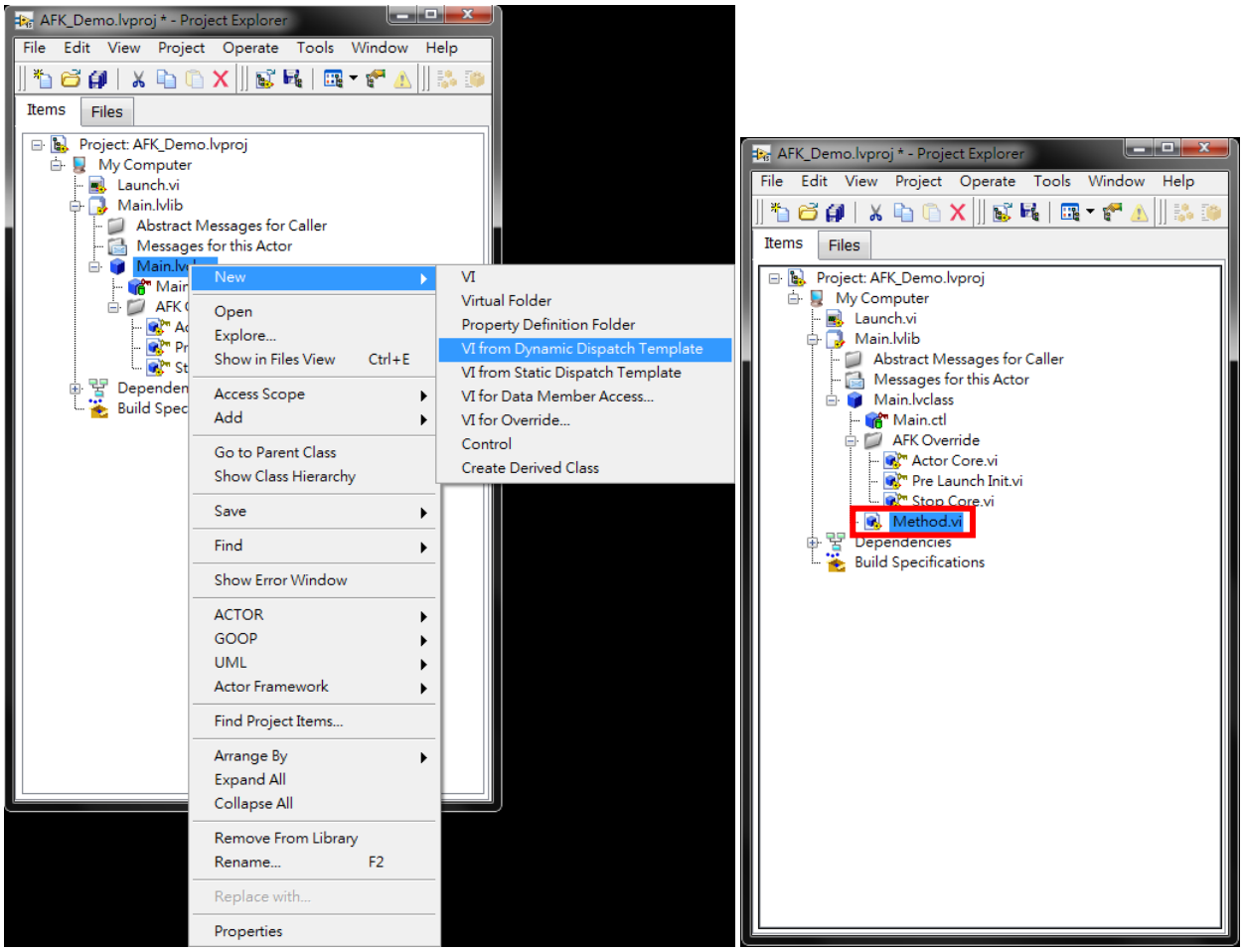
圖：編輯 Launch.vi 的程式碼。

整體已經完成 Actor 的架構，在運行 Launch VI 後，可以看到先前設定的 Main-Actor Core 視窗彈出，並且可以透過 Stop 關閉程式。



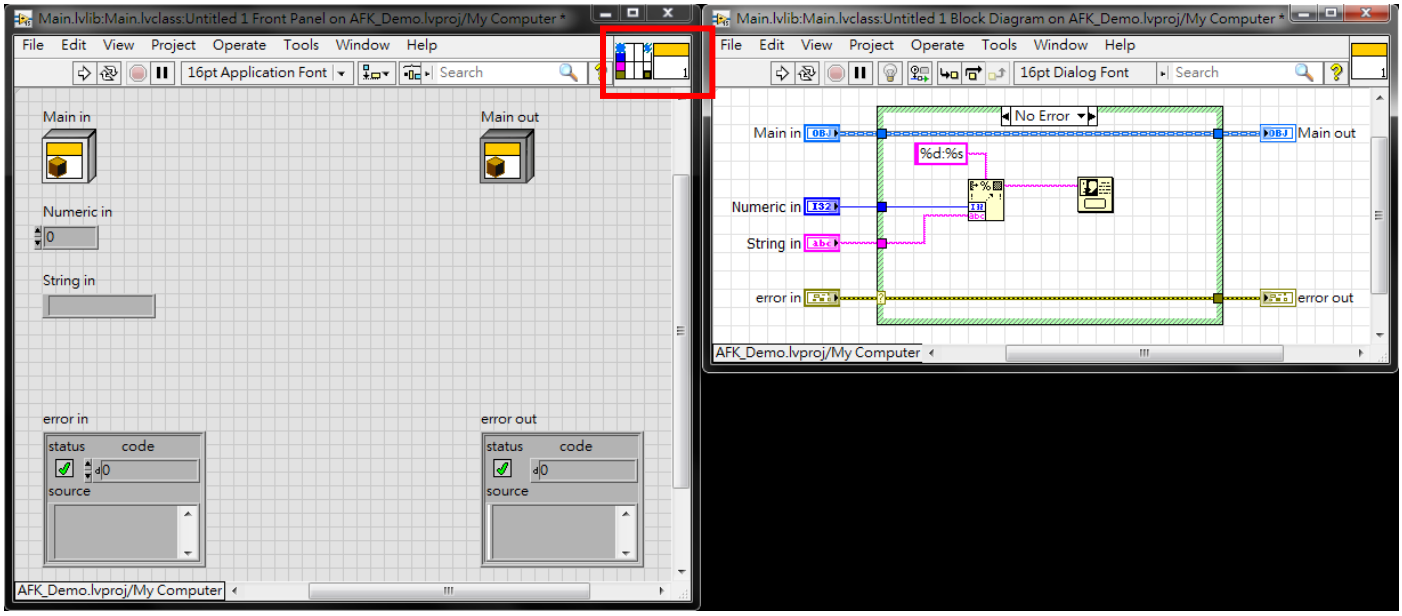
圖：透過 Launch VI 啟動的 Main Actor Core

框架有了之後，就可以新增功能了，第一步先新增基礎的 Method。



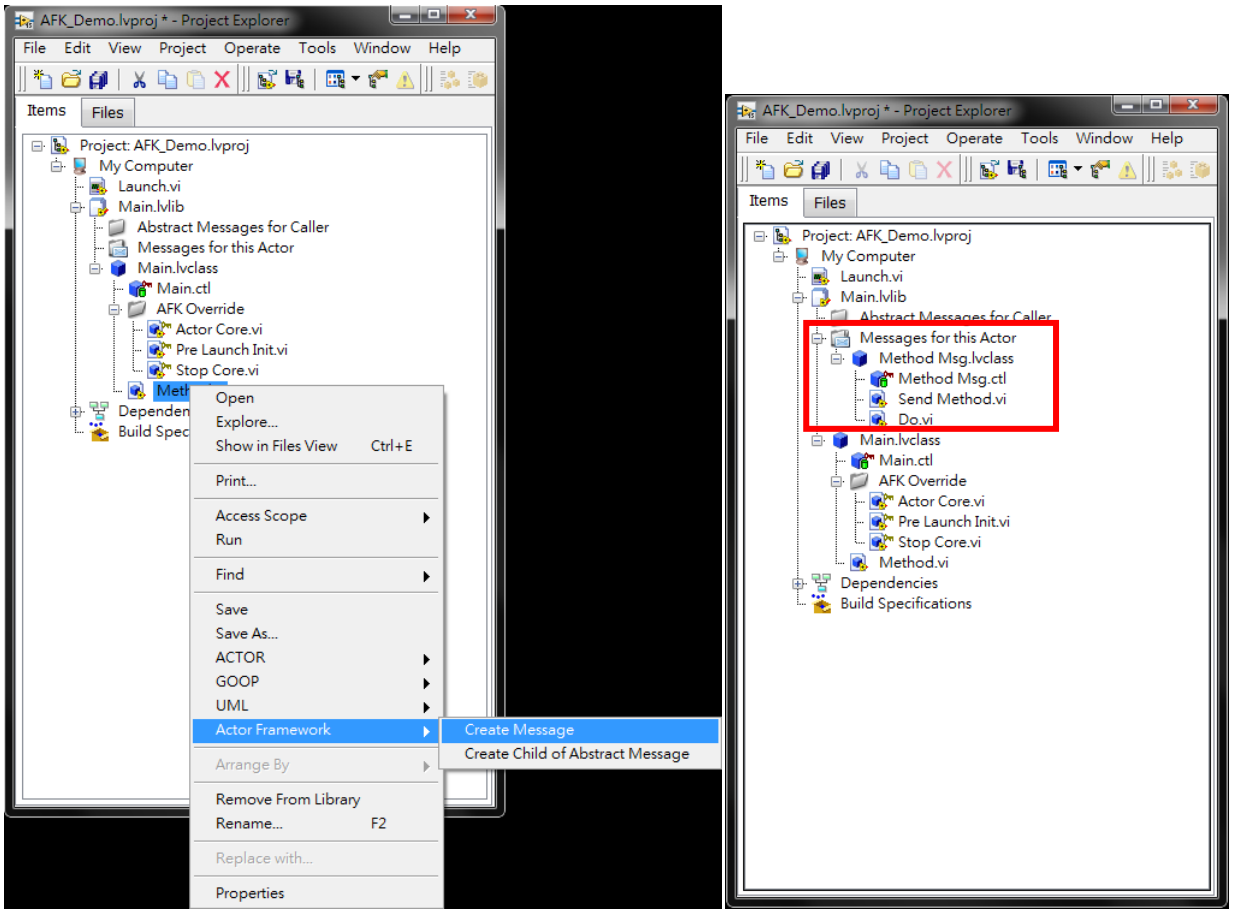
圖：新增 Dynamic Dispatch Template VI，並取名為 Method。

編輯 Method，設計測試功能，根據輸入的數值，彈出對話視窗。**注意要記得接線。**



圖：編輯 Method 程式碼

創造這個 Method 的 Message，創造後的 Message 會被放在 Messages for this Actor 資料夾底下。



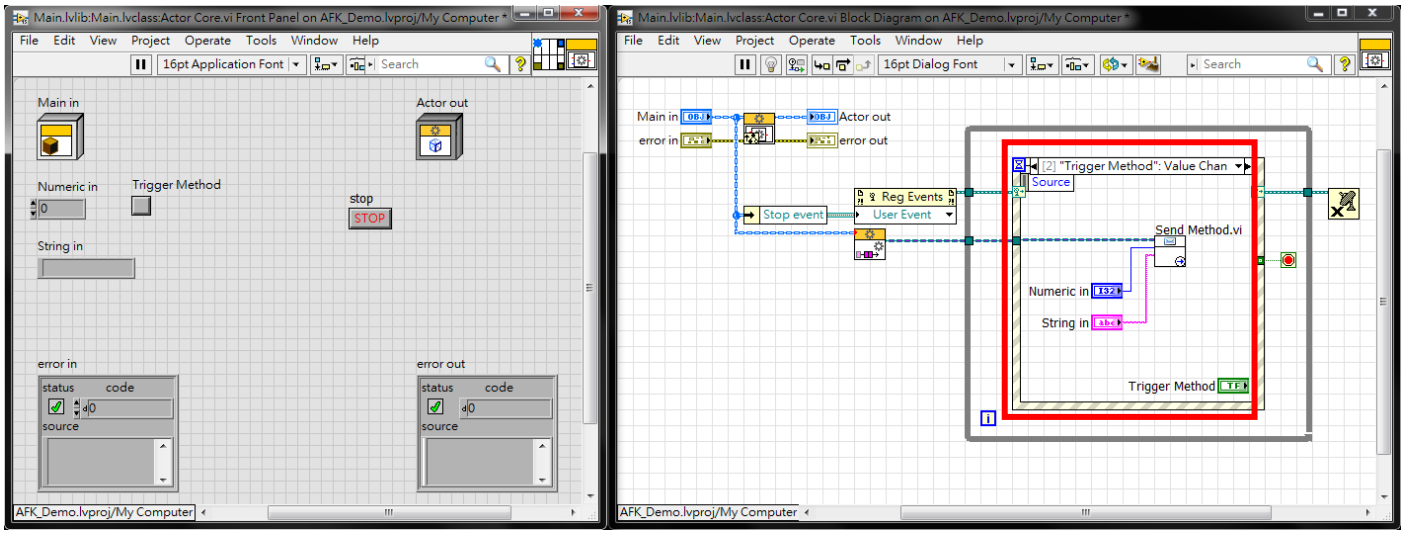
圖：創造 Method 的 Message。

(Method 右鍵→Actor Framework→Create Message)

為了執行這個 Method，必須使用這個被產生出來的 Message。透過 Method Msg.lvclass 底下的 Send Method.vi 來發送 Message。

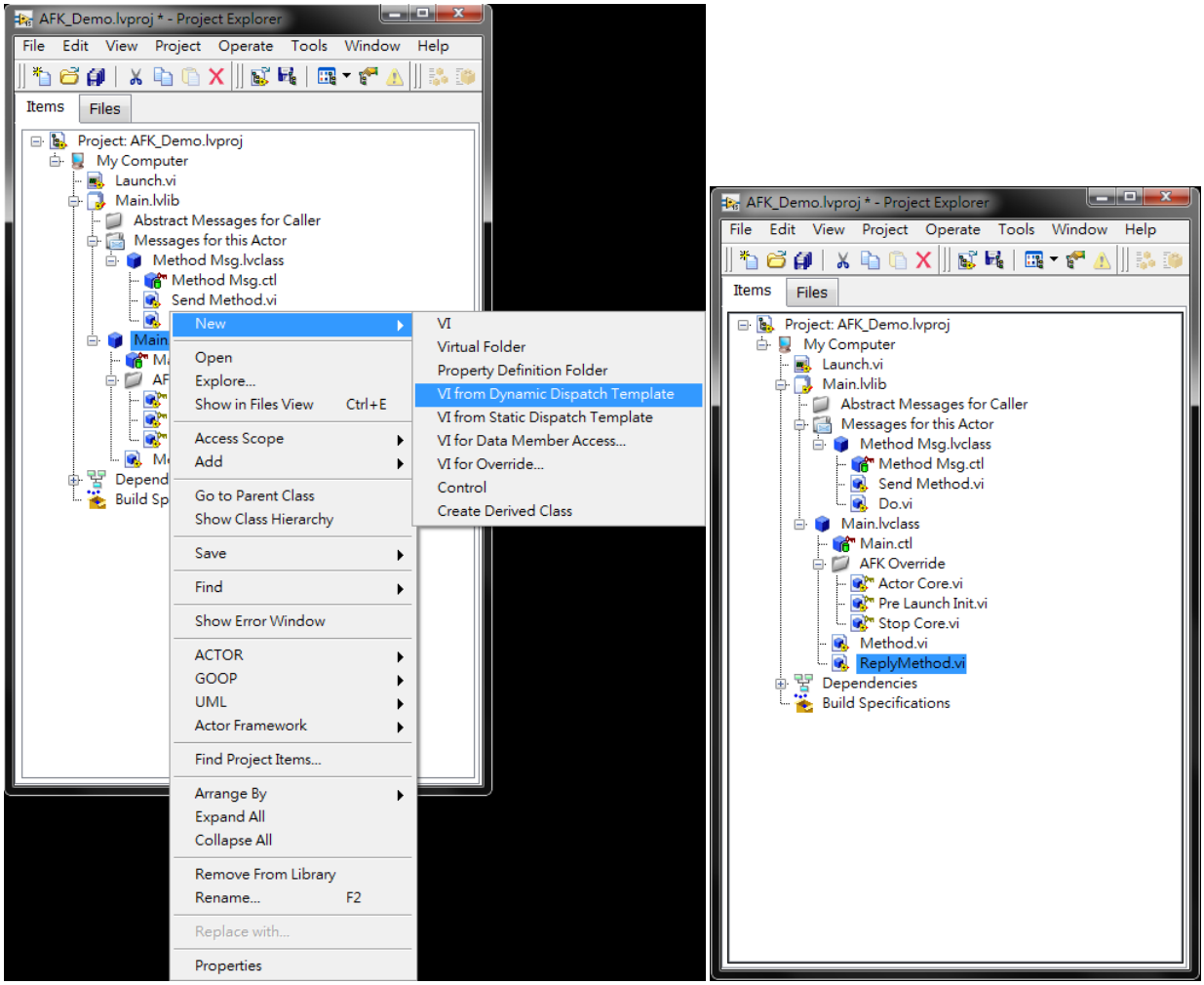
使用方式就是指定 Actor 的 Enqueuer，包含 Method 所需的變數丟進 Send Method.vi。

下圖是將 Method 加入 Main Actor Core，並指定一個 Trigger 訊號。



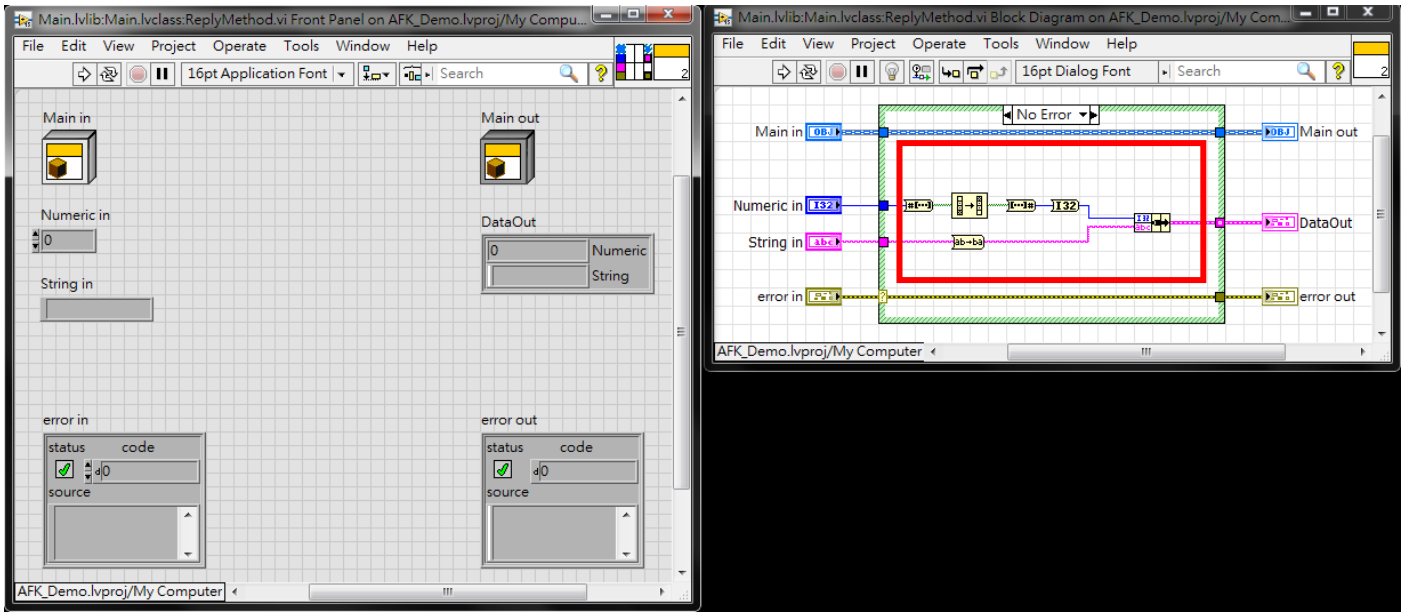
圖、修改 Actor core 加入 Method Message。

接著新增需要回傳值的 Method。



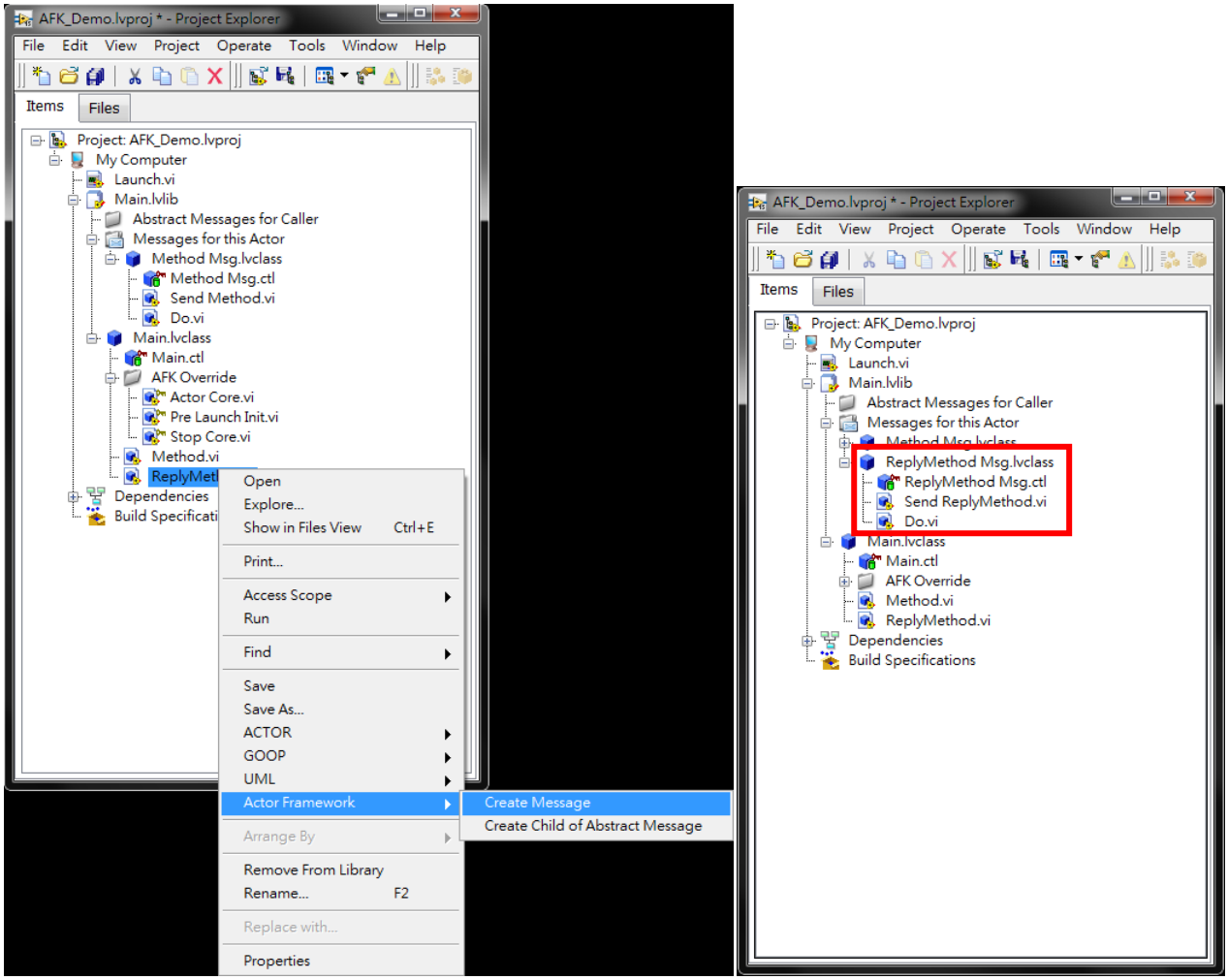
圖：新增 Dynamic Dispatch Template VI，並取名為 ReplyMethod。

編輯 ReplyMethod，設計測試功能，根據輸入的數值，會將資訊 Bundle 起來並回傳 Cluster。



圖：編輯 ReplyMethod 程式碼

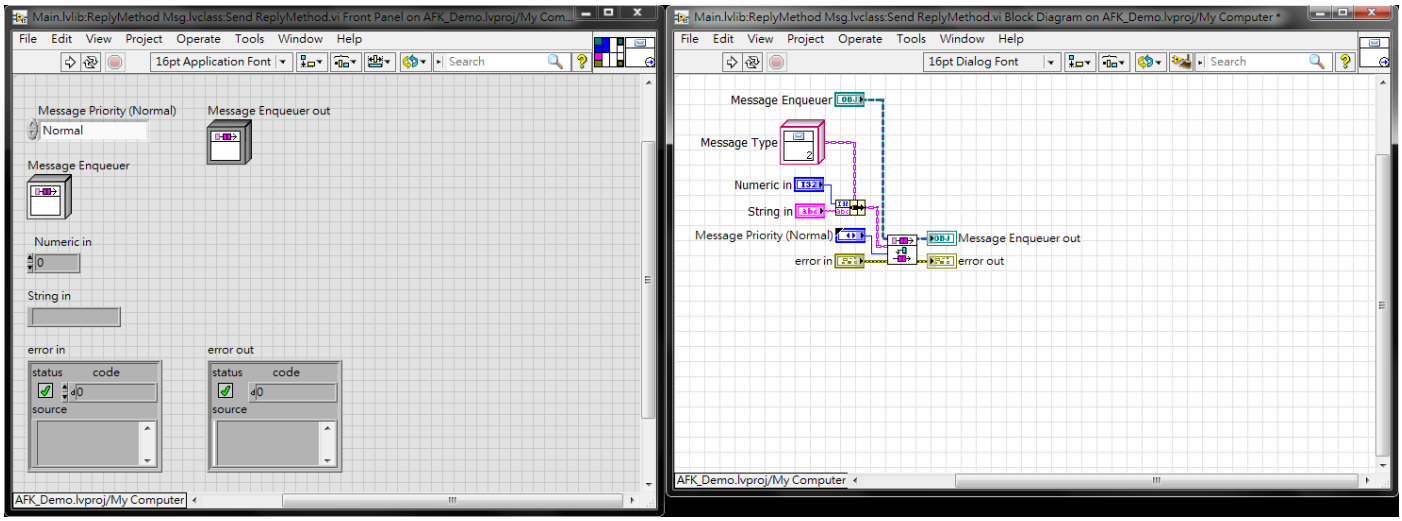
創造這個 ReplyMethod 的 Message，創造後的 Message 會被放在 Messages for this Actor 資料夾底下。



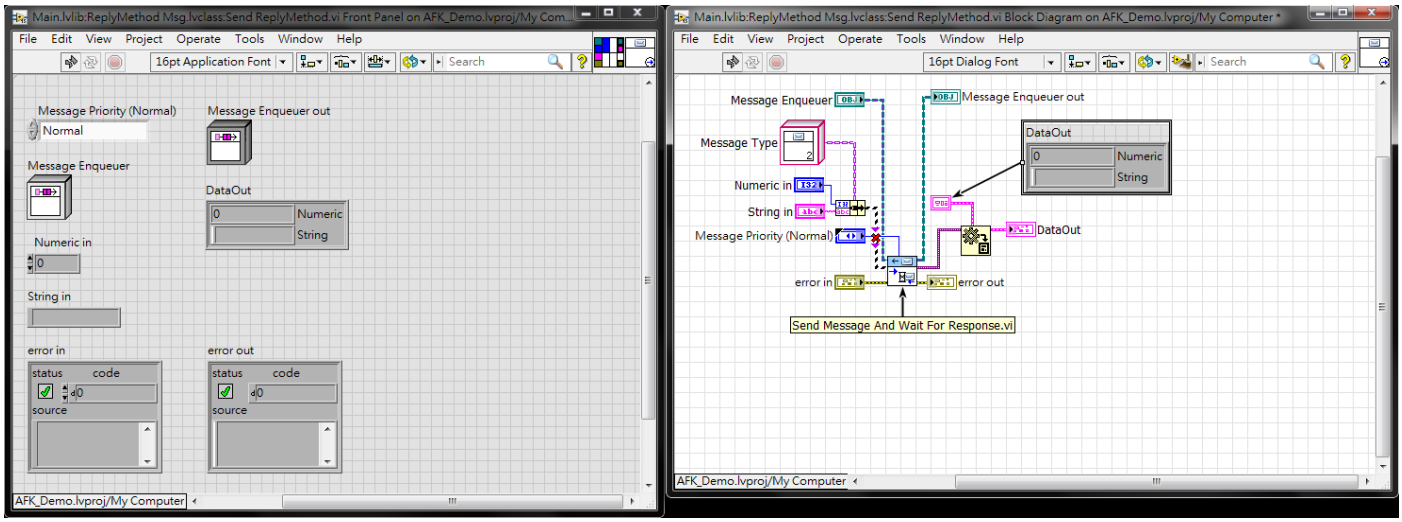
圖：創造 ReplyMethod 的 Message。

(ReplyMethod 右鍵→Actor Framework→Create Message)

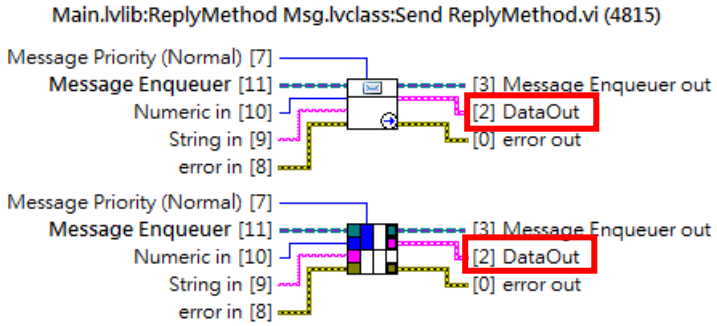
和先前創造 Method message 不同的地方是，這個 ReplyMethod 會需要得到回傳值，所以需要修改 ReplyMethod Msg.lvclass 的 Send ReplyMethod.vi，需要先將 Enqueue.vi 替換為 Send Message And Wait For Response.vi，並且接上 Output 的接點。但由於原本 Send Message And Wait For Response.vi 是繼承自 Reply Msg.lvclass，目前呈現斷線的狀態，所以還需要手動修改繼承關係。



圖：修改 Send ReplyMethod (修改前)

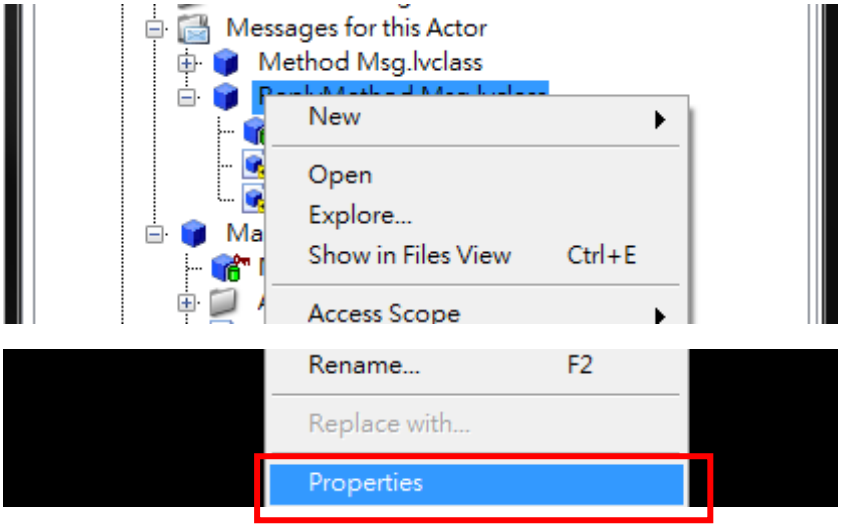


圖：修改 Send ReplyMethod (修改後)



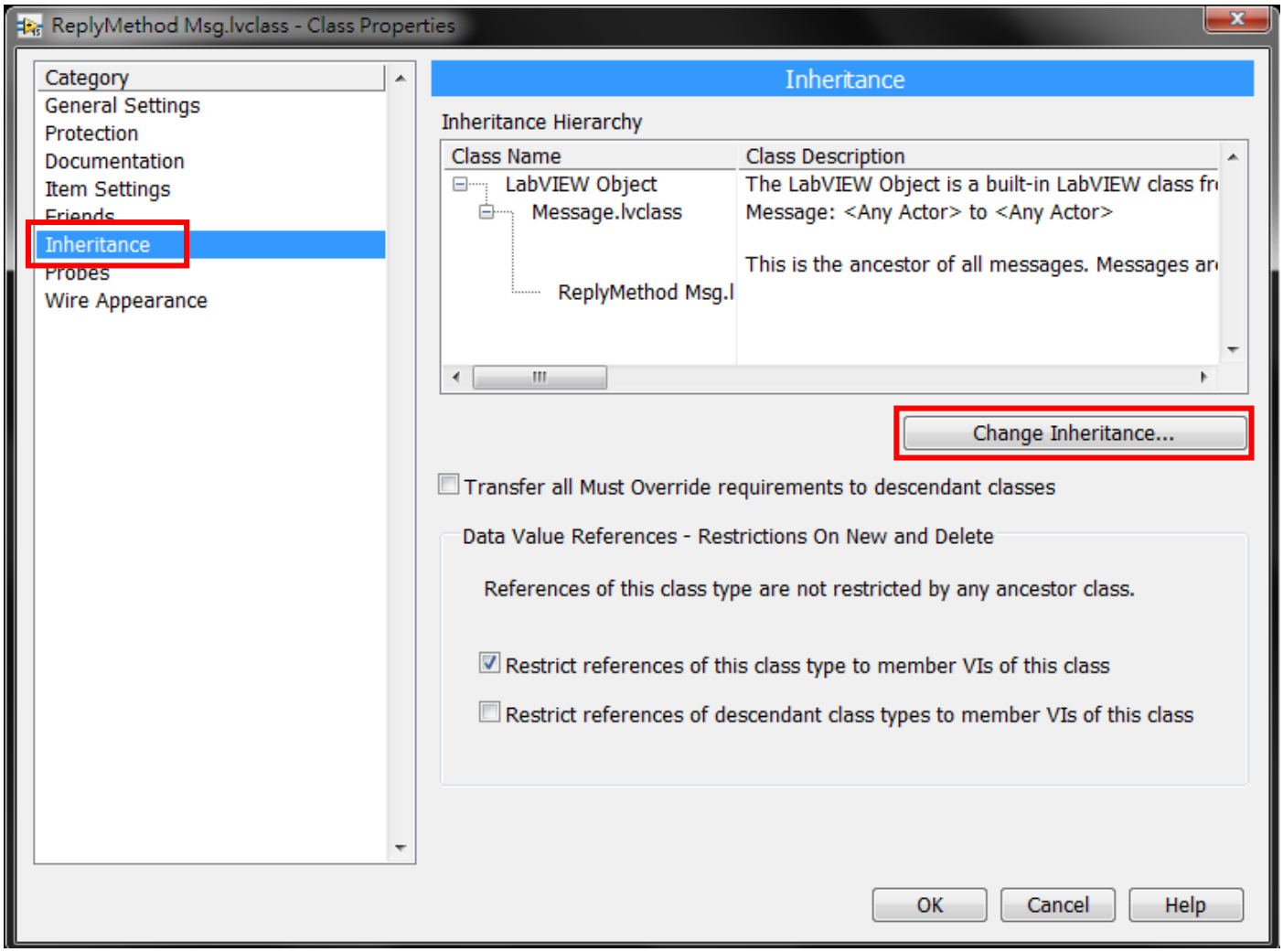
圖：Send Message And Wait For Response.vi 接點

首先，進入 ReplyMethod Msg.lvclass 的 properties 頁面。



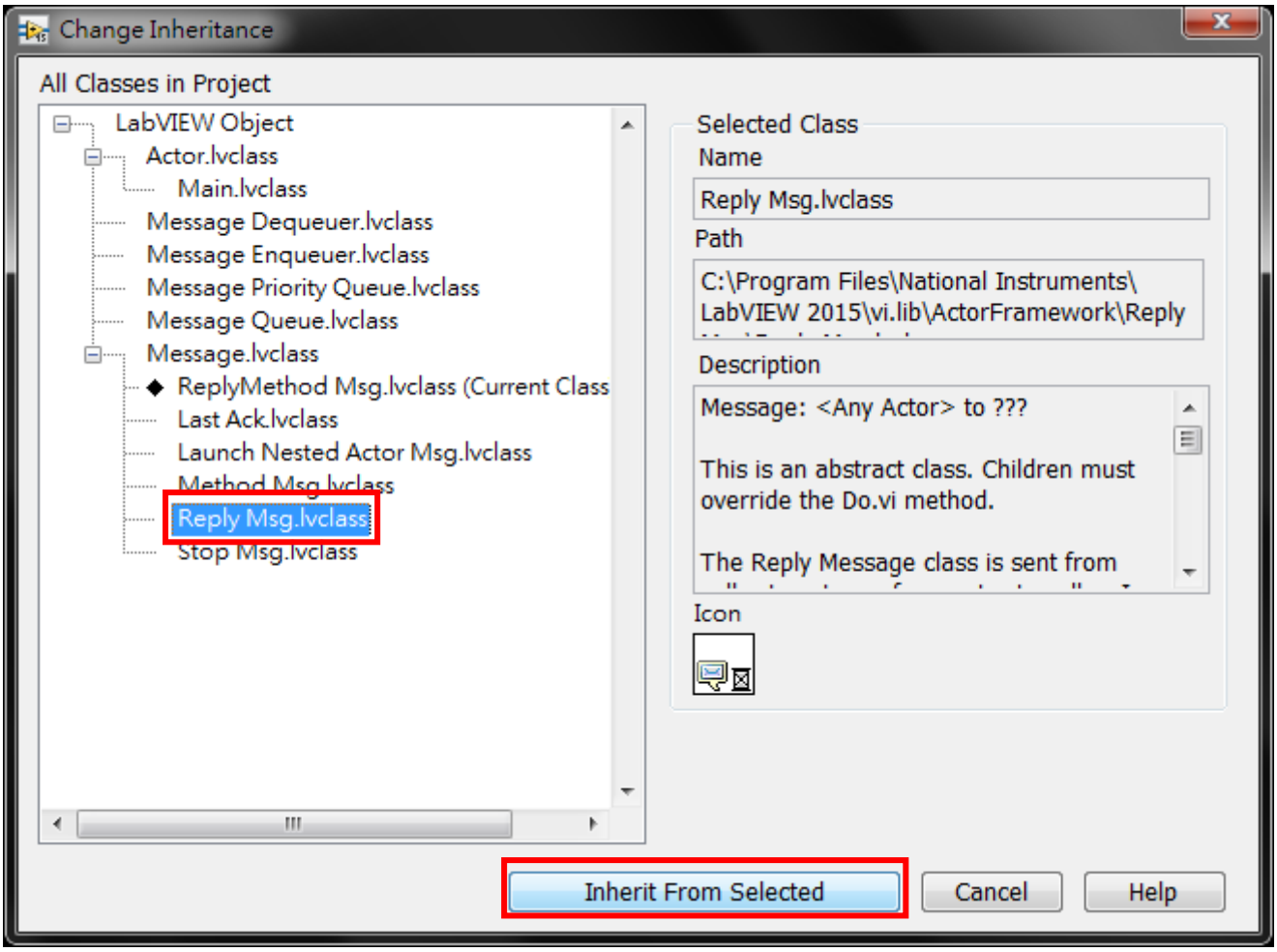
圖：進入 ReplyMethod Msg.lvclass 的 properties 頁面

接著，切換 Inheritance 頁面，並按下 Change Inheritance…。



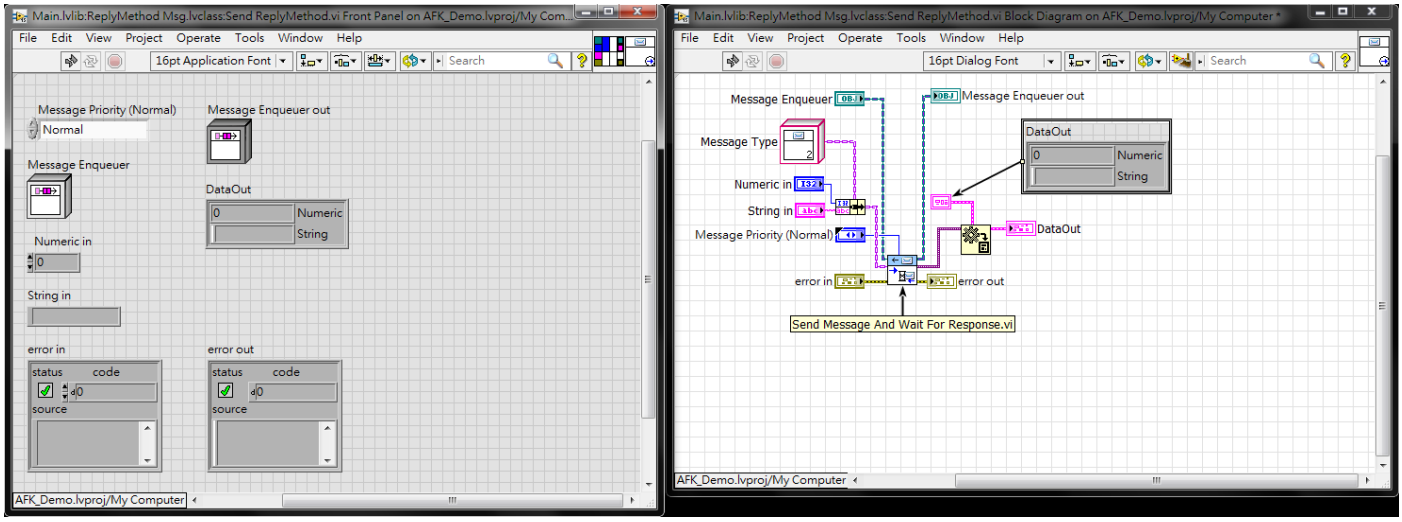
圖：修改 ReplyMethod Msg.lvclass 的繼承關係。

可以看到當前 ReplyMethod Msg.lvclass 是繼承自 Message.lvclass 底下，為了讓 ReplyMethod Msg.lvclass 擁有 Reply Msg.lvclass 的功能，所以選擇 Reply Msg.lvclass 並按下 Inherit From Selected。



圖：選擇 ReplyMethod Msg.lvclass 的繼承自 Reply Msg.lvclass。

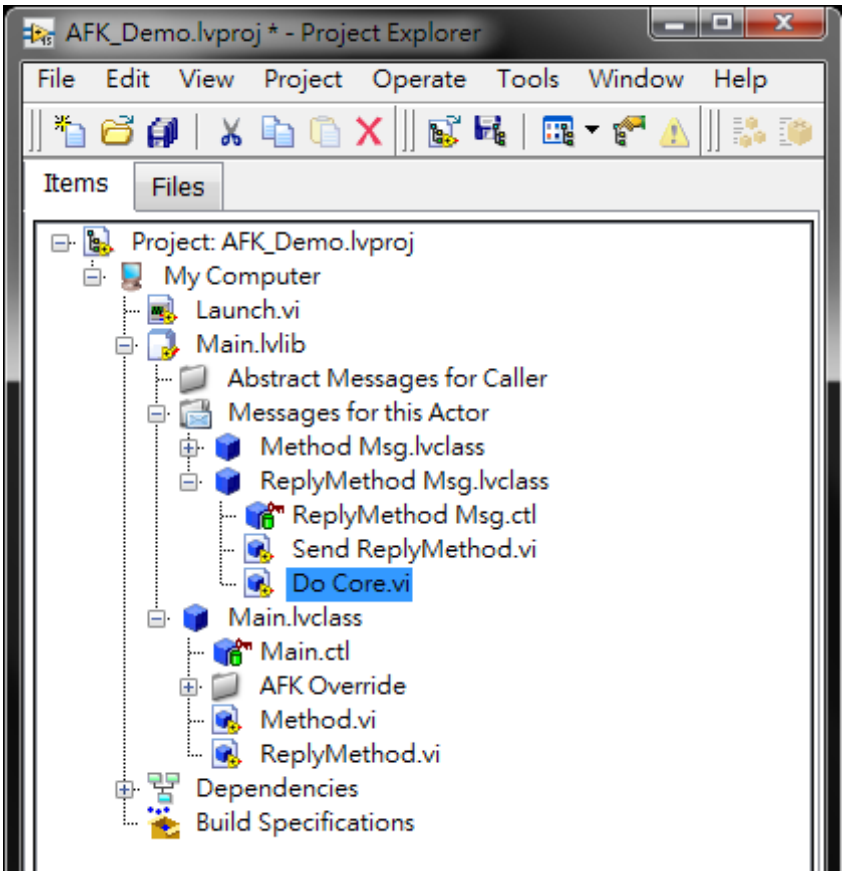
可以發現完成修改繼承關係後，原本的斷線已經解決了，但是會發現 VI 還是呈現無法通過編譯。需要繼續修改其他部分。



圖：解決 ReplyMethod Msg.lvclass 的斷線問題。

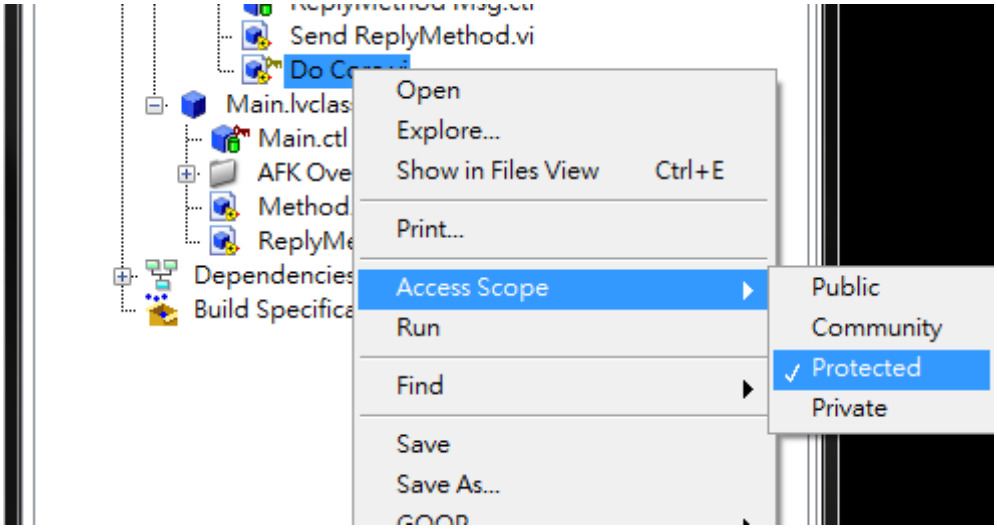
因為原本 Message.lvclass 執行 send message 的方法是利用 Do.vi，而 ReplyMethod Msg.lvclass 則是因為 override Do.vi，所以執行動作的方式改由 Do Core.vi 啟動，所以這個部分也需要進一步修改。

首先將原本的 Do.vi 更名為 Do Core.vi。



圖：將 Do.vi 更名為 Do Core.vi。

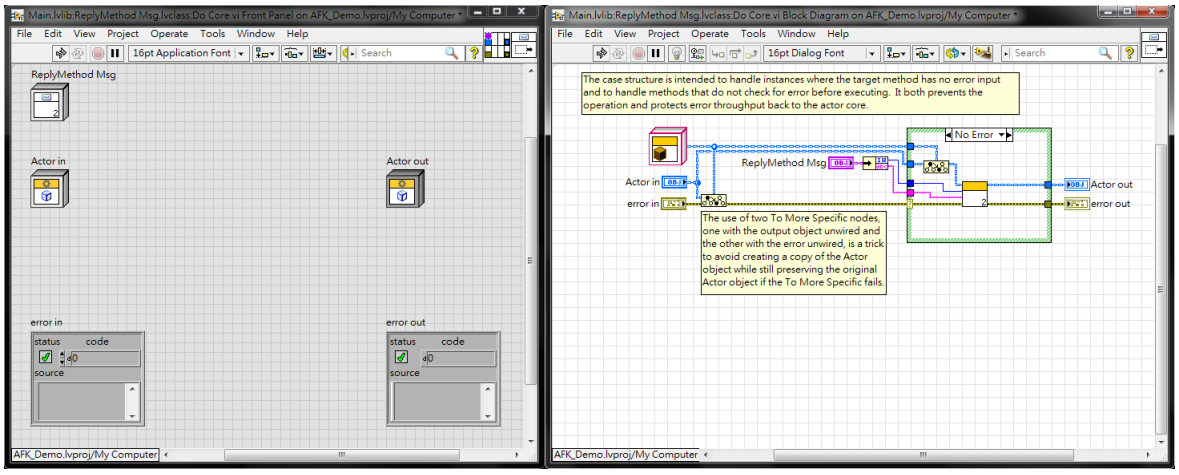
將 Do Core.vi 的 Access Scope 修改成跟 Base Method 一樣的 Protected。



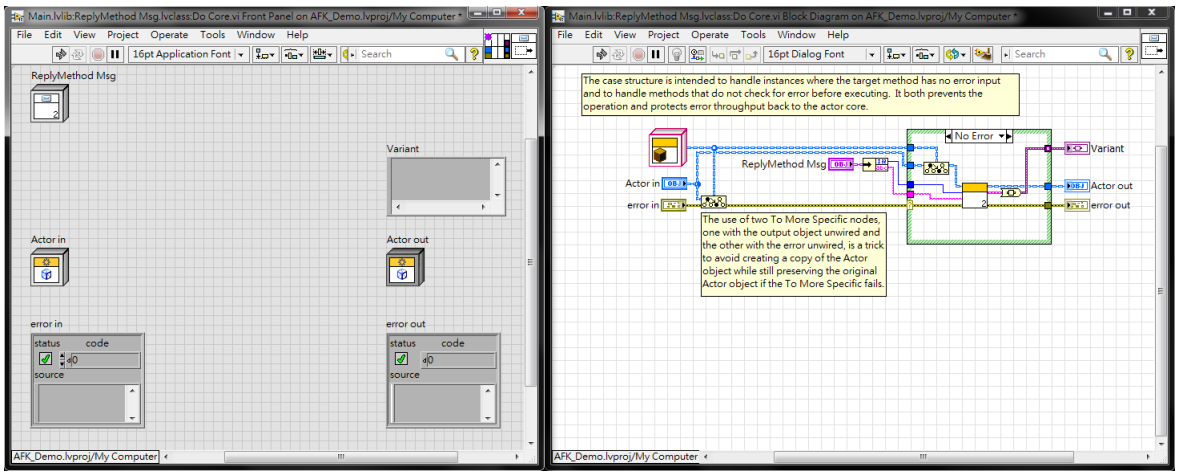
圖：將 Do Core.vi 的 Access Scope 修改成 Protected。

接著修改 Do Core.vi 的程式碼，需要把原本 ReplyMethod 的輸出轉換為 Variant 並輸出，並且依照 Base Method 重新指定接點，但因為接點屬性還沒有一致，所以目前 Do Core.vi 依舊無法通過編譯。

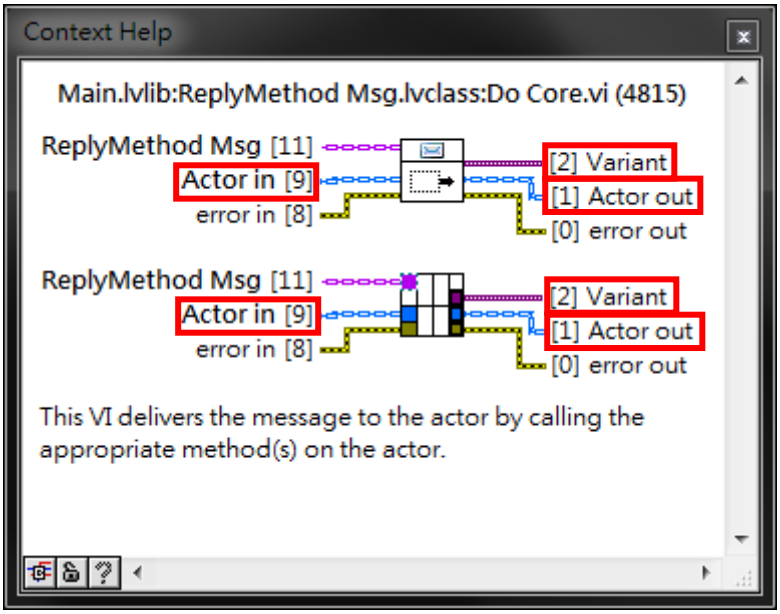
最後依照 Base Method 的接線規則，將 Actor in 的接點屬性設定為 Required。修改完後可以發現原本無法通過編譯的 VI 已經將問題排除完畢了。



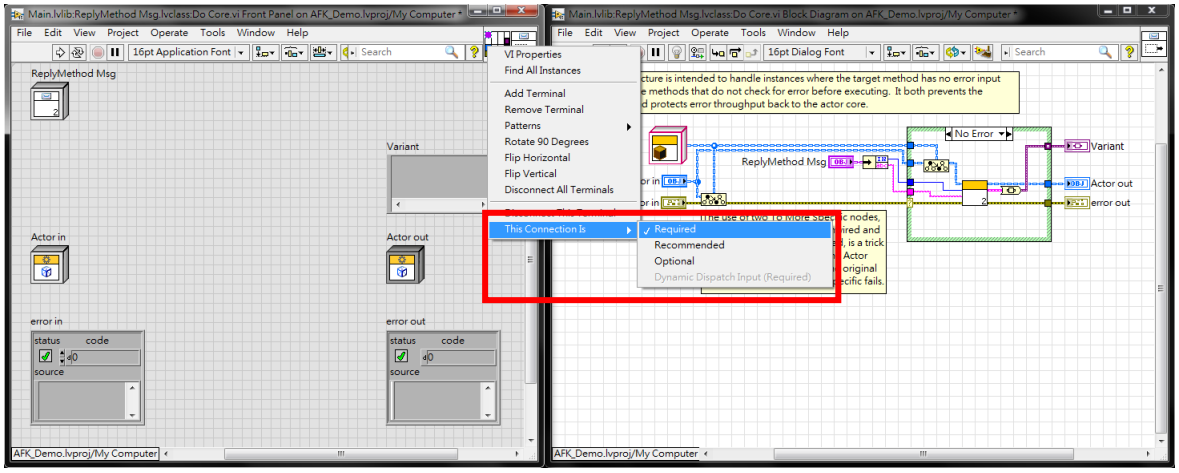
圖：修改 Do Core.vi (修改前)



圖：修改 Do Core.vi (修改後)



圖：按此設定 Do Core.vi 接線

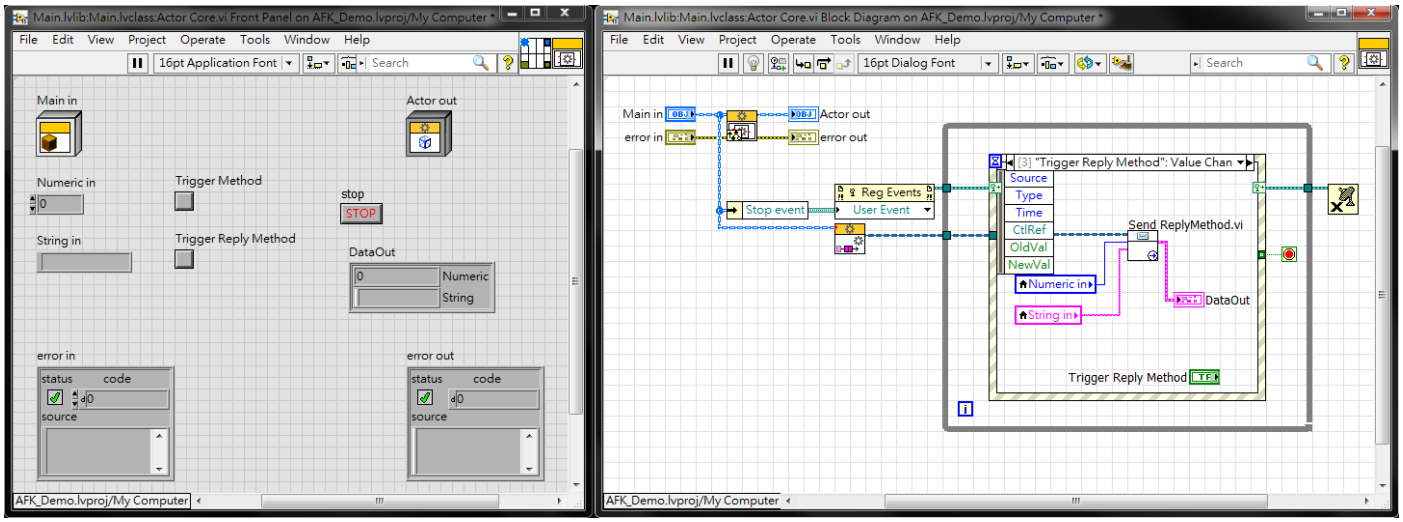


圖：修改 Do Core.vi 中 Actor In 的接點屬性。

(在接點上按右鍵→This Connection Is→Required)

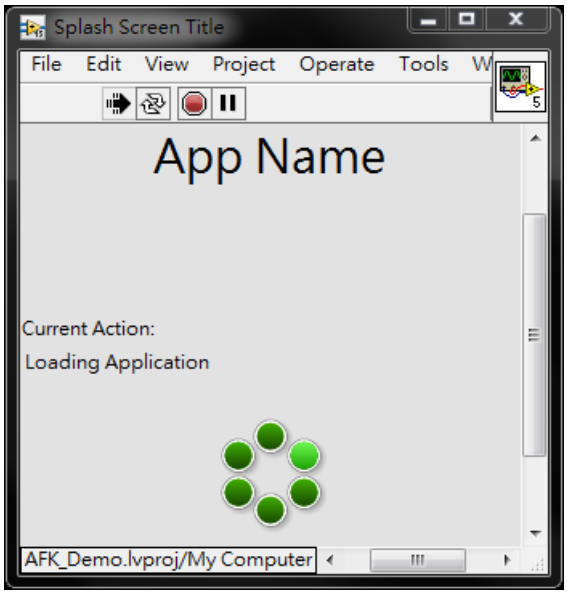
為了測試新加入的 Method，和之前一樣，只要透過底下的 Send ReplyMethod.vi 便可以啟動。

底下是將 ReplyMethod 加入 Main Actor Core，並指定一個 Trigger 訊號。與先前不同的是，現在可以透過 Send Message 來取得 Method 的回傳值了。

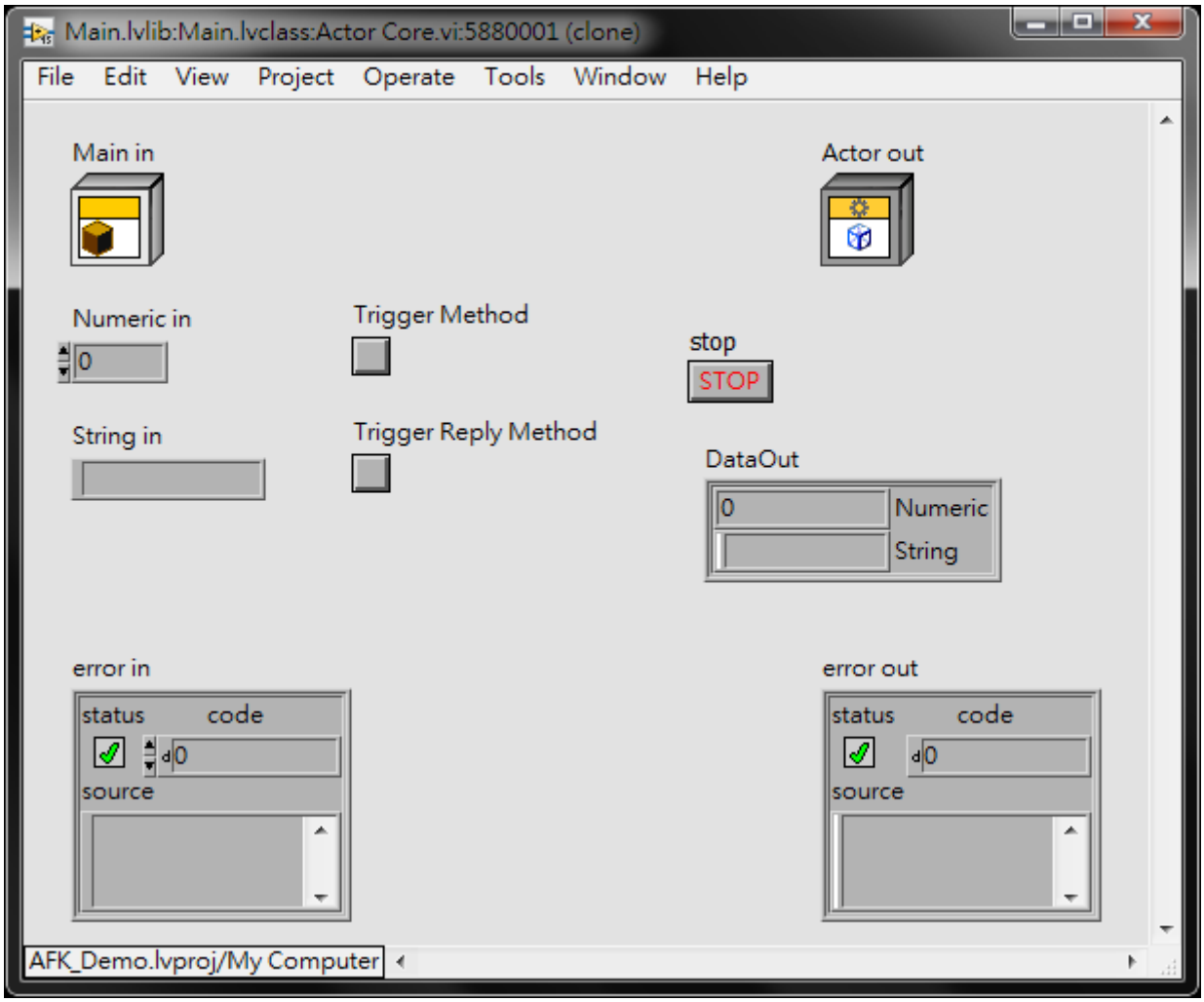


圖、修改 Actor core 加入 ReplyMethod Message。

最後測試一下已完成的部分，一樣是從 Launch.vi 進入 Main Actor。

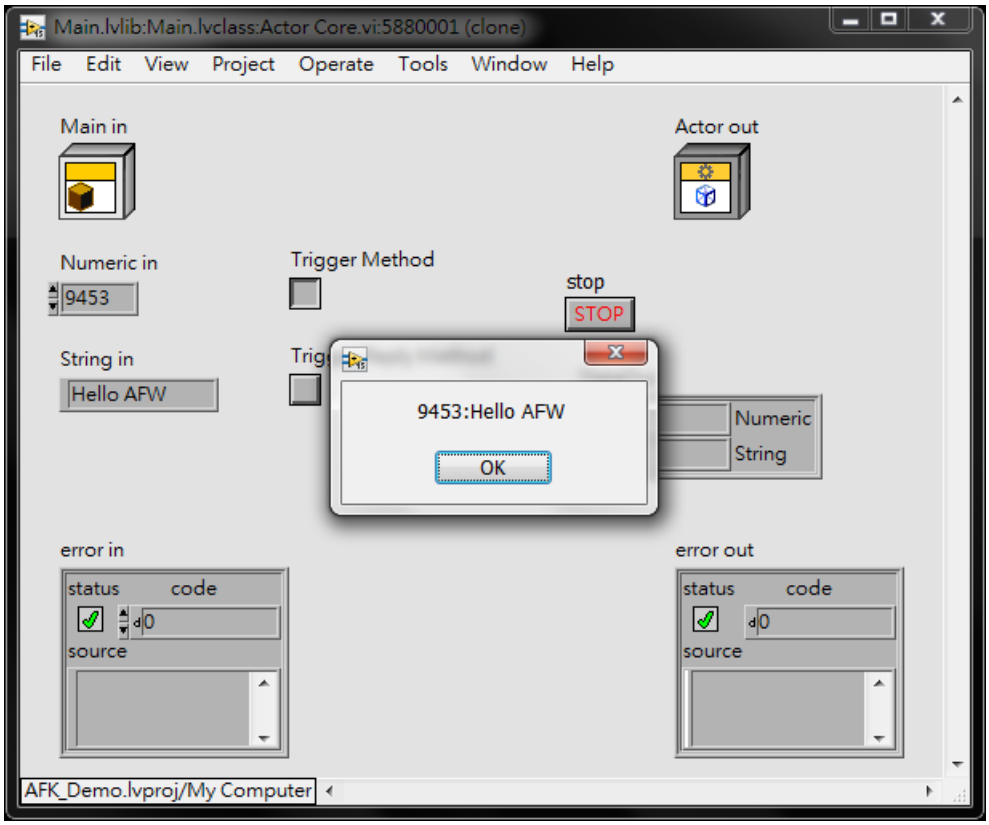


圖、開啟並執行 Launch.vi 已進入 Main Actor。



圖、Main Actor 的主畫面。

測試 Method：按下 Trigger Method 後，會彈出一個包含 Numeric in 和 String in 內容的視窗。



圖、測試 Method 的功能。

測試 ReplyMethod：按下 Trigger Reply Method 後，會將 Numeric in 和 String in 的內容後更新到 DataOut。