

單元11： 事件結構

主題：

- a. 事件結構概述
- b. 如何使用事件結構
- c. 使用事件結構須注意的事項

什麼是事件導向程式語言

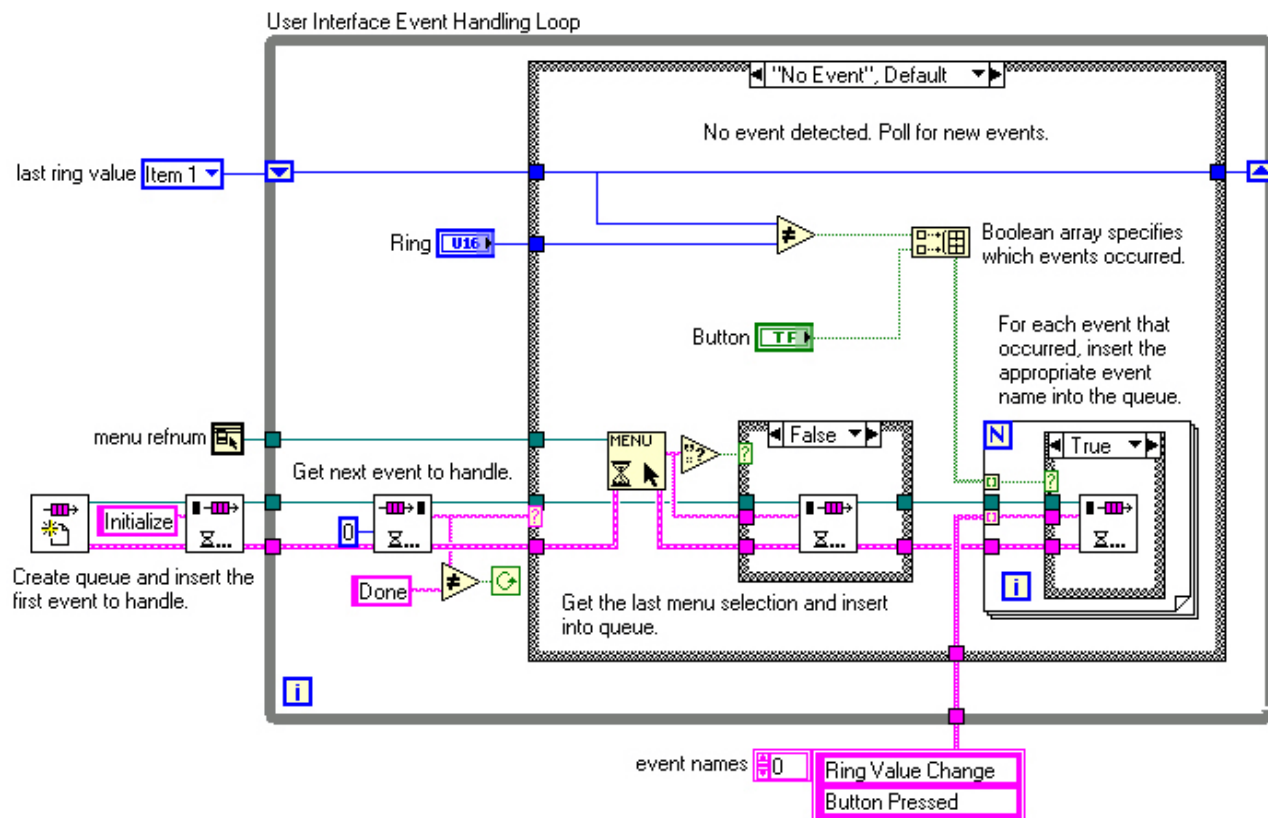
- Event-driven是一個在其他如LabWindows CVI、Visual Basic等程式語言環境中早就是一個普遍的範例。有了Event-driven的功能您的應用程式可以休息到有一件您有興趣的事件發生在Front Panel為止，所以作業系統可在這一段時間內將CPU交給其他的程式使用。
- 執行的順序由事件(event)決定
- 執行的程式碼反應相對應的事件
- 不需要輪詢系統的狀態有否改變
- 不可能會漏失事件或不依順序執行
- 事件(Event)是非同步的觸發動作
 - 使用者介面事件(User interface events)
 - 程式事件(Programmatic events)

什麼是事件結構(Event Structure)?

- 一個事件結構 (Event Structure) 就好像一個 “Wait on Occurrence” function與一個Case structure的混合物。就好像Case structure一樣它也具備多層subdiagram，每一層都可規劃來Handle一個或多個事件(event)，例如滑鼠移動或某個鍵被按下等事件，當您放入一個事件結構 (Event Structure) 在您的Diagram當中就好像其他的物件一般，它的執行流程規則跟一般的沒有什麼不一樣
- 當LabVIEW執行到事件結構 (Event Structure) 時，將使得這個VI進入睡眠狀態，直到有一個被設定的事件(event)發生，這個時候事件結構 (Event Structure) 就會自動醒過來並且依照使用者設定的條件執行相關的動作
- 每一個subdiagram的內側的左邊都有一個Event Data node，提供使用者來定義或使用有關這個事件(event)的相關訊息。這個node看起來與功能就好像一個Unbundle by Name function，所以使用者可以改變它的大小並選擇自己需要的資料欄位來使用。

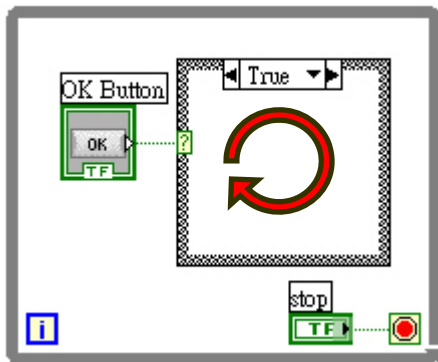
LabVIEW6i之前的版本

■ 在LabVIEW6i之前，沒有事件結構，當要寫一個與人機介面有互動的程式是相對較複雜的，而使用while loop來不斷掃描人機介面的結果也造成系統效能低落等問題



過去與現在

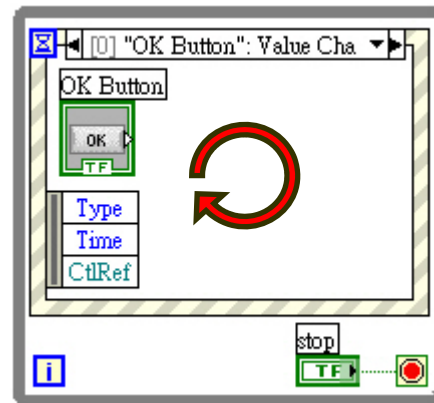
過去



缺點：

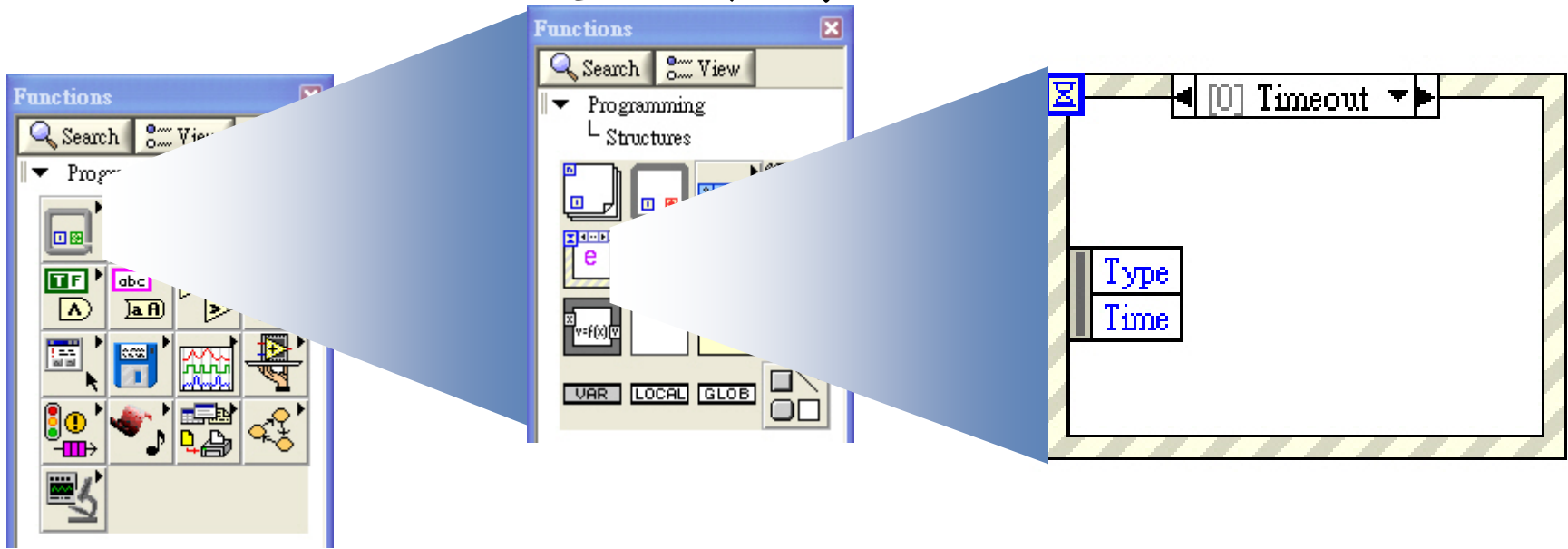
1. 造成效能低落
2. 容易漏失事件

現在



解決了左邊的缺點，效能提升，而且不會漏失事件

LabVIEW6.1後，才有Event Structure



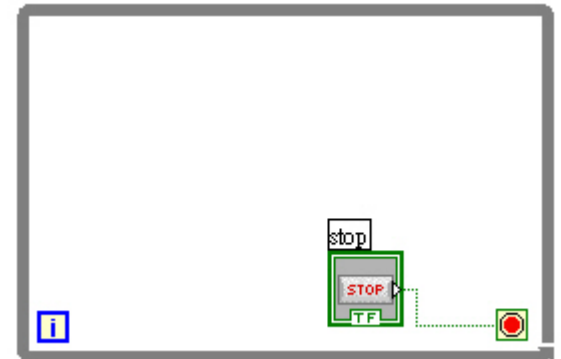
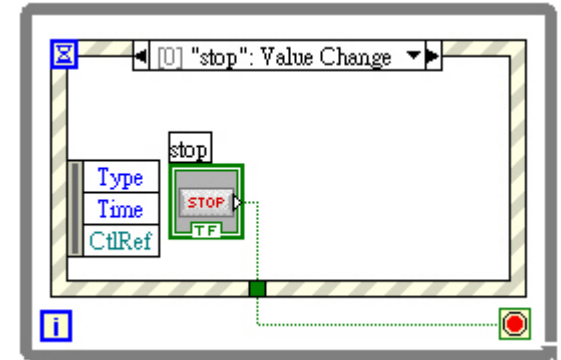
■ 可以觸發的事件列舉

- Menu被選取的事件
- 變更控制元的值
- 關閉視窗
- 條整視窗尺寸

- 滑鼠點擊
- 滑鼠從上方經過某元件
- 滑鼠從否元件上方離開

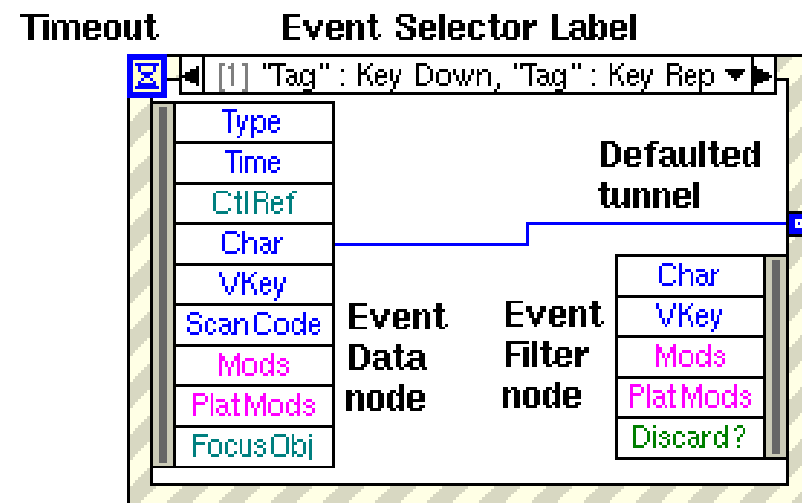
基本Event Structure使用

- 通常Event Structure會與While Loop搭配使用
- 每一個while loop的執行都會使用到一個Event
- Event structure的左邊有一個Event Data node，提供使用者來定義或使用有關這個事件(event)的相關訊息
- 可以設定多種觸發方式，例如，當人接介面的元件一或元件二任何之一有觸發動作時，就會觸發Event



事件結構(Event Structure)的組成

- **Event Selector Label** – 此事件結構的名稱標籤
- **Timeout** – 單位為ms，意思是經過多少時間及認定逾時，並執行Time Out內的程式碼。如果沒有接Time Out值，就是永不逾時的意思。
- **Event Data Node** -提供使用者來定義或使用有關這個事件(event)的相關訊息
- **Event Filter Node** – 提供使用者可以取消此事件的繼續進行，如：取消關閉視窗
- **Defaulted Tunnels** – 當沒有值傳出Event Structure通道時，就以預設值來代替



Notify Events與Filter Events

Events

- Key Down
- Key Repeat
- Key Up
- Mouse Down
- Mouse Enter
- Mouse Leave
- Mouse Move
- Mouse Up
- Value Changed

➔ Notify Events

提醒LabVIEW：使用者已經做了一個觸發動作。**發生事件之後執行程式，因事件已發生不能取消。**

➔ Filter Events

在使用者觸發的事件尚未執行前先讓Filter Event觸發，以進行判斷。甚至可以中止這個使用者事件的繼續進行。**發生事件之後先確認再執行，事件可取消。**有Event Filter Node與Event Data Node可以使用。


練習 11.1 – 熟悉 Event Structure

- 開啟檔案「<CD>\Ch11\Event Tracking.vi」
- 執行此程式
- 點選人機介面上的按鈕，看看程式有什麼反應
- 切換到程式區，觀察事件結構的運作方式

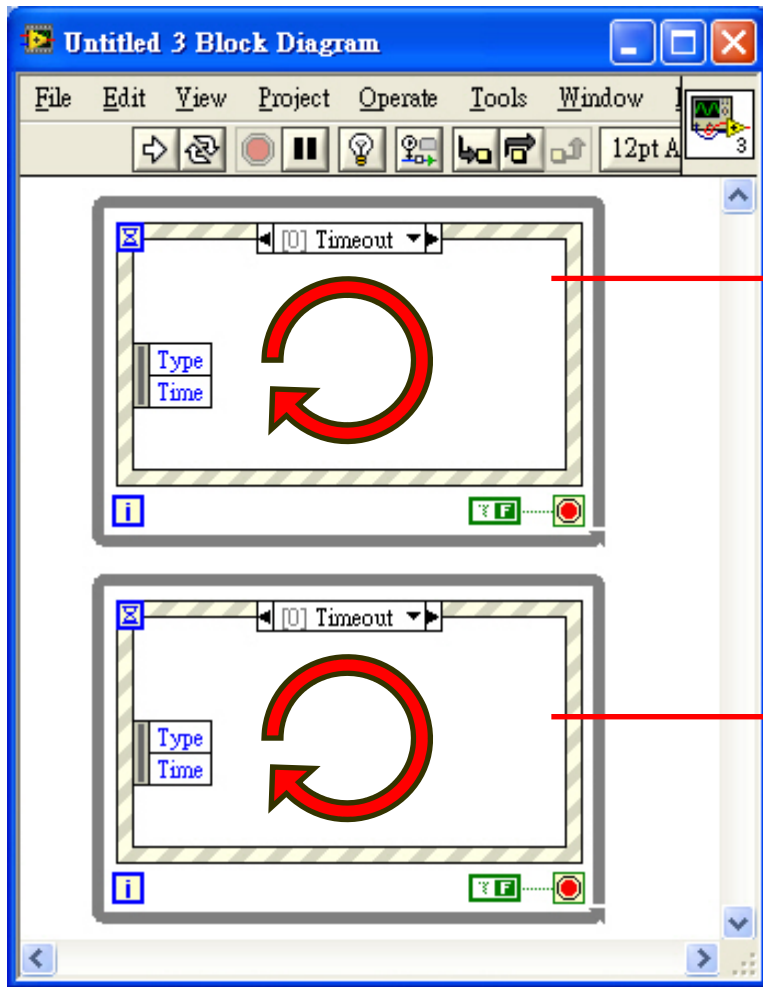
練習11.2 – 寫一個事件導向程式

- 寫一個事件導向的程式，這個程式包含一個按鈕與一個數字的接收器。
- 當你按下按鈕時，數字的接收器就「加1」

使用事件結構須注意的事項

- 假設event1與event2都放在同一個Event Structure中，當觸發event1時，LabVIEW預設值是鎖定人機介面，當event1執行完畢後，才會開放人機介面的互動。解決方法為取消「 Lock front panel until the event case for this event completes」的核取方塊，即可避免
- 取消了上述的核取方塊，如果event1要執行很久，那麼雖然event2會被觸發，也必須要等到event1執行完畢後，才會開始執行event2。解決方法為把需要長時間進行的event獨立出來單獨放在一個While Loop中，把需要短時間的event集中在一起。

把長、短時間的Event分離



花費較短時間的Event

需花費較長時間的Event

- 善用LabVIEW的多執行緒的特性，可以令迴圈分別進行

- 迴圈之間的參數傳遞可以使用Local Variable或是Global Variable

本章重點回顧

- 事件結構在撰寫使用者互動頻繁的應用程式時，十分常用，時常與While Loop搭配使用
- 使用使建結構可以節省花費在偵測使用者較面改變的系統資源，並且能在使用者觸發事件後，以最快方式觸發程式內部
- 事件結構有兩種，Notify Events與Filter Events，Filter Events可以阻止觸發的繼續進行，如終止關閉視窗的動作。
- 使用多執行緒的觀念，把event分類為需要花費長時間的與短時間的事件，並放在不同的event structure