{A}

視窗使用者介面類別

Visual C# 2008

在 Microsoft .Net Framework 中，處理視窗使用者介面之類別主要是 System.Windows.Forms 命名空間，此命名空間概分爲以下之類別：

- 表單（Form）：設計視窗、Modeless 視窗、MDI 多重文件介面之類別。
- 控制項：包括設計使用者介面的標籤（Label）、文字方塊（TextBox）、指令鈕（Button）、選項按鈕（RadioButton）、下拉式方塊（ComboBox）、核取方塊（CheckBox）、群組方塊（GroupBox）、清單檢視（ListView）、工具列（ToolBar）等常用控制項。
- 選單功能表：如選單（Menu）、選單項目（MenuItem）和快速鍵功能表（ContextMenu）等。
- 通用對話盒：提供 Windows 標準對話盒，如開啓檔案（OpenFileDialog）、儲存（SaveFileDialog）、色彩對話盒（ColorDialog）、字型選取（FontDialog）、列印之版面設定（PageSetupDialog）、預覽列印（PrintPreviewDialog）、列印（PrintDialog）等類別。
- 訊息對話盒：提供顯示及取得資訊的訊息對話盒類別（MessageBox）。

► Form 類別

System.Windows.Forms.Form 類別是建構 Windows 應用程式的基礎類別，可用以設計視窗、Modeless 視窗、MDI 多重文件介面等應用。其組件名稱爲 System.Windows.Forms，存在 System.Windows.Forms.dll 檔案。

建構一個表單，實際上是繼承 System.Windows.Forms.Form 類別，例如在 Visual Studio 2008 中，使用新增專案建立 Windows 應用程式，Visual Studio 2008 會自動產生一 Windows 視窗表單（預設表單的大小爲高度 300 像素與寬度 300 像素）。

需注意的是，C# 2008 與 C# .Net 2002 或 2003 的最大差別在於 Windows Form 設計工具所產生的程式碼與使用者定義之程式碼所存放的檔案有所不同。

C# .Net 2002 或 2003 是將 Windows Form 設計工具所產生的程式碼與使用者定義之程式碼存放於同一檔案中，例如 Form1.cs，其缺點是由於程式碼包括設計工具所產生的程式碼，因此感覺有些複雜：

```
using System;
using System.Drawing;
```

```
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsApplication1
{
  /// <summary>
  /// Form1 的摘要描述。
  /// </summary>
  public class Form1 : System.Windows.Forms.Form
  {
    /// <summary>
    /// 設計工具所需的變數。
    /// </summary>
    private System.ComponentModel.Container components = null;

    public Form1()
    {
      //
      // Windows Form 設計工具支援的必要項
      //
      InitializeComponent();

      //
      // TODO: 在 InitializeComponent 呼叫之後加入任何建構函式程式碼
      //
    }

    /// <summary>
    /// 清除任何使用中的資源。
    /// </summary>
    protected override void Dispose( bool disposing )
    {
      if( disposing )
      {
        if (components != null)
        {
```

```
            components.Dispose();
          }
        }
        base.Dispose( disposing );
      }

      #region Windows Form Designer generated code
      /// <summary>
      /// 此為設計工具支援所必需的方法 - 請勿使用程式碼編輯器修改
      /// 這個方法的內容。
      /// </summary>
      private void InitializeComponent()
      {
        this.components = new System.ComponentModel.Container();
        this.Size = new System.Drawing.Size(300,300);
        this.Text = "Form1";
      }
      #endregion

      /// <summary>
      /// 應用程式的主進入點。
      /// </summary>
      [STAThread]
      static void Main()
      {
        Application.Run(new Form1());
      }
    }
  }
```

而 C# 2008 則是將 Windows Form 設計工具所產生的程式碼與使用者定義之程式碼存放於不同的檔案中，例如 Form1.Designer.cs 與 Form1.cs，其中 Form1.Designer.cs 存放由 Windows Form 設計工具所產生的程式碼，Form1.cs 則是存放使用者定義之程式碼。由於設計工具所產生的程式碼可由 Windows Form 設計工具修改，使用者甚少會自行修改，因此使用者只要負責維護如 Form1.cs 之程式碼則可。

請參考範例 A-1。其中 Form1 繼承 System.Windows.Forms.Form 類別，以建構 Windows 視窗表單：

► **Form1.Designer.cs：**

```
namespace FormApplication
{
  partial class Form1
  {
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources
    /// should be disposed; otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
      if (disposing && (components != null))
      {
        components.Dispose();
      }
      base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
      this.components = new System.ComponentModel.Container();
```

```
        this.AutoScaleMode =
          System.Windows.Forms.AutoScaleMode.Font;
        this.Text = "Form1";
      }

      #endregion
    }
  }
```

► **Form1.cs：**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace FormApplication
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
  }
}
```

System.Windows.Forms.Form 類別的建構函式爲：

```
public Form()
```

包含以下常見之屬性、方法及事件：

屬性	說明
AcceptButton	設定或取得按下 Enter 鍵時，所按下的按鈕
ActiveForm	取得應用程式目前現用（Active）的表單
ActiveMdiChild	取得應用程式目前現用的多重文件介面（MDI）子視窗
Anchor	設定或取得控制項附著於容器的邊緣 *
AutoScale	表單是否要調整大小以配合表單上所使用的字型高度
AutoScaleBaseSize	設定或取得表單自動縮放的基礎大小
AutoScroll	表單是否啓動自動捲軸功能
BackgroundImage	設定或取得背景影像 *
CancelButton	設定或取得按下 ESC 鍵時，所按下的按鈕
ClientSize	設定或取得表單工作區（Client）的大小
ContextMenu	設定或取得與控制項關聯的快速鍵功能表 *
ControlBox	是否顯示表單控制項方塊（Control Box）
Cursor	設定或取得滑鼠游標 *
DesktopBounds	設定或取得桌面上表單的大小和位置
DesktopLocation	設定或取得桌面上表單的位置
Enabled	設定或取得是否可回應互動（Enable）*
Focused	判斷是否擁有輸入焦點 *
Font	設定或取得文字字型 *
ForeColor	設定或取得前景顏色 *
FormBorderStyle	設定或取得表單的框線（Border）樣式
Handle	取得視窗控制代碼 *
Height	設定或取得表單高度 *
HelpButton	是否顯示說明按鈕
Icon	設定或取得表單圖示
Left	設定或取得表單的 X 座標（單位爲像素）*
Location	設定或取得表單的左上角座標 *
MaximizeBox	是否顯示最大化按鈕
MaximumSize	取得表單所能調整的大小上限
Menu	設定或取得表單的選單功能表

屬性	說明
MinimizeBox	是否顯示最小化按鈕
MinimumSize	取得表單所能調整的大小下限
Name	設定或取得表單名稱
Right	設定或取得表單右界與左界的距離 *
ShowInTaskbar	表單是否顯示 Windows 工作列（Taskbar）中
Size	設定或取得表單大小
Text	設定或取得表單的標題
Top	設定或取得表單的 Y 座標（單位爲像素）*
TopLevel	表單是否爲最上層顯示
Visible	是否顯示表單 *
Width	設定或取得表單的寬度 *
WindowState	設定或取得表單的視窗狀態

*：繼承自 System.Windows.Forms.Control。

方法	說明
Activate	啓動表單
BringToFront	將表單上推至最頂層 *
Close	關閉表單
Dispose	釋放表單所使用的資源 *
DoDragDrop	開始拖放（Drag & Drop）*
Equals	繼承自 System.Object，判斷兩物件是否相等
Focus	設定表單輸入焦點 *
GetType	繼承自 System.Object，取得表單型別
Hide	隱藏表單 *
Refresh	強制失效並重繪表單 *
ResetBackColor	重設 BackColor 屬性值 *
ResetCursor	重設 Cursor 屬性值 *
ResetFont	重設 Font 屬性值 *
ResetForeColor	重設 ForeColor 屬性值 *
ResetText	重設 Text 屬性值 *
Scale	縮放表單 *
Select	啓動表單 *

方法	說明
SendToBack	將表單下推至最底層 *
SetAutoScrollMargin	繼承自 ScrollableControl，設定自動捲軸邊界大小
SetDesktopBounds	設定表單在桌面座標界限
SetDesktopLocation	設定表單在桌面座標位置
Show	顯示表單 *
ShowDialog	顯示表單爲強制回應對話盒
Update	重繪表單工作區域 *
Validate	繼承自 ContainerControl，驗證表單控制項

*: 繼承自 System.Windows.Forms.Control。

事件	說明（觸發事件時程）
Activated	當啓動表單時
Click	當按下表單時 *
Closed	當表單已關閉時
Closing	當表單正在關閉時
CursorChanged	當變更 Cursor 屬性值時 *
Deactivate	當表單失去焦點時
DoubleClick	當雙按表單時 *
DragDrop	當完成拖放（Drag & Drop）時 *
DragEnter	當物件拖曳進入表單邊框 *
DragLeave	當物件拖曳移出表單邊框 *
DragOver	當物件拖曳至表單上 *
EnabledChanged	當變更 Enabled 屬性值時 *
Enter	當輸入表單時 *
FontChanged	當變更 Font 屬性值時 *
ForeColorChanged	當變更 ForeColor 屬性值時 *
GotFocus	當取得表單焦點時 *
InputLanguageChanged	當已變更表單輸入語言時
InputLanguageChanging	當正在變更表單輸入語言時
KeyDown	當按下按鍵時 *
KeyPress	當按下按鍵時 *

事件	說明（觸發事件時程）
KeyUp	當放開按鍵時 *
Leave	當輸入焦點離開表單時 *
Load	當表單顯示前
LocationChanged	當變更 Location 屬性值時 *
LostFocus	當表單失去焦點時 *
MouseDown	當按下滑鼠按鍵時 *
MouseEnter	當滑鼠游標進入表單時 *
MouseHover	當滑鼠游標停留在表單上時 *
MouseLeave	當滑鼠游標離開表單時 *
MouseMove	當滑鼠游標移至表單上時 *
MouseUp	當放開滑鼠按鍵時 *
MouseWheel	當轉動滑鼠滾輪時 *
Move	當表單移動時 *
Paint	當表單重繪時 *
Resize	當表單重設大小時 *
SizeChanged	當變更 Size 屬性值時 *
TextChanged	當變更 Text 屬性值時 *
VisibleChanged	當變更 Visible 屬性值時 *

*：繼承自 System.Windows.Forms.Control。

以下程式片段爲介紹如何使用繼承 Form 類別產生一新的視窗，並以 Form 類別的 ShowDialog 方法顯示，請參考範例 A-2：

```
namespace FormApplication
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }

    private void Button1_Click(object sender, EventArgs e)
```

```
        {
          // 繼承 Form 類別產生新的視窗表單
          Form Form2 = new Form();

          Form2.Cursor = System.Windows.Forms.Cursors.Cross;
          Form2.FormBorderStyle = FormBorderStyle.Sizable;
          Form2.Height = 400;
          Form2.HelpButton = true;
          Form2.MaximizeBox = true;
          Form2.MinimizeBox = true;
          Form2.Name = "Form2";
          Form2.ShowInTaskbar = true;
          Form2.StartPosition = FormStartPosition.CenterParent;
          Form2.Text = "New Form";
          Form2.Width = 500;
          Form2.WindowState = FormWindowState.Normal;
          Form2.Enabled = true;

          // 以 Form 類別的 ShowDialog 方法顯示視窗表單
          Form2.ShowDialog();
        }
      }
    }
```

本範例利用 Form 類別的繼承產生一新的視窗表單物件 Form2：

```
Form Form2 = new Form();
```

表單物件 Form2 產生之後，接著設定其相關屬性：

- Cursor：滑鼠游標（Cursors.Cross）。
- Enabled：是否可回應互動（true）。
- FormBorderStyle：表單框線樣式（FormBorderStyle.Sizable）。
- Height：表單高度（400）。
- HelpButton：是否顯示說明按鈕（true）。
- MaximizeBox：是否顯示最大化按鈕（true）。

- MinimizeBox：是否顯示最小化按鈕（true）。
- Name：表單名稱（Form2）。
- ShowInTaskbar：是否顯示於 Windows 工作列中（true）。
- Text：標題（New Form）。
- Width：表單寬度（500）。
- WindowState：表單視窗狀態（FormWindowState.Normal）。

最後以 Show 或 ShowDialog 方法顯示視窗，兩者差別在於 Modal 與 Modaless，前者爲非獨佔強制回應，滑鼠仍可點選其他表單，後者爲獨佔強制回應，滑鼠僅能點選自己的表單：

```
public void Show()

public void Show(IWin32Window owner)

public DialogResult ShowDialog()

public DialogResult ShowDialog(IWin32Window owner)
```

其中參數 owner 爲父視窗（Owner Window）之 IWin32Window（Handle to a Window）。

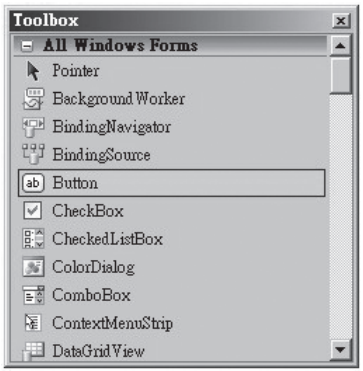
► 控制項

視窗使用者介面控制項（Control）包括以下常用控制項：

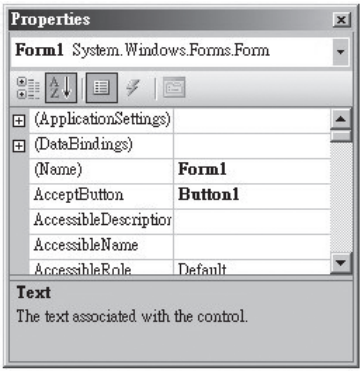
- 標籤：System.Windows.Forms.Label 類別。
- 文字方塊：System.Windows.Forms.TextBox 類別。
- 指令鈕：System.Windows.Forms.Button 類別。
- 選項按鈕：System.Windows.Forms.RadioButton 類別。
- 下拉式方塊：System.Windows.Forms.ComboBox 類別。
- 核取方塊：System.Windows.Forms.CheckBox 類別。
- 群組方塊：System.Windows.Forms.GroupBox 類別。
- 清單檢視：System.Windows.Forms.ListView 類別。
- 工具列：System.Windows.Forms.ToolBar 類別。

控制項類別皆繼承自 System.Windows.Forms.Control 類別，因此其屬性、方法及事件皆與 System.Windows.Forms.Form 類別類似。

在 Visual Studio 2008 中引用視窗使用者介面控制項，可直接自元件工具箱中拖曳控制項至表單上，接著由屬性視窗設定控制項屬性。



元件工具箱（Toolbox）



屬性視窗（Properties）

設定視窗控制項屬性之後，Visual Studio 2008 會自動產生相對應的程式碼，例如標籤、文字方塊、指令鈕，請參考範例 A-3 的 Form1.Designer.cs 內容：

```
namespace FormApplication
{
  partial class Form1
  {
```

```
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources
/// should be disposed; otherwise, false.</param>
protected override void Dispose(bool disposing)
{
  if (disposing && (components != null))
  {
    components.Dispose();
  }
  base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
  this.Button1 = new System.Windows.Forms.Button();
  this.TextBox1 = new System.Windows.Forms.TextBox();
  this.Label1 = new System.Windows.Forms.Label();
  this.SuspendLayout();
  //
  // Button1
  //
  this.Button1.Location = new System.Drawing.Point(77, 53);
  this.Button1.Name = "Button1";
  this.Button1.Size = new System.Drawing.Size(85, 32);
  this.Button1.TabIndex = 1;
```

```
    this.Button1.Text = "&OK";
    //
    // TextBox1
    //
    this.TextBox1.Location = new System.Drawing.Point(54, 13);
    this.TextBox1.Name = "TextBox1";
    this.TextBox1.Size = new System.Drawing.Size(168, 22);
    this.TextBox1.TabIndex = 0;
    //
    // Label1
    //
    this.Label1.Location = new System.Drawing.Point(16, 16);
    this.Label1.Name = "Label1";
    this.Label1.Size = new System.Drawing.Size(48, 16);
    this.Label1.TabIndex = 6;
    this.Label1.Text = "Name:";
    //
    // Form1
    //
    this.AcceptButton = this.Button1;
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 12F);
    this.AutoScaleMode =
      System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(238, 99);
    this.Controls.Add(this.Button1);
    this.Controls.Add(this.TextBox1);
    this.Controls.Add(this.Label1);
    this.FormBorderStyle =
      System.Windows.Forms.FormBorderStyle.FixedSingle;
    this.MaximizeBox = false;
    this.Name = "Form1";
    this.StartPosition =
      System.Windows.Forms.FormStartPosition.CenterScreen;
    this.Text = "Form1";
    this.ResumeLayout(false);
    this.PerformLayout();
  }
```

```
    #endregion

    internal System.Windows.Forms.Button Button1;
    internal System.Windows.Forms.TextBox TextBox1;
    internal System.Windows.Forms.Label Label1;
  }
}
```

► Label 類別

System.Windows.Forms.Label 類別用以顯示包含文字及圖像標籤，在執行階段，Label 類別無法被編輯修改，其建構函式爲：

```
public Label()
```

例如：

```
Label label1 = new Label();

label1.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D ;
label1.Location = New System.Drawing.Point(16, 16) ;
label1.Name = "Label1" ;
label1.Size = New System.Drawing.Size(48, 16) ;
label1.TabIndex = 0 ;
label1.Text = " Login:" ;
```

欲顯示圖像，可使用 Label 類別的 Image 或 ImageList、ImageIndex、ImageAlign 屬性，例如：

```
Label1.Image = ((System.Drawing.Bitmap)
    (resources.GetObject("Label1.Image"))) ;
```

或

```
Label1.ImageList = imageList1 ;
Label1.ImageIndex = 1 ;
Label1.ImageAlign = ContentAlignment.TopLeft ;
```

另外，爲處理超連結（Hyperlink），可使用 System.Windows.Forms.LinkLabel 類別，LinkLabel 類別繼承自 Label 類別，因此 Label 類別所提供的屬性、事件及方法皆可繼續延用，而關於處理超連結部份，LinkLabel 類別提供：

屬性	說明
ActiveLinkColor	設定或取得作用中（Active）超連結的色彩
DisabledLinkColor	設定或取得停用中（Disabled）超連結的色彩
LinkArea	設定或取得超連結文字範圍
LinkBehavior	設定或取得超連結動作
LinkColor	設定或取得超連結的色彩
Links	LinkLabel 類別的超連結集合
LinkVisited	是否將超連結顯示爲已瀏覽

事件	說明（觸發事件時程）
LinkClicked	當點選超連結時

請參考範例 A-4。

```
namespace LabelApplication
{
  public partial class Form1 : Form
  {
    ...
    private void Form1_Load(object sender, EventArgs e)
    {
      LinkLabel1.Links.Add(0, 16,
        "http://www.microsoft.com/Taiwan");

      LinkLabel1.LinkClicked += new
        LinkLabelLinkClickedEventHandler(
        this.linkLabel1_LinkClicked);
    }

    private void linkLabel1_LinkClicked(object sender,
        System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
    {
```

```
        LinkLabel1.Links[LinkLabel1.Links.IndexOf(e.Link)].Visited
          = true;

        System.Diagnostics.Process.Start(
          e.Link.LinkData.ToString());
      }
    }
  }
```

► TextBox 類別

System.Windows.Forms.TextBox 類別用以顯示、編輯及輸入文字的文字方塊。前面所提到的 Label 與 LinkLabel 類別並不提供編輯的功能，而 TextBox 類別則允許顯示、編輯及輸入文字之用。其建構函式爲：

```
public TextBox()
```

例如：

```
TextBox TextBox1 = new TextBox() ;
TextBox1.Location = new System.Drawing.Point(84, 12) ;
TextBox1.Size = new System.Drawing.Size(136, 22) ;
TextBox1.Text = "Normal Text" ;
```

另外，TextBox 類別支援密碼遮罩字元（Password-Masked Character）及多行文字輸入，其屬性分別爲：

1. PasswordChar：設定或取得 TextBox 類別的密碼遮罩字元，例如：

```
TextBox TextBox1 = new TextBox() ;
TextBox1.PasswordChar = "*" ;
```

2. Multiline：設定或取得 TextBox 類別是否可多行文字輸入，例如：

```
TextBox TextBox1 = new TextBox() ;
TextBox1.Multiline = true ;
```

3. ScrollBars：設定或取得 TextBox 類別是否顯示水平或垂直捲軸，其中 ScrollBars 列舉值有：
   - ScrollBars.Both：同時顯示水平及垂直捲軸。
   - ScrollBars.Horizontal：只顯示水平捲軸。
   - ScrollBars.None：不顯示捲軸。
   - ScrollBars.Vertical：只顯示垂直捲軸。

```
TextBox TextBox1 = new TextBox() ;
TextBox1.ScrollBars = System.Windows.Forms.ScrollBars.Both ;
```

請參考範例 A-5。

► Button 類別

System.Windows.Forms.Button 類別用以產生包含文字或圖形的指令鈕。其建構函式爲：

```
public Button()
```

Button 類別常見之屬性、方法及事件爲：

屬性	說明
Anchor	設定或取得 Button 類別附著於容器的邊緣
Cursor	設定或取得滑鼠游標 *
DialogResult	設定或取得按下 Button 類別時之回傳值
Enabled	設定或取得是否可回應互動 *
Focused	判斷是否擁有輸入焦點 *
Handle	取得 Button 類別之控制代碼 *
Image	設定或取得 Button 類別之圖示
ImageAlign	設定或取得 Button 類別之圖示的對齊方式
ImageIndex	設定或取得 Button 類別之圖示的影像清單索引值
ImageList	設定或取得 Button 類別之圖示的 ImageList
Name	設定或取得 Button 類別之名稱
Text	設定或取得 Button 類別之文字
Visible	是否顯示 Button 類別 *

*：繼承自 System.Windows.Forms.Control。

方法	說明
Dispose	繼承自 System.ComponentModel.Component，釋放控制項所使用的資源
Equals	繼承自 System.Object，判斷兩物件是否相等
Focus	設定 Button 類別之輸入焦點 *
Hide	隱藏控制項 *
PerformClick	產生 Button 類別的 Click 事件
Refresh	強制失效並重繪 Button 類別 *
Show	顯示 Button 類別 *
Update	重繪 Button 類別之工作區域 *

*: 繼承自 System.Windows.Forms.Control。

事件	說明（觸發事件時程）
Click	當按下 Button 類別時 *
CursorChanged	當變更 Cursor 屬性值時 *
DragDrop	當完成拖放（Drag & Drop）時 *
DragEnter	當物件拖曳進入 Button 類別之邊框 *
DragLeave	當物件拖曳移出 Button 類別之邊框 *
DragOver	當物件拖曳至 Button 類別上 *
Enter	當輸入 Button 類別時 *
KeyDown	當按下按鍵時 *
KeyPress	當按下按鍵時 *
KeyUp	當放開按鍵時 *
Leave	當輸入焦點離開 Button 類別時 *
LostFocus	當 Button 類別失去焦點時 *
MouseDown	當按下滑鼠按鍵時 *
MouseEnter	當滑鼠游標進入 Button 類別時 *
MouseHover	當滑鼠游標停留在 Button 類別上時 *
MouseLeave	當滑鼠游標離開 Button 類別時 *
MouseMove	當滑鼠游標移至 Button 控制項上時 *
MouseUp	當放開滑鼠按鍵時 *
Resize	當 Button 控制項重設大小時 *
TextChanged	當變更 Text 屬性值時 *

*: 繼承自 System.Windows.Forms.Control。

其中 DialogResult 屬性爲以下之列舉值：

- Windows.Forms.DialogResult.Abort
- Windows.Forms.DialogResult.Cancel
- Windows.Forms.DialogResult.Ignore
- Windows.Forms.DialogResult.No
- Windows.Forms.DialogResult.None
- Windows.Forms.DialogResult.OK
- Windows.Forms.DialogResult.Retry
- Windows.Forms.DialogResult.Yes

以下之程式片段介紹如何繼承 Button 類別產生一新的指令鈕：

```
Button Button1 = new Button() ;
Button1.Location = new System.Drawing.Point(40, 48) ;
Button1.Name = "Button1" ;
Button1.Size = new System.Drawing.Size(108, 28) ;
Button1.Text = "Button1" ;
```

請參考範例 A-6。

```
private void Button1_Click(object sender, EventArgs e)
{
  TextBox1.Text = "Button1 Click";
}

private void Button1_MouseDown(object sender, MouseEventArgs e)
{
  TextBox1.Text = "Button1 Mouse Down";
}

private void Button1_MouseHover(object sender, EventArgs e)
{
  TextBox1.Text = "Button1 Mouse Hover";
}
```

Button 類別常用的事件如 Click、MouseEnter、MouseHover、MouseLeave 等，其事件的委派均爲 System.EventHandler，委派是物件呼叫其它物件方法的行爲，程式可利用委派指定事件處理的方法，其宣告方式爲：

```
public delegate void EventHandler(Object sender, EventArgs e)
```

參數 e 爲 System.EventArgs 類別形式。

此外，Button 類別尚有 MouseClick、MouseDoubleClick、MouseDown、MouseMove、MouseUp、MouseWheel 等事件，其事件的委派均爲 System.Windows.Forms.MouseEventHandler，其宣告方式爲：

```
public delegate void MouseEventHandler(Object sender, MouseEventArgs e)
```

參數 e 爲 System.Windows.Forms.MouseEventArgs 類別形式，可使用 MouseEventArgs 類別以下之屬性，瞭解滑鼠事件之狀態或所按下之滑鼠按鍵：

- Button：取得按下滑鼠第幾個按鍵，爲 MouseButtons 之列舉值，如 MouseButtons.Left（滑鼠左鍵）、MouseButtons.Middle（滑鼠中間鍵）、MouseButtons.None（未按任何鍵）、MouseButtons.Right（滑鼠右鍵）等。
- Clicks：取得按下並釋放滑鼠按鍵的次數。
- Delta：取得滾動滾輪滑鼠時，其滾動的 Unit 單位數目。
- Location：取得在產生滑鼠事件時，滑鼠相對於物件左上角的位置，並回傳 System.Drawing.Point 結構型別，分別代表滑鼠 X（水平）與 Y（垂直）座標，可藉由 Point.X 與 Point.Y 屬性取得此座標值。
- X：取得在產生滑鼠事件時，滑鼠相對於物件左上角的 X（水平）座標值。
- Y：取得在產生滑鼠事件時，滑鼠相對於物件左上角的 Y（垂直）座標值。

滑鼠事件依下列順序發生：

1. MouseEnter
2. MouseMove
3. MouseHover/MouseDown/MouseWheel
4. MouseUp
5. MouseLeave

► 通用對話盒

System.Windows.Forms.CommonDialog 類別用以處理通用對話盒（Common Dialog），常見的通用對話盒有：

- 開啓檔案：System.Windows.Forms.OpenFileDialog 類別。
- 儲存檔案：System.Windows.Forms.SaveFileDialog 類別。
- 色彩：System.Windows.Forms.ColorDialog 類別。
- 字型：System.Windows.Forms.FontDialog 類別。
- 列印之版面設定：System.Windows.Forms.PageSetupDialog 類別。
- 預覽列印：System.Windows.Forms.PrintPreviewDialog 類別。
- 列印：System.Windows.Forms.PrintDialog 類別。

上述對話盒類別皆繼承自 System.Windows.Forms.CommonDialog，因此其屬性、方法及事件皆雷同，並以 ShowDialog 方法顯示對話盒：

```
public DialogResult ShowDialog()

public DialogResult ShowDialog(IWin32Window owner)
```

其中參數 owner 爲父視窗（Owner Window）之 IWin32Window（Handle to a Window）。

ShowDialog 方法回傳以下 DialogResult 之列舉值：

- Windows.Forms.DialogResult.Abort
- Windows.Forms.DialogResult.Cancel
- Windows.Forms.DialogResult.Ignore
- Windows.Forms.DialogResult.No
- Windows.Forms.DialogResult.None
- Windows.Forms.DialogResult.OK
- Windows.Forms.DialogResult.Retry
- Windows.Forms.DialogResult.Yes

► 開啓檔案對話盒

System.Windows.Forms.OpenFileDialog 類別用以處理開啓檔案對話盒，其常用之屬性、方法及事件如下：

屬性	說明
AddExtension	是否自動加入檔案的副檔名
CheckFileExists	是否顯示指定檔案不存在的警告訊息
CheckPathExists	是否顯示指定路徑不存在的警告訊息
DefaultExt	設定或取得預設的副檔名
DereferenceLinks	回傳捷徑所參照的檔案位置或捷徑的位置
FileName	設定或取得開啓檔案對話盒所選取檔案的名稱
FileNames	取得開啓檔案對話盒所有選取檔案的名稱
Filter	設定或取得檔案名稱篩選字串
FilterIndex	設定或取得檔案名稱的篩選條件索引
InitialDirectory	設定或取得檔案對話盒的初始目錄
Multiselect	是否允許選取多個檔案
ReadOnlyChecked	是否選取「以唯讀方式開啓」之核取方塊
RestoreDirectory	是否在關閉檔案對話盒前之還原目錄
ShowHelp	是否顯示說明按鈕
ShowReadOnly	是否顯示「以唯讀方式開啓」之核取方塊
Title	設定或取得開啓檔案對話盒的標題
ValidateNames	是否僅接受 Win32 檔案名稱

方法	說明
OpenFile	開啓所選取的檔案
ShowDialog	顯示開啓檔案對話盒

事件	說明（觸發事件時程）
FileOk	繼承自 System.Windows.Forms.FileDialog，當按下檔案對話盒的開啓或儲存按鈕時
HelpRequest	繼承自 System.Windows.Forms.CommonDialog，當按下對話盒的說明按鈕時

請參考範例 A-7。

```
StreamReader sr = null;
FileStream fs = null;

OpenFileDialog1.InitialDirectory = "C:\\";
OpenFileDialog1.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
OpenFileDialog1.FilterIndex = 2;
OpenFileDialog1.RestoreDirectory = true;
OpenFileDialog1.ShowHelp = true;
OpenFileDialog1.ShowReadOnly = true;
OpenFileDialog1.Title = "Open File Dialog";

if (OpenFileDialog1.ShowDialog() == DialogResult.OK)
{
  TextBox1.Text = OpenFileDialog1.FileName;
  TextBox2.Text = "";

  if (OpenFileDialog1.ReadOnlyChecked)
    chkReadOnly.CheckState = CheckState.Checked;
  else
    chkReadOnly.CheckState = CheckState.Unchecked;

  try
  {
    fs = new FileStream(OpenFileDialog1.FileName,
      FileMode.Open, FileAccess.Read);
    sr = new StreamReader(fs, System.Text.Encoding.Default);

    TextBox2.Text = sr.ReadToEnd();
  }
  catch (Exception ex)
  {
    ...
  }
  finally
  {
      if (sr != null)
        sr.Close();
```

```
        if (fs != null)
          fs.Close();
    }
  }
```

本範例除了介紹以 OpenFileDialog 類別開啓檔案對話盒取得所選取檔案的名稱之外，並示範以 System.IO.FileStream 及 System.IO.StreamReader 類別取得所選檔案的內容。

► 儲存檔案對話盒

System.Windows.Forms.SaveFileDialog 類別用以處理儲存檔案對話盒，請參考 OpenFileDialog 類別之屬性、方法及事件說明。

請參考範例 A-8。

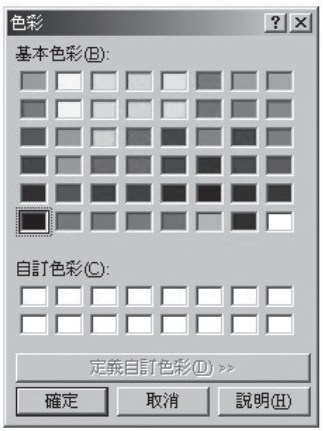
```
StreamWriter sw = null;
FileStream fs = null;

SaveFileDialog1.InitialDirectory = "C:\\";
SaveFileDialog1.Filter =
  "Text files (*.txt)|*.txt|All files (*.*)|*.*";
SaveFileDialog1.FilterIndex = 1;
SaveFileDialog1.RestoreDirectory = true;
SaveFileDialog1.ShowHelp = true;
SaveFileDialog1.Title = "Save File Dialog";

if (SaveFileDialog1.ShowDialog() == DialogResult.OK)
{
  TextBox1.Text = SaveFileDialog1.FileName;

  try
  {
    fs = new FileStream(SaveFileDialog1.FileName,
      FileMode.Create, FileAccess.Write);
    sw = new StreamWriter(fs, System.Text.Encoding.Default);
```

```
      sw.Write(TextBox2.Text);
    }
    catch (Exception ex)
    {
      ...
    }
    finally
    {
      if (sw != null)
        sw.Close();

      if (fs != null)
        fs.Close();

      ...
    }
  }
```

本範例除了介紹以 SaveFileDialog 類別儲存檔案對話盒儲存檔案之外，並示範以 System.IO.FileStream 及 System.IO.StreamWriter 類別將資料內容儲存至所選檔案中。

► 色彩對話盒

System.Windows.Forms.ColorDialog 類別用以處理色彩對話盒。



色彩對話盒

其常用之屬性、方法及事件如下：

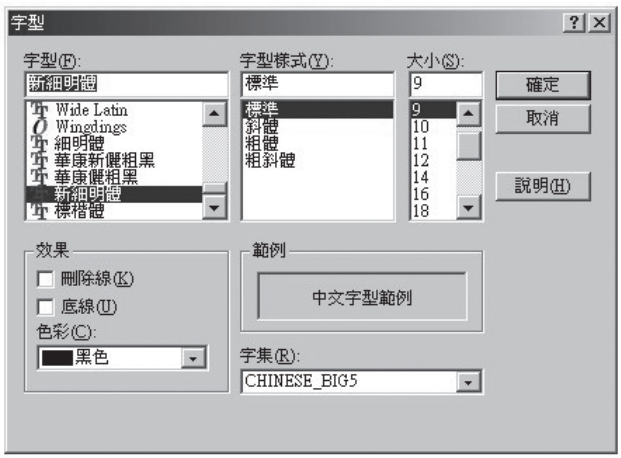
屬性	說明
AllowFullOpen	是否使用色彩對話盒自訂色彩，預設爲 true
AnyColor	是否顯示所有可用色彩，預設爲 false
Color	設定或取得所選取的色彩，預設爲 Color.Black
CustomColors	顯示自訂色彩集，預設爲 null
FullOpen	是否顯示自訂色彩控制項，預設爲 false
ShowHelp	是否顯示說明按鈕，預設爲 false
SolidColorOnly	設定或取得只能選取純色（Solid Color），預設爲 false

方法	說明
ShowDialog	顯示色彩對話盒

事件	說明（觸發事件時程）
HelpRequest	繼承自 System.Windows.Forms.CommonDialog，當按下色彩對話盒的說明按鈕時

► 字型對話盒

System.Windows.Forms.FontDialog 類別用以處理字型對話盒。



字型對話盒

其常用之屬性、方法及事件如下：

屬性	說明
AllowSimulations	是否允許模擬繪圖裝置介面（GDI）字型，預設爲 true
AllowVectorFonts	是否允許選取向量字型，預設爲 true
AllowVerticalFonts	是否顯示垂直字型，預設爲 true
Color	設定或取得字型色彩，預設爲 Color.Black
FixedPitchOnly	是否只允許選取固定字幅（Fixed-Pitch）的字型，預設爲 false
Font	設定或取得所選取的字型
FontMustExist	是否顯示指定字型樣式不存在的警告訊息
MaxSize	設定或取得可選取字型的最大點數，預設爲 0
MinSize	設定或取得可選取字型的最小點數，預設爲 0
ScriptsOnly	是否允許選取非 OEM、Symbol 及 ANSI 的字型，預設爲 false
ShowApply	是否顯示套用（Apply）按鈕，預設爲 false
ShowColor	是否顯示色彩選擇，預設爲 false
ShowEffects	是否顯示刪除線、底線等控制項，預設爲 true
ShowHelp	是否顯示說明按鈕，預設爲 false

方法	說明
ShowDialog	顯示字型對話盒

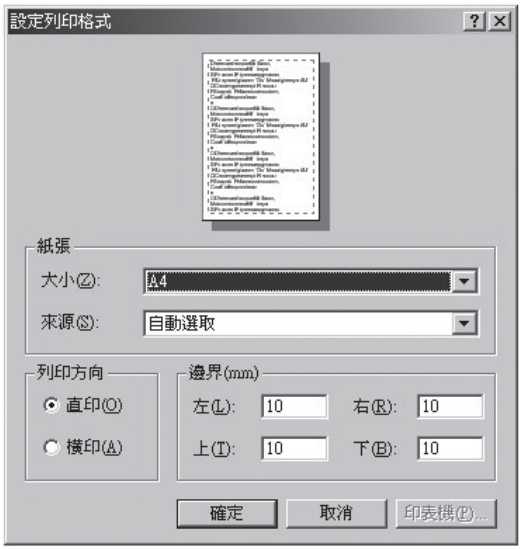
事件	說明（觸發事件時程）
Apply	當按下字型對話盒的套用（Apply）按鈕時
HelpRequest	繼承自 System.Windows.Forms.CommonDialog，當按下字型對話盒的說明按鈕時

請參考範例 A-9。

```
switch (ToolBar1.Buttons.IndexOf(e.Button))
{
  case 0:
    // Show Color Dialog
    ColorDialog1.AllowFullOpen = false;
    ColorDialog1.ShowHelp = true;
    ColorDialog1.Color = TextBox1.ForeColor;

    if (ColorDialog1.ShowDialog() == DialogResult.OK)
      TextBox1.ForeColor = ColorDialog1.Color;

    break;

  case 1:
    // Show Font Dialog
    FontDialog1.Font = TextBox1.Font;
    FontDialog1.ShowHelp = true;
    FontDialog1.Color = TextBox1.ForeColor;
    FontDialog1.ShowColor = true;

    if (FontDialog1.ShowDialog() == DialogResult.OK)
    {
      TextBox1.Font = FontDialog1.Font;
      TextBox1.ForeColor = FontDialog1.Color;
    }

    break;
}
```

► 列印之版面設定對話盒

System.Windows.Forms.PageSetupDialog 類別用以處理列印之版面設定對話盒。



列印之版面設定對話盒

其常用之屬性、方法及事件如下：

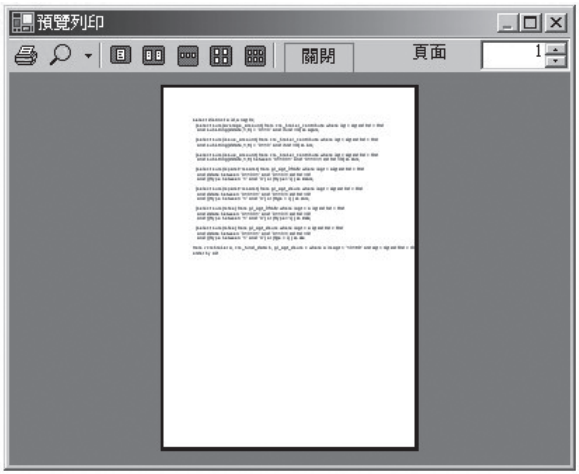
屬性	說明
AllowMargins	是否啓用版面設定之邊界設定，預設爲 true
AllowOrientation	是否啓用版面設定之列印方向設定(直印、橫印)，預設爲 true
AllowPaper	是否啓用版面設定之紙張設定(大小、來源)，預設爲 true
AllowPrinter	是否啓用印表機按鈕，預設爲 true
Document	設定或取得 PrintDocument，表示 PrintDocument 從何處取得版面設定，預設爲 null
MinMargins	設定或取得最小邊界(1/100 英吋)，預設爲 null
PageSettings	頁面設定，預設爲 null
PrinterSettings	設定或取得修改的印表機設定，預設爲 null
ShowHelp	是否顯示說明按鈕，預設爲 false
ShowNetwork	是否顯示網路按鈕，預設爲 true

方法	說明
ShowDialog	顯示版面設定對話盒

事件	說明（觸發事件時程）
HelpRequest	繼承自 System.Windows.Forms.CommonDialog，當按下版面設定對話盒的說明按鈕時

► 預覽列印對話盒

System.Windows.Forms.PrintPreviewDialog 類別用以處理預覽列印對話盒。



預覽列印對話盒

其常用之屬性、方法及事件如下：

屬性	說明
DialogResult	設定或取得按下 Button 類別時回傳的值
Document	設定 PrintDocument 預覽文件

方法	說明
ShowDialog	顯示預覽列印對話盒

事件	說明（觸發事件時程）
HelpRequest	繼承自 System.Windows.Forms.CommonDialog，當按下預覽列印對話盒的說明按鈕時

► 列印對話盒

System.Windows.Forms.PrintDialog 類別用以處理列印對話盒。

列印對話盒

其常用之屬性、方法及事件如下：

屬性	說明
AllowPrintToFile	是否啓用「列印到檔案」核取方塊，預設爲 true
AllowSelection	是否啓用列印範圍之選擇範圍選項，預設爲 false
AllowSomePages	是否啓用列印範圍之「從 ... 到 ... 頁面」選項，預設爲 false
Document	設定 PrintDocument，預設爲 null
PrinterSettings	設定或取得修改的印表機設定，預設爲 false
PrintToFile	是否核取「列印到檔案」，預設爲 null
ShowHelp	是否顯示說明按鈕，預設爲 false
ShowNetwork	是否顯示網路按鈕，預設爲 true

方法	說明
ShowDialog	顯示列印對話盒

事件	說明（觸發事件時程）
HelpRequest	繼承自 System.Windows.Forms.CommonDialog，當按下列印對話盒的說明按鈕時

請參考範例 A-10。

► 訊息對話盒

在 Microsoft Visual Studio 6.0 中，訊息對話盒（Message Box）是以 MsgBox 或 MessageBox API 顯示。在 Visual Studio 2008 中，則使用 System.Windows.Forms.MessageBox 類別。

MessageBox 類別提供顯示及取得資訊的訊息對話盒，並以 Show 方法顯示訊息對話盒，並回傳以下 DialogResult 屬性之列舉值：

- Windows.Forms.DialogResult.Abort
- Windows.Forms.DialogResult.Cancel
- Windows.Forms.DialogResult.Ignore
- Windows.Forms.DialogResult.No
- Windows.Forms.DialogResult.None
- Windows.Forms.DialogResult.OK
- Windows.Forms.DialogResult.Retry
- Windows.Forms.DialogResult.Yes

Show 方法共有：

```
public static DialogResult Show(string text)
```

僅顯示文字訊息，並預設顯示確定（OK）按鈕且不顯示標題，參數 text 爲指定之訊息，例如：


```
public static DialogResult Show(string text, string caption)
```

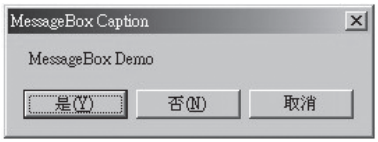
顯示文字訊息及標題，並預設顯示確定（OK）按鈕的訊息對話盒，參數 caption 爲訊息對話盒之標題，例如：

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons)
```

顯示指定文字訊息、標題及指定按鈕的訊息對話盒，參數 buttons 爲以下 MessageBoxButtons 之列舉值：

- MessageBoxButtons.AbortRetryIgnore：顯示中止（Abort）、重試（Retry）及忽略（Ignore）按鈕。
- MessageBoxButtons.OK：顯示確定（OK）按鈕。
- MessageBoxButtons.OKCancel：顯示確定（OK）及取消（Cancel）按鈕。
- MessageBoxButtons.RetryCancel：顯示重試（Retry）及取消（Cancel）按鈕。
- MessageBoxButtons.YesNo：顯示是（Yes）及否（No）按鈕。
- MessageBoxButtons.YesNoCancel：顯示是（Yes）、否（No）及取消（Cancel）按鈕。

例如：




```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon)
```

顯示指定文字訊息、標題、指定按鈕及圖示的訊息對話盒，參數 icon 爲以下 MessageBoxIcon 之列舉值及圖示：

列舉	說明	圖示
MessageBoxIcon.Asterisk	顯示小寫 i 圖示	
MessageBoxIcon.Error	顯示錯誤圖示	
MessageBoxIcon.Exclamation	顯示驚嘆號圖示	
MessageBoxIcon.Hand	顯示錯誤圖示	
MessageBoxIcon.Information	顯示小寫 i 圖示	

列舉	說明	圖示
MessageBoxIcon.None	不顯示圖示	
MessageBoxIcon.Question	顯示問號圖示	
MessageBoxIcon.Stop	顯示錯誤圖示	
MessageBoxIcon.Warning	顯示驚嘆號圖示	

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton)
```

顯示指定文字訊息、標題、指定按鈕、圖示及預設按鈕的訊息對話盒，參數 defaultButton 爲以下 MessageBoxDefaultButton 之列舉值：

- MessageBoxDefaultButton.Button1：第一個按鈕爲預設按鈕。
- MessageBoxDefaultButton.Button2：第二個按鈕爲預設按鈕。
- MessageBoxDefaultButton.Button3：第三個按鈕爲預設按鈕。

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options)
```

顯示指定文字訊息、標題、指定按鈕、圖示、預設按鈕及選項的訊息對話盒，參數 options 爲以下 MessageBoxOptions 之列舉值：

- MessageBoxOptions.DefaultDesktopOnly：顯示訊息對話盒於預設桌面上。
- MessageBoxOptions.RightAlign：訊息對話盒靠右對齊。
- MessageBoxOptions.RtlReading：訊息對話盒由右至左顯示指定訊息。
- MessageBoxOptions.ServiceNotification：即使使用者沒有登入，仍會在目前作用中的桌面上顯示訊息對話盒。

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options, bool displayHelpButton)
```

參數 displayHelpButton 設定是否顯示【說明】按鈕。

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options, string helpFilePath)
```

參數 helpFilePath 設定當使用者按下【說明】按鈕時，所顯示之說明檔的路徑和名稱。

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options, string helpFilePath,
        HelpNavigator navigator)
```

參數 navigator 爲以下 HelpNavigator 之列舉值：

- HelpNavigator.AssociateIndex：說明檔將開啓指定主題的第一個字母之索引項目。
- HelpNavigator.Find：說明檔將開啓搜尋頁面。
- HelpNavigator.Index：說明檔將開啓索引。
- HelpNavigator.KeywordIndex：開啓含指定索引項目的主題說明檔，否則顯示最接近指定關鍵字的索引項目。
- HelpNavigator.TableOfContents：說明檔將開啓目錄。
- HelpNavigator.Topic：說明檔將開啓指定的主題。
- HelpNavigator.TopicId：說明檔將開啓數字主題識別項所表示的主題。

```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options, string helpFilePath,
        string keyword)
```

參數 keyword 爲當使用者按下【說明】按鈕時，所顯示之說明關鍵字。

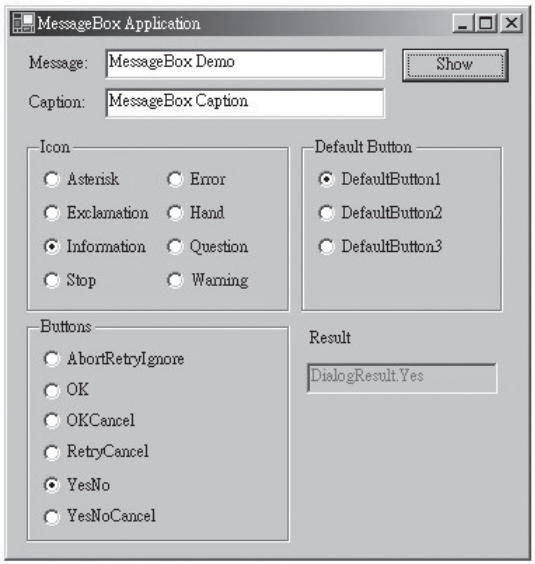
```
public static DialogResult Show(string text, string caption,
        MessageBoxButtons buttons, MessageBoxIcon icon,
        MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options, string helpFilePath,
        HelpNavigator navigator, Object param)
```

參數 param 爲當使用者按下【說明】按鈕時，所顯示之說明主題的 ID。

除了以上的方法之外，亦可加入 IWin32Window 參數代表父視窗（Owner Window）之 IWin32Window（Handle to a Window），例如：

```
public static DialogResult Show(IWin32Window owner, string text,
        string caption, MessageBoxButtons buttons,
        MessageBoxIcon icon, MessageBoxDefaultButton defaultButton,
        MessageBoxOptions options, string helpFilePath,
        HelpNavigator navigator, Object param)
```

請參考範例 A-11。



❋【參考資料】

[1] Microsoft Developer Network.