الجزء الثالث

تطوير تطبيقات
Windows

PDF

نماذج Windows Forms

ان كنت تنوي تطوير تطبيقات تحاكي تطبيقات Windows القياسية، فعليك إظهار مخرجاتك على نوافذ عوضا عن الكائن ArabicConsole، حيث ستحتاج إلى استخدام مجموعة كبيرة من الفئات مشمولة في مجال الأسماء System.Windows.Forms.

يحتوي الجزء الثالث **تطوير تطبيقات Windows** من هذا الكتاب على أربعة فصول تختص بتطوير مشاريع من نوع Windows Application. الأول مدخلك لاستخدام نماذج Windows Forms، الثاني يتمحور حول الأدوات Controls التي تحضنها النماذج، الثالث موجه لاستخدام تقنية GDI+، اما الرابع فيلقي الضوء على مجموعة من المواضيع المتفرقة والتي يتعامل معها مطورو تطبيقات Windows بشكل مكثف.

ملاحظة

إن ما زلت مستمر على المشاريع من النوع Console Application، فعليك استيراد مجال الاسماء التالي:

```
Imports System.Windows.Forms
```

اما ان كنت متابع لهذا الفصل، فلست بحاجة لاستيراده حيث ان المشاريع من نوع Windows Application تقوم باستيراده بشكل تلقائي في خانة التبويب Imports من صندوق الحوار Project Property Pages (**شكل** 2-7).

مدخلك إلى نماذج Windows Forms

في .NET. Visual Basic، نافذة النموذج ما هي إلا فئة تقليدية مشتقة وراثيا من الفئة System.Windows.Forms.Form تصرحها في اي مكان من البرنامج كما تصرح الفئات الأخرى:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    …
    …
End Class
```

بعد تصريحك لهذه الفئة، يمكنك انشاء كائن منها واستدعاء الطريقة Show() لإظهار النافذة:

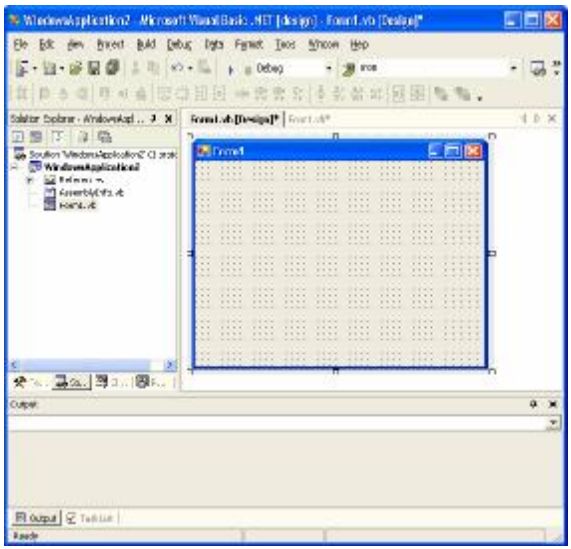
```
Dim myForm As New Form1

myForm.Show ()
```

مع ذلك، تتطلب فئات النماذج مجموعة إضافية من الشيفرات المصدرية عليك اضافتها بين فكي التركيب Class … End Class قبل إنشاء الكائنات منها. مع انه يمكنك كتابة هذه الشيفرات يدويا بنفسك، إلا أني لا أجد سبب مقنعا يمنعك من استخدام مصمم النماذج Form Designer.

مصمم النماذج Form Designer

يصنف مصمم النماذج من برامج **مولدة الشيفرات Code Generator**، حيث يقوم بتوليد الشيفرات التي توافق التصاميم التي تنجزها بنقرات بسيطة بالفأرة (**شكل 13-1**). يمكن للملف الواحد ان يحتوي على أكثر من فئة نموذج Form Class، ولكن الشيفرات المولدة من مصمم النماذج ستوجه إلى الفئة المسطورة في أعلى الملف فقط.

PDF



شكل 13-1: مصمم النماذج Form Designer.

ملاحظة

لا يشترط ان تكون فئة النموذج في أعلى الملف فقط لاستخدام مصمم النماذج، بل حتى لا يمكن للفئة ان تكون محضونة في فئة اخرى أو وحدة برمجية Module.
نستنتج من هذه الملاحظة أيضا، ان محدد الوصول Private لا يمكن استخدامه مع فئات النماذج لاستخدام مصمم النماذج.

أنشئ مشروع جديد من نوع Windows Application، ستلاحظ ان بيئة التطوير Visual Studio .NET قد أنشأت ملفات المشروع وأظهرت لك نافذة خالية بعنوان Form1. النقر المزدوج على هذه النافذة يفتح لك نافذة محرر الشيفرة Code Editor، سترى ان مصمم النماذج قد ولد الشيفرة التالية بشكل ابتدائي (تعمدت ترجمة التعليقات إلى اللغة العربية):

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        ' هذا الاستعداء مطلوب من مصمم النماذج.
        InitializeComponent()

        ' اضف أي شيفرات اضافية لهذا المشيد بعد هذا السطر

    End Sub

    ' مهدم الفئة.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' متغير خاص بمصمم النماذج
    Private components As System.ComponentModel.IContainer

    ' الاجراء التالي خاص بمصمم النماذج
    ' يمكنك تعديل محتوياته من خلال نافذة المصمم فقط
    ' ولا تحاول تعديل الشيفرة يدويا
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 269)
        Me.Name = "Form1"
        Me.Text = "Form1"

    End Sub

#End Region
End Class
```

قم بقنص الحدث Click (الموجود في الفئة القاعدية Base Class) والذي يتم تفجيره لحظة نقر المستخدم على النافذة، وأضف شيفرة تبين لنا ردة الفعل. عرفت هنا إجراء باسم formWasClicked لقنص الحدث Click يعرض رسالة ترحيبية باستخدام الدالة MsgBox:

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    …
    …
    Private Sub formWasClicked(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles MyBase.Click

        MsgBox("Windows Form مرحبا بك في")
    End Sub
End Class
```

انظر أيضا

لمزيد من التفاصيل حول الأحداث وقنصها، راجع الفصل الثالث **الفئات والكائنات**.

بعد تنفيذك للبرنامج ستظهر لك نافذة النموذج، وان قمت بالنقر عليها (بزر الفأرة الأيسر أو الأيمن) في أي مكان، ستظهر لك الرسالة الترحيبية (**شكل 13-2**):

شكل 13-2: الرسالة الترحيبية ظهرت نتيجة للنقر على النافذة.

نظرة حول الشيفرة المولدة

استيعابك للشيفرة التي ولدها مصمم النماذج يعتمد اعتماد كلي على استيعابك للفصول الخمس الاولى من هذا الكتاب، فلايوجد شيء جديد بها سوى الشيفرات المتعلقة بالمتغير components والخاص بمصمم النماذج فقط وليس له أي تأثير كبير في الكائن المنشئ من هذه الفئة لحظة التنفيذ.

فـــي أعلـــى الـــشيفرة عرفنـــا فئـــة باســـم Form1 مـــشتقة وراثيـــا مـــن الفئـــة System.Windows.Forms.Form -وهي احد متطلبات فئات النماذج:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    …
    …
End Class
```

بالنسبة لمـــشيد الفئـــة، فعليـــه اســـتدعاء مـــشيد الفئـــة القاعديـــة واســـتدعاء الإجـــراء InitializeComponent() والخاص بمصمم النماذج اولا، ومن ثم يمكنك اضافة أي شيفرات تود تنفيذها لحظة إنشاء نسخة جديدة من كائن:

```
Public Class Form1
    …
    …
    Public Sub New()
        MyBase.New()

        ' هذا الاستعداء مطلوب من مصمم النماذج.
        InitializeComponent()

        ' اضف أي شيفرات اضافية لهذا المشيد بعد هذا السطر
    End Sub
    …
    …
End Class
```

بعد ذلك، نقوم بإعادة تعريف Overloads وإعادة قيادة Overrides الطريقة Dispose() من الفئة القاعدية، ليكون المهدم Destructor الخاص بالفئة (من الضروري استدعاء مهدم الفئة القاعدية):

```
Public Class Form1
    …
    Protected Overloads Overrides Sub Dispose(ByVal _
        disposing As Boolean)

        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    …
End Class
```

اخيرا، الإجراء InitializeComponent()، وهو خاص بمصمم النماذج، حيث يتم فيه توليد الشيفرة الناتجة عن التعديلات التي تفعلها بمصمم النماذج (كتغيير الخصائص، إضافة الأدوات، وغيرها) وقت التصميم.

```
Public Class Form1
    …
    …
    ' الإجراء التالي خاص بمصمم النماذج
    ' يمكنك تعديل محتوياته من خلال نافذة المصمم فقط
    ' ولا تحاول تعديل الشيفرة يدويا
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 269)
        Me.Name = "Form1"
        Me.Text = "Form1"
    End Sub

End Class
```

ملاحظة

لا تحاول تعديل الشيفرة المصدرية والمولدة في الإجراء InitializeComponent يدويا بنفسك، وإنما اعتمد على مصمم النماذج فهو يقوم باللازم نيابة عنك بدقة أكثر.

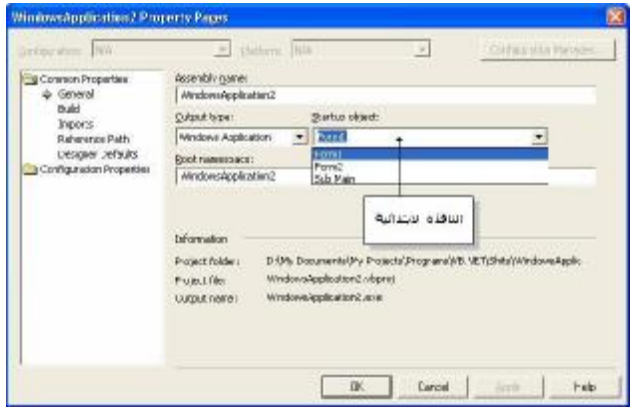
لست مضطر لاضافة جميع الشيفرات السابقة في كل مرة تعرف فيها فئة نموذج جديدة، اذ يمكنك حذف الشيفرات الخاصة بمصمم النماذج وتكتب الشيفرات الضرورية لهذه الفئات فقط (انصحك بشدة بعدم عمل ذلك):

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        MyBase.New()
    End Sub
```

```
    Protected Overloads Overrides Sub Dispose(ByVal _
        disposing As Boolean)

        MyBase.Dispose(disposing)
    End Sub
End Class
```

التعامل مع أكثر من نموذج

بادئ ذي بدء، عليك معرفة ان لكل مشروع من النوع Windows Application نافذة رئيسية تسمى **النافذة الابتدائية Startup Window** يمكنك تحديدها عند خانة Startup Object في نافذة Project Property Pages (**شكل 13-3**). هذه النافذة لها طابع خاص عن سائر نوافذ البرنامج، فهي ستظهر مع تنفيذ البرنامج بشكل تلقائي دون الحاجة لتعريف كائن من فئتها واستدعاء الطريقة ()Show. شيء اخر مهم حول هذه النافذة، وهو ان نهاية البرنامج تعتمد على اغلاق المستخدم لهذه النافذة، وحتى لو وجدت عشرات النوافذ المفتوحة سيتم إغلاقها أيضا.



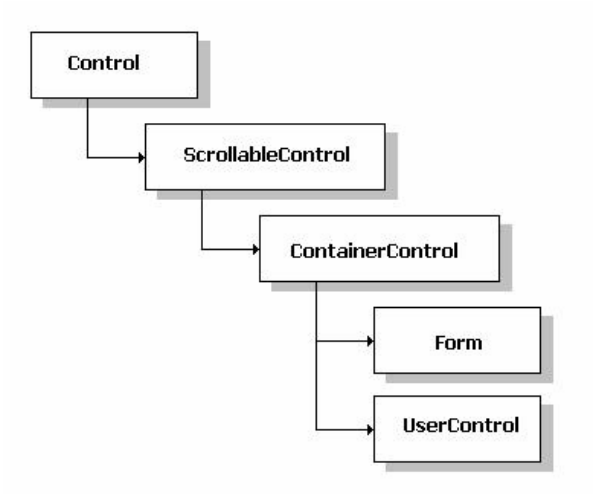
شكل 13-3: تحديد النافذة الابتدائية للمشروع.

مع ذلك، يمكنك تحديد النافذة الابتدائية وقت التنفيذ في كل مرة تستدعي فيها الإجراء Application.Run() والذي تتطلب وسيطته الوحيدة كائن النافذة، أفضل مكان يمكنك استخدام الطريقة Run() السابقة هو الإجراء الابتدائي Sub Main() حيث تحدد من البداية النافذة الابتدائية:

```
Module MainModule
    Sub main()
        Dim frmMain As New Form1()

        Application.Run(frmMain)
    End Sub
End Module
```

محل الفئة Form من الإعراب

محل الفئة Form من الإعراب في فضاء الاسماء System.Windows.Forms يوضحه لك **(شكل 13-4)**، حيث يعرض لك العلاقة الوراثية بين الفئة Form ومجموعة من الفئات في فضاء الاسماء System.Windows.Forms.



شكل 13-4: العلاقة الوراثية بين الفئة Form ومجموعة من الفئات الاخرى.

الفئة القاعدية Control تمثل أداة Control تقليدية تظهر في صندوق الأدوات ToolBox **(شكل 14-1 بالفصل القادم)** يمكنك وضعها على نافذة النموذج. والفئة ScrollableControl تمثل أيضا أداة Control تقليدية ولكن لها سمات إضافية تتمحور حول دعم أشرطة التمرير Scroll Bars للأداة. بالنسبة للفئة ContainerControl فهي مشتقة من الفئة ScrollableControl تضيف اليها قابلية ان تكون الاداة حاضنة Container لأدوات أخرى.

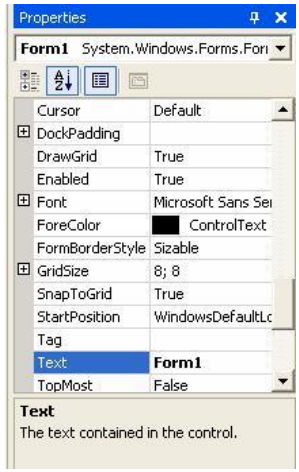
سأتحدث عن جميع الفئات الموجودة في (**الشكل 13-4**) السابق بالتفصيل الممل في الفصل القادم **الأدوات Controls**، حيث كان هدفي من هذه الفقرة فقط توضيح الفئات القاعدية للفئة Form واعلامك ان كل الخصائص، الطرق، والاحداث التابعة لهذه الفئات (والتي لن أتطرق لها في هذا الفصل) موجودة أيضا في الفئة Form.

الخصائص، الطرق، والأحداث

كما ذكرت قبل قليل ان معظم الخصائص، الطرق، والأحداث التابعة للفئة Form هي مشتقة من فئات اخرى أجلت تفصيلها إلى الفصل القادم. لذلك، سأعرض لك في هذا القسم أعضاء الفئة Form فقط بشكل مختصر، وبإمكانك مراجعة مكتبة MSDN للحصول على اكبر قدر من التفاصيل.

خصائص النموذج

قبل ان ابدأ في عرض خصائص النموذج، بودي اخبارك انك تستطيع تعديل قيمها وقت التصميم عن طريق نافذة الخصائص Properties (**شكل 13-5**) والتي تصل لها بالضغط على المفتاح [F4] أو اختيار الامر Properties Windows من قائمة View.



شكل 13-5: نافذة الخصائص Properties.

العمود الايسر يمثل اسماء الخصائص بينما الأيمن قيمة كل خاصية من هذه الخصائص، تذكر ان أي تعديل على هذه الخصائص يقوم بتوليد الشيفرة المناسبة في الإجراء InitializeComponent() التابع لمصمم النماذج Form Designer في فئة النافذة. جرب – مثلا- تغيير قيمة الخاصية Text، ستلاحظ ان مصمم النماذج أضاف هذا السطر:

```
Private Sub InitializeComponent()
    …
    …
    Me.Text = "النافذة الرئيسية"
End Sub
```

خصائص المظهر:

للتحكم في شريط النافذة العلوي، لديك الخاصية Text السابقة والتي كان غرضها تعديل العنوان الذي يظهر في شريط النافذة العلوي وهي خاصة حرفية من النوع String. بينما الخاصيتان MaximizeBox و MinimizeBox منطقية من النوع Boolean حيث تحدد فيها إمكانية استخدام زر التكبير والتصغير الخاص بالنافذة، ان جعلت قيم كلا الخاصيتين False فستختفي الأزرار من شريط النافذة، كما سيختفي صندوق التحكم وزر الإغلاق أيضا ان أسندت القيمة False للخاصية ControlBox.

الخاصية Icon فتحدد فيها رمز (ايقونة) للنافذة، والخاصية BackgroundImage تمكنك من وضع صورة تغطي سطح النافذة، ضع في اعتبارك بان الصورة ستتكرر حتى تغطي كامل سطح النافذة (كتأثير الاختيار Tile الذي تحدده عن تحديد صورة لخلفية سطح المكتب Wall Paper).

الخاصية Opacity تسند لها قيمة مجالها من 1 إلى 0 تحدد فيها مقدار شفافية النافذة، القيمة 1 تظهر النافذة دون شفافية، والقيمة 0 تخفي النافذة تماما، جرب وضع القيمة 0.5 لتصبح النافذة شبه شفافة.

بالنسبة للخاصية TransparencyKey، ففيها تحدد اللون الذي تريد إخفائه من النافذة لحظة التنفيذ. تذكر ان اخفاء اللون من النافذة يعطي فرصة كبيرة لظهور النوافذ التي خلف النافذة الحالية، بل ويمكن أيضا المستخدم من اختراق النافذة ليصل إلى النوافذ التي خلفها.

من الخصائص أيضا، الخاصية ShowInTaskBar والتي تظهر زر للنافذة في شريط المهام Task Bar والخاص بنظام التشغيل.

يمكنك اختيار حد من 7 حدود تدعمه الخاصية FormBorderStyle: القيمة Sizable تضمن للمستخدم قدرته على تحجيم النافذة، اما القيمتين FixedSingle و Fixed3D فتمنع المستخدم من تحجيم النافذة، كما تفعل القيمة None والتي تلغي الحدود بشكل نهائي.

اخيرا، يمكنك اسناد القيمة Hide للخاصية SizeGripStyle ان اردت اخفاء عصا التحجيم Size Grip للنافذة (عصا التحجيم هي خطين يظهران في الزاوية السفلى اليمني للنافذة).

خصائص الموقع والحجم:

بالنسبة للخاصيتين DesktopBounds و DesktopLocation فتوجد نسختين منهما معرفة كطرق هما SetDesktopBounds و SetDesktopLocation سأعود اليهما لاحقا في فقرة **طرق النموذج**.

يمكنك جعل نافذة النموذج تظهر في وسط الشاشة باسناد القيمة CenterScreen إلى الخاصية StartPosition. كما تستطيع تحديد الحجم الابتدائي للنافذة عن طريقة الخاصية WindowState والتي يمكن ان تكون مكبرة Maximized، مصغرة Minimized، أو الحجم الطبيعي Normal.

وعلى ذكر حجم النافذة، يمكنك أيضا تحديد مقاييس لحجم التكبير والتصغير للنافذة عن طريق الخاصيتين MaximumSize و MinimumSize، رغم اني انصحك بعدم تغيير هاتين القيمتين حتى لا تسبب إرباك لمستخدمي نوافذك والذين اعتادوا على ان يكون الحجم الكبير يغطي الشاشة والصغير يظهر شريط العنوان فقط للنافذة.

اخيرا، اسند القيمة True إلى الخاصية TopMost ان اردت جعل النافذة فوق النوافذ الاخرى لكافة تطبيقات Windows.

خصائص أشرطة التمرير Scrolling:

كل ما هو مطلوب منك إسناد القيمة True إلى الخاصية AutoScroll ان اردت اضافة اشرطة تمرير Scroll Bars إلى نافذة النموذج، ضع في اعتبارك ان اشرطة التمرير لن تظهر إلا ان كان حجم النافذة لا يغطي كافة الأدوات المحضونة بها. يمكنك الاستعلام وقت التنفيذ ما اذا كان شريط التمرير الافقي أو العمودي قد ظهرت فعلا عن طريق الخاصيتين HScroll و VScroll.

الخاصية AutoScrollMargin تحدد فيها حجم الحدود الوهمية التي تحيط بالأداة، هذه الحدود الوهمية ليست ظاهرة، وانما تستخدمها نافذة النوذج لبدء عرض اشرطة التمرير ان وصلت حدود النموذج هذه الحدود الوهمية. يفضل إبقائها كما هي (0 ;0).

PDF

تستطيع الاستعلام ومعرفة مواقع أشرطة التمرير الحالية عن طريق الخاصية AutoScrollPosition والتي تعود بكائن من النوع Point. يمكنك أيضا اسناد قيم لها بإنشاء كائن من النوع Point لتحريك اشرطة التمرير برمجيا، الإحداثيات التي ترسلها تمثل النقطة الظاهرة في الزاوية العلوية اليسرى للنافذة:

```
' حرك اشرط التمرير بحيث تمثل تظهر النقطة (10، 20) في
' الزاوية العليا اليسرى من النافذة
Me.AutoScrollPosition = New Point(10, 20)
```

انظر أيضا

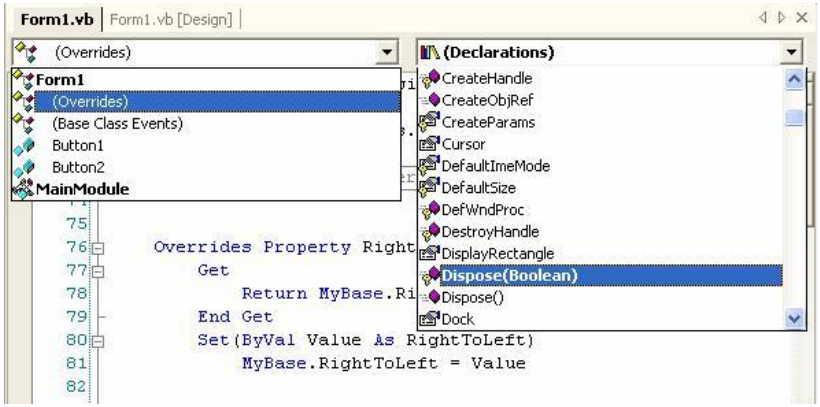
لي عودة حول الكائنات من النوع Point في الفصل القادم **الأدوات** Controls.

إلى جانب الخاصية AutoScrollPosition، توجد الطريقة ()ScrollControlIntoView والتي تحرك اشرطة التمرير بحيث تظهر لك الاداة التي ترسل كوسيطة لها:

```
Me.ScrollControlIntoView (TextBox1)
```

طرق النموذج

قبل عرض الطرق الخاصة بنافذة النموذج، بودي تذكيرك بان معظم الطرق (والخصائص أيضا) قابلة لإعادة القيادة Overrides (ففئات النماذج مشتقة من الفئة Form)، ويمكنك الاستعانة بمحرر الشيفرة لإعادة قيادتها باختيار Overrides من القائمة العلوية اليسرى ومن ثم اسم الطريقة أو الخاصية من القائمة العلوية اليمنى (**شكل 13-6 بالصفحة التالية**):



شكل 13-6: الاستعانة بمحرر الشيفرة لإعادة قيادة الطرق والخصائص.

يمكنك اظهار نافذة النموذج وقت التنفيذ –كما ذكرت- باستدعاء الطريقة Show() للكائن المنشئ منها، ولكن ضع في عين الاعتبار ان الشيفرة التي تلي استدعاء الطريقة Show() سيتم الاستمرار في عملية تنفيذها أيضا:

```
Dim MyForm As New Form2

MyForm.Show ()

' الشيفرة التالية سيتم تنفيذها أيضا
…
…
```

السبب في استمرار تنفيذ الشيفرة التي تلي الاستدعاء للطريقة Show() يتعلق بالبنية التحتية لتركيبة نوافذ نظام التشغيل Windows، حيث ان الاستدعاء Show() يظهر نافذة تسمى –في عالم برمجة Windows- بالـ **Modeless Window** وهو اسلوب يمكن الشيفرة المستدعية للنافذة من الاستمرار في التنفيذ، كما يمكن مستخدم النافذة أيضا من العودة للنافذة وتنشيطها. اما استدعائك للطريقة ShowDialog() فهو مناسب جدا لفتح نوافذ من النوع **Modal Window** (اغلب صناديق الحوار Dialog Boxes يتم فتحها بهذا الاسلوب) حيث توقف عمل النافذة الحالية ولن يتمكن المستخدم من العودة إلى النافذة الفاتحة لها حتى يغلق النافذة المفتوحة:

```
Dim MyForm As New Form2

MyForm.ShowDialog()

' لن يتم تنفيذ الشيفرة التالية حتى
' يغلق المستخدم النافذة
…
…
```

بعد فتح النافذة بالطريقة Show() ستصبح هي النافذة النشطة Active Window في البرنامج بشكل تلقائي، مع ذلك يمكن تغيير النافذة النشطة في البرنامج باستخدام الطريقة Activate().

يمكنك في أي وقت اخفاء النافذة باستدعاء الطريقة Hide() والتي تخفي النافذة من عين المستخدم فقط، حيث أنها لا تزال على قيد الحياة، يمكنك اعادة عرضها مرة اخرى باستدعاء الطريقة Show() أيضا. اما ان رغبت في اغلاق النافذة بشكل نهائي، فالطريقة Close() ستكون وافية وكافية.

الطريقة SetDesktopLocation() تمكنك من تحريك النافذة وتغيير موقعها على سطح المكتب، الوسيطة الاولى تمثل المحور السيني x والثانية المحور الصادي y، حيث تمثل النقطة (0، 0) الزاوية العليا اليسرى من سطح المكتب، تزيد أفقيا كلما اتجهنا يمينا، وعموديا كلما اتجهنا إلى الاسفل. وان رغبت في تحريك النافذة مع تغيير حجما في ضربة واحدة، استدعي الطريقة SetDesktopBounds() والتي تتطلب وسيطتين إضافيتين هما عرض النافذة وارتفاعها:

```
Dim MyForm As New Form2

MyForm.SetDesktopBounds(0, 0, 200, 100)
```

الطريقتين AddOwnedForm() و RemoveOwnedForm():

في بيئة Windows توجد فلسفة **النوافذ المالكة Owner Windows والنوافذ المملوكة Owned Windows**. النوافذ المملوكة تظهر دائما فوق النافذة المالكة لها، ويتم إغلاقها بشكل تلقائي ان أغلقت النافذة المالكة. المزيد أيضا، عند تصغير النافذة المالكة Minimize سيتم تصغير كافة النوافذ المملوكة وتظهر جميعها في زر واحد في اسفل شريط المهام Task Bar.

يمكنك لنوافذك ان تمتلك نوافذ اخرى بإضافتها بالطريقة AddOwnedForm():

```
Dim x As New ownedForm()
Dim y As New ownedForm()
```

```
Me.AddOwnedForm(x)
Me.AddOwnedForm(y)

x.Show()
y.Show()
```

بعد اضافة النافذة كمملوكة، يمكنك الاستعلام عنها بالخاصية OwnedForms والتي تعود بمرجع يمثل جميع النوافذ المملوكة:

```
Dim frm As Form

For Each frm In Me.OwnedForms
    Frm.Text = "نافذة مملوكة"
Next
```

يمكنك معرفة ما اذا كانت النافذة مملوكة لنافذة اخرى عن طريق الخاصية Owner والتي تعود بمرجع لنافذة النموذج المالكة:

```
If Me.Owner Is Nothing
    ' النافذة الحالية لا تمتلكها أي نافذة اخرى
    …
    …
Else
    ' النافذة الحالية مملوكة من قبل نافذة اخرى
    …
    …
End If
```

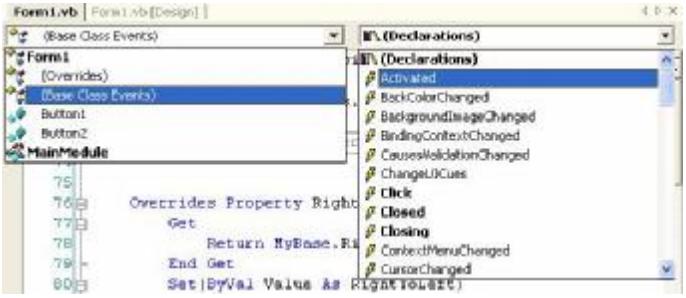
اخيرا، تستطيع الاستغناء عن ملكية النافذة باستدعاء الطريقة RemoveOwnedForm() والتي تتطلب منك وسيطة واحد تمثل مرجع النافذة المملوكة.

أحداث النموذج

نافذة النموذج هي أكثر كائن من كائنات إطار عمل NET. Framework. بشكل عام يحتوي على أحداث، اغلب أحداثه مشتقه وراثيا من الفئة Control ولذلك سأؤجل تفصيل هذه الأحداث حتى الفصل القادم، اما هنا فسأعرض الأحداث الخاصة بنافذة النموذج فقط.

قبل البدء بعرض الأحداث الخاصة بنافذة النموذج، دعني أذكرك بان محرر الشيفرات والخاص ببيئة التطوير Visual Studio .NET يسهل عليك امر قنص الأحداث، وذلك باختيار

Base Class Events من القائمة العلوية اليسرى، ومن ثم اختيار الحدث المراد قنصه من القائمة العلوية اليمنى (**شكل 13-7**).



شكل 13-7: الاستعانة بمحرر الشيفرات لقنص الأحداث.

يمكنني ان الخص لك تسلسل الأحداث التي تقع على نافذة النموذج من بداية إنشاء كائنه وفتح النافذة حتى اغلاق النافذة وقتل كائنه بهذا التسلسل:

المشيد New() -> الحدث Load -> الحدث Paint -> الحدث Activated -> الحدث Deactivated -> الحدث Closing -> الحدث Closing -> الحدث Closed -> المهدم Dispose().

ملاحظة

توجد أحداث اخرى لم أتطرق لها داخلة في عناصر السلسلة السابقة (كـ Resize، Move، LostFocus، GotFocus ...الخ) فضلت تأجيلها إلى الفصل القادم، وذلك لأنها تتبع للفئة القاعدية Control بالأصل وليس Form.

المشيد New() والمهدم Dispose():

تحدثت ما فيه الكفاية عن المشيدات والمهدمات سابقا في الفصل الثالث **الفئات والكائنات**، ولن اضيف شيئا جديدا هنا إلا تذكيرك باستدعاء المشيد والمهدم للفئة القاعدية Base Class قبل كتابة حرف واحد من حروف الشيفرات المصدرية:

```
Public Sub New()
    MyBase.New()
    …
    …
End Sub

Protected Overloads Overrides Sub Dispose(ByVal _
    disposing As Boolean)

    If disposing Then
        …
        …
    End If

    MyBase.Dispose(disposing)
    …
    …
    …
End Sub
```

انظر أيضا

يمكنك مراجعة الفقرة الفرعية **أسلوب أكثر أمانا لكتابة المهدمات** في الفصل الثالث **الفئات والكائنات** لاستيعاب المهدم ()Dispose. كما تحدثت عن محدد الوصول Protected وإعادة التعريف Overloadding وإعادة القيادة Overriding في الفصل الرابع **الوراثة**.

الحدث Load:

يتم تنفيذ هذا الحدث بمجرد البدء في تحميل النافذة وقبل إكمال ظهورها. عند استدعائك للطريقة ()Show التي تظهر النافذة، سيتم تنفيذ هذا الحدث مرة واحدة فقط ولكن قبل ان تظهر النافذة، وللتأكد من ذلك جرب كتابة هذه الشيفرة في إجراء قنص الحدث Load:

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    MsgBox("جاري تحميل النافذة")
End Sub
```

عند فتح النافذة بالطريقة ()Show، ستلاحظ ان الرسالة السابقة ستظهر قبل ظهور النافذة.

PDF

الحدث Paint:

يتم تنفيذ الحدث Paint كلما دعت الحاجة إلى اعادة رسم النافذة، فلو وضعت النافذة س فوق النافذة ص ومن ثم حركت النافذة س لتظهر النافذة ص، سيتم تنفيذ الحدث Paint التابع للنافذة ص. كذلك، عند ظهور أي جزء مخفي من النافذة نتيجة تحريكها خارج حدود سطح المكتب سيتم تفجير الحدث Paint أيضا.

عند ظهور أو اختفاء اشرطة التمرير Scrollbars والتابعة للنافذة، سيتم تنفيذ الحدث Paint أيضا، كذلك الحال عند تحريك اشرطة التمرير من قبل المستخدم فهو سبب بديهي لاعادة رسم النافذة.

من الحالات الاخرى التي تؤدي إلى تنفيذ الحدث Paint تكبير حجم النافذة من قبل المستخدم، وذلك لظهور اجزاء اضافية تتطلب اعادة الرسم، اما تصغير حجم النافذة فلا يؤدي إلى تنفيذ الحدث، وان كان لديك سبب وجيه لتنفيذ الحدث Paint عند تصغير حجم النافذة، فيمكنك استدعائه من داخل الحدث:

```
Private Sub Form1_Resize(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Resize

    Me.Form1_Paint(Me, Nothing)
End Sub
```

الاستدعاء السابق صحيح ولكنه يطلق الحدث Paint مرتين الاول بسبب اعادة الرسم والثانية بسبب الاستدعاء الناتج من تغيير الحجم، لذلك يفضل استدعاء الطريقة Refresh() والتي تعيد رسم النافذة كلما دعت الحاجة:

```
Private Sub Form1_Resize(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Resize

    Me.Refresh()
End Sub
```

الحدثان Activated و Deactivated:

في كل مرة تصبح النافذة الحالية هي النافذة النشطة Active Window، سيتم تنفيذ الحدث Activated، وبمجرد اختفاء التركيز عن النافذة سيتم تنفيذ الحدث Deactivated.

الحدثان Closing و Closed:

يتم تنفيذ الحدث Closing بمجرد البدء باغلاق النافذة سواء برمجيا باستدعاء الطريقة Close() أو ان قام المستخدم بالضغط على زر اغلاق النافذة. مع ذلك، لديك فرصة كبيرة في منع عملية الاغلاق وذلك بارسال القيم True إلى الخاصية Cancel التابعة لوسيطة الحدث:

```
Private Sub Form1_Closing(ByVal sender As Object, ByVal e _
    As ComponentModel.CancelEventArgs) Handles MyBase.Closing

    e.Cancel = True
End Sub
```

ان تم اغلاق النافذة بشكل نهائي واختفت من عين المستخدم، سيأتي دور تنفيذ الحدث Closed معطيا لك فرصة اخيرة لعمل ما تريد، مع العلم ان هذا الحدث يعني ان النافذة قد تم اغلاقها ولن تتمكن من الغاء عملية الاغلاق أو حتى اعادة اظهار النافذة باستدعاء الطريقة Me.Show().

أحداث اخرى:

من الأحداث الاخرى والخاصة بنافذة النموذج الحدث MinimumSizeChange الذي يتم تنفيذه بمجرد تغيير قيمة الخاصية MinimumSize، الحدث MaximumSizeChange يتم تنفيذه بمجرد تغيير قيمة الخاصية MaximumSize، والحدث MaximizedBoundsChange المرافق للخاصية MaximizedBounds.

نماذج MDI Forms

يمكن لأي نافذة نموذج –مهما كان اصلها وفصلها- ان تكون نافذة من النوع MDI، وهو النوع الذي يمكن النافذة من ان تحضن نوافذ اخرى. كل ما هو مطلوب منك لجعل النافذة من النوع MDI اسناد القيمة True إلى الخاصية IsMdiContainer والتي يمكن ان تكتب وتقرأ وقت التصميم والتنفيذ.

من المزايا في نماذج Windows Forms هو امكانية تعريف أكثر من نافذة MDI في برنامج واحد. مع ذلك، لا تحاول عمل ذلك إلا ان دعت الحاجة فعلا لذلك، فالأغلبية الساحقة لتطبيقات Windows من النوع MDI تستخدم نافذة MDI واحدة.

النوافذ الأبناء Child Windows

كما يمكن لاي نافذة –مهما كان اصلها وفصلها- ان تكون نافذة من النوع MDI، يمكن لاي نافذة أيضا ان تكون نافذة ابنة (النوافذ المحضونة في نوافذ MDI تسمى **نوافذ ابنـــاء Child**)، اســند مرجع إلى النافذة MDI في الخاصية MdiParent التابعة للنافذة الابنة (**شكل 13-8**):

```
Sub main()
    Dim frmMain As New Form1()
    Dim frmChild As New Form1()

    frmMain.Text = "MDI Form"
    frmMain.IsMdiContainer = True

    frmChild.MdiParent = frmMain
    frmChild.Text = "Child Form"
    frmChild.Show()

    Application.Run(frmMain)
End Sub
```

شكل 13-8: نافذة ابنة Child محضون داخل نافذة MDI.

ضع في عين الاعتبار، ان النافذة الابن تكون مملوكة للنافذة MDI المحددة في الخاصـــية MdiParent، نستنتج من ذلك، ان النافذة الابن سيتم اغلاقها تلقائيا ان أغلقت النافذة المالكة، كما سيتم تصغيرها تلقائيا ان تم تصغير النافذة المالكة.

الفرق البسيط بين ملكية النوافذ باستخدام الطريقة AddOwnedForm(), وملكية النوافذ باسناد قيمة للخاصية MdiParent هو ان الثانية تكون النوافذ المملوكة محضونة داخل حدود النافذة المالكة، بينما في الحالة الاولى فتكون النوافذ المملوكة خارج حدود النافذة المالكة.

فرق اخر يتعلق بطريقة اظهار القوائم Menus، حيث ان قوائم النوافذ المملوكة تعرض في نفس نافذة MDI (النافذة المالكة)، بينما في حالة الملكية باستخدام الطريقة AddOwnedForm() فكل قائمة تعرض في نافذتها بشكل مستقل.

ملاحظة

عند تكبير النافذة الابن Maximize وهي في داخل النافذة MDI، سيتم تغيير العنوان الظاهر في أعلى النافذة MDI بحيث يشمل عنوان نافذة الابن المكبرة والنافذة الام.

خصائص وطرق إضافية

من الطرق والخصائص التي تتعلق بالنوافذ الابناء Child Windows الخاصية IsMdiChild التي تعود بالقيمة True ان كانت النافذة ابنة:

```
If Me.IsMdiChild Then
    …
    …
End If
```

مع ذلك، انصحك بالعودة إلى الخاصية MdiParent حيث انها تعود بمرجع للنافذة MDI –ان وجدت- عوضا عن القيمة True، مما يمكنك من الوصول إلى كافة طرق وخصائص النافذة الحاضنة:

```
If Not Me.MdiParent Is Nothing Then
    Me.MdiParent.Text = "..."
    …
    …
End If
```

وفي سياق نافذة MDI الام، فيمكنك معرفة جميع النوافذ المحضونة بها عن طريق الخاصية MidiChildren، والتي تعود بمصفوفة تمثل مراجع إلى جميع نوافذ الابناء:

```
Dim childForm As Form

For Each childForm In Me.MdiChildren
    childForm.Text = "....."
Next
```

خاصية اخرى تعود بمرجع للنافذة الابن وهي الخاصية ActiveMdiChild والتي تمثل النافذة الابن النشطة:

```
Me.ActiveMdiChild.Text ="النافذة النشطة"
```

في المقابل، يمكنك تنشيط النافذة الابن باستدعاء الطريقة ()ActivateMdiChild والتي تتطلب مرجع إلى النافذة الابن ترسله كوسيطة لها:

```
Me.ActivateMdiChild(Me.MdiChildren(0))
```

المزيد أيضا، النماذج من النوع MDI يتم تفجير حدث إضافي لها هو MdiChildActivate ، يتم تنفيذه بمجرد ان تحصل النافذة الابن على التركيز –أي تكون هي النافذة النشطة:

```
Private Sub MDIForm_MdiChildActivate(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.MdiChildActivate

End Sub
```

اخيرا، الطريقة ()LayoutMdi تستخدم لترتيب النوافذ الابناء، حيث ترسل لها قيمة من اربع قيم للتركيب MdiLayout هي: ترتيب الرموز ArrangeIcons، تتالي Cascade، تجانب افقي TileHorizontal، وتجانب عمودي TileVertical:

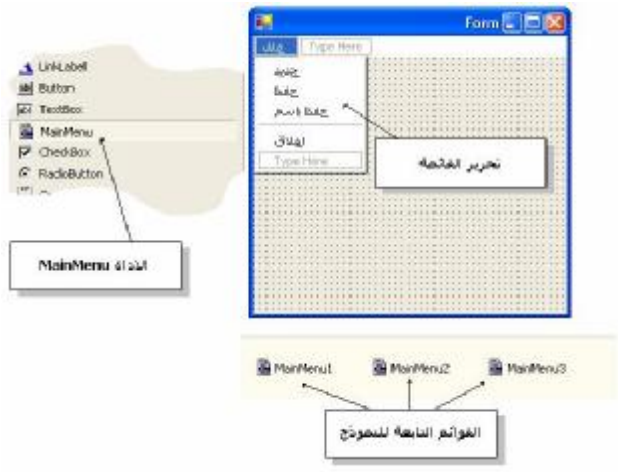
```
Me.LayoutMdi(MdiLayout.ArrangeIcons)
Me.LayoutMdi(MdiLayout.Cascade)
Me.LayoutMdi(MdiLayout.TileHorizontal)
Me.LayoutMdi(MdiLayout.TileVertical)
```

القوائم Menus

في عالم Windows Forms، القوائم Menus ما هي إلا أدوات كالأدوات الموجودة في صندوق الأدوات (**شكل 15-4**)، يمكنك استخدام الأداة المعنونة MainMenu وإضافتها على نافذة النموذج، بل تستطيع إضافة أكثر من أداة MainMenu على نافذة النموذج و سيظهرها لك مصمم

PDF Factory Pro trial version www.fastio.com

النماذج لرموز في اسفل نافذة النموذج، يمكنك تحديد الاداة ومن ثـم البـدء بتحريرهـا مرئيـا Visulally (**شكل 13-9**).



شكل 13-9: إضافة وتحرير القوائم.

يمكنك في أي وقت من حذف قائمة من القوائم التابعة للنموذج بالضغط بزر الفأرة الايمـن على رمزها –اسفل النافذة- واختيار الامر Delete من القائمة المنبثقة، كمـا تـستطيع تغييـر خصائص وقت التصميم عن طريق نافذة الخصائص Properties Window، والتي تمكنك مـن تعديل خصائص اداة القائمة بشكل عام، أو خصائص كل عنصر من عناصر هذه القائمة.

ملاحظة

الرموز أسفل نافذة النموذج تسمى Designer's Tray Area تمثل نوعية من الأدوات لا تظهر بالشكل المتوقع لحظة التنفيذ. سترى الكثير منها في الفصل القادم **الأدوات** Controls.

ان احتوت نافذة النموذج على أكثر من أداة قائمة MainMenu، يمكنك اختيار احدها بإسناد قيمتها إلى الخاصية Menu والتابعة لنافذة النموذج، سواء وقت التصميم أو التنفيذ:

```
Me.Menu = MainMenu2
```

الخصائص، الطرق، والأحداث

فئات القوائم MainMenu مشتقة من الفئة Menu. من ناحية اخرى، عليك معرفة ان الفئة MainMenu تمثل اداة القائمة كلها بما تحتوي، اما بالنسبة لعناصر القائمة فهي كائنات من النوع MenuItem. وبمجرد إضافتك لعناصر جديدة، سيقوم مصمم النماذج بتوليد كائنات إضافية من النوع MenuItem لكل عنصر من هذه العناصر.

بالنسبة للخاصية Text فهي تمثل النص الظاهر على عنصر القائمة، يمكنك استخدام الحرف & وإتباعه بحرف خر يتم تسطيره Underline، حتى تمكن المستخدم من الضغط على المفتاح [Alt] وذلك الحرف لاختيار عنصر القائمة، يمكنك معرفة هذا الحرف وقت التنفيذ عن طريق الخاصية Mnemonic. كما تستطيع اسناد القيمة الحرفية "-" إلى الخاصية Text لعمل فاصل مجموعات للعناصر.

من الخصائص أيضا الخاصية Checked والتي تسند لها القيمة True لوضع علامة على عنصر القائمة، وهي شبيه بالخاصية RadioCheck ولكنها تضع علامة دائرية وجرى العرف على ان تكون الوحيدة المعلمة في المجموعة.

يمكنك معرفة جميع العناصر الفرعية عن طريقة الخاصية MenuItems والتي تعود بمصفوفة من النوع MenuItem تمثل مراجع للعناصر الفرعية، كما تستطيع معرفة القائمة الحاضنة للقائمة الفرعية عن طريقة الخاصية Parent.

المزيد أيضا، يمكنك تحديد اختصار لوحة المفاتيح لتفعيل عنصر القائمة عن طريق الخاصية Shortcut، كما تستطيع اسناد القيمة False إلى الخاصية ShowShortcut ان اردت اخفاء نص اختصار لوحة المفاتيح المجانب لاسم العنصر.

بالنسبة للطرق، فهي تحتوي على مجموعة من الطرق أبرزها الطريقة GetMainMenu() التي تعود بكائن من النوع MainMenu يمثل اداة القائمة، وهناك الطريقة PerformClick() لتنفيذ الحدث Click التابع للعنصر، والطريقة PerformSelect() لتنفيذ الحدث Select التابع للعنصر.

وعلى ذكر الأحداث السابقة، فيوجد حدث النقر Click الذي يتم تنفيذه بمجرد اختيار القائمة من قبل المستخدم، و هناك أيضا الحدث Select الذي يتم تنفيذه ان كان المؤشر فوق العنصر (سواء كان مؤشر الفأرة أو لوحة المفاتيح).

القوائم المنبثقة Popup-Menu

يمكنك عمل قوائم منبثقة Popup-Menus بسهولة شديدة، تحتاج إلى استخدام الأداة ContextMenu عوضا عن الأداة MainMenu، ويمكنك تحريرها بمصمم النماذج كما تفعل مع القوائم الرئيسية. اما ان اردت استخدامها، فعليك ربطها بالأداة وذلك بإسناد كائن النافذة المنبثقة إلى الخاصية ContextMenu التابعة للاداة (سواء وقت التصميم أو تنفيذ):

```
Button1.ContextMenu = ContextMenu1
```

تستطيع ربط نفس القائمة المنبثقة مع أكثر من اداة، وعند قيامك بربطها سيتم اظهارها ان قام المستخدم بالضغط بزر الفأرة الايمن على الاداة.

مع ذلك، لست بحاجة إلى انتظار نقرة المستخدم بزر الفأرة الايمن لعرض القائمة المنبثقة، إذ يمكنك استدعاء الطريقة Show() وارسال الاداة التابعة لها مع موقعها –بالنسبة للاداة:

```
ContextMenu1.Show(Button1, New Point(0, 0))
```

انظر أيضا

الفئة من النوع Point تمثل نقطة، لي عودة اخرى حول هذه الفئة في الفصل القادم **الأدوات** Controls.

تحتوي القوائم المنبثقة على الخاصية DefaultItem والتي تسند القيمة True لها إن أردت ان يكون عنصر القائمة هو العنصر الافتراضي Default Item، والعنصر الافتراضي ليس سوى عنصر مثل باقي العناصر ولا يميزه إلا طريقة كتابته بالزي السميك Bold فقط.

اخيرا، عند ظهور القائمة المنبثقة ContextMenu، فسيتم تنفيذ حدث تابع لها وهو Popup.

نماذج MDI مرة اخرى

اذا كان لكلا النافذة الحاضنة MDI والنافذة المحضونة Child قائمة، فسيتم دمجهما في قائمة واحدة تظهر في النافذة MDI، يمكنك التحكم في طريقة الدمج عن طريق الخاصية MergeType –والمدعومة ليس فقط في الكائن MainMenu بل حتى عناصر القائمة MenuItem.

يمكنك اسناد قيمة من اربع قيم للخاصية MergeType هي: **Add**: سيتم اضافة القائمة إلى قائمة النافذة الحاضنة MDI كما هي، تحديد موقع العنصر يعتمد على ترتيبه في الخاصية MergeOrder. **Remove**: هذه القائمة لن تظهر ابدا في النافذة الحاضنة MDI. **Replace**: سيتم تبديل القائمة بالقائمة الموجودة في النافذة الحاضنة MDI والتي تحمل نفس الترتيب في الخاصية MergeOrder. **MergeItems**: سيتم دمج عناصر هذه القائمة مع عناصر قائمة في النافذة الحاضنة MDI والتي تحمل نفس الترتيب في الخاصية MergeOrder.

مثال للاستخدام الامثل:

عملية الدمج ليست بالصورة السهلة التي تتوقعها، حيث ان بها قليلا من التعقيد، وحتى ترى الاستخدام الامثل للخاصيتين MergeType و MergeOrder، فضلت عرض هذا المثال. بافتراض ان النافذة الحاضنة MDI تحتوي على هذه القائمة:

العنصر		الخاصية MergeType	الخاصية MergeOrder
ملف		MergeItems	0
	جديد	MergeItems	0
	فتح	Remove	1
	حفظ	Remove	2
	انهاء	MergeItems	9
أدوات		Add	5
	خيارات	Add	0
تعليمات		MergeItems	9
	المحتويات	MergeItems	0
	حول...	MergeItems	2

وبافتراض ان النافذة الابنة Child تحتوي على هذه القائمة:

PDF

العنصر		الخاصية MergeType	الخاصية MergeOrder
ملف		MergeItems	0
	جديد	Remove	0
	فتح	Replace	1
	حفظ	Replace	2
	اغلاق	Add	3
تحرير		Add	1
	نسخ	Add	0
تعليمات	MergeItems	9	
	الفهرس	Add	1

عند الدمج، ستكون القائمة النهائية في النافذة الحاضنة MDI بهذا الشكل:

ملف		**(مدمجة من كلا النافذتين)**
	جديد	**(من النافذة الحاضنة MDI)**
	فتح	**(من النافذة المحظونة Child)**
	حفظ	**(من النافذة المحظونة Child)**
	اغلاق	**(من النافذة المحظونة Child)**
	انهاء	**(من النافذة الحاضنة MDI)**
تحرير		**(من النافذة المحظونة Child)**
	نسخ	**(من النافذة المحظونة Child)**
أدوات		**(من النافذة الحاضنة MDI)**
	خيارات	**(من النافذة الحاضنة MDI)**
تعليمات		**(مدمجة من كلا النافذتين)**
	المحتويات	**(من النافذة الحاضنة MDI)**
	الفهرس	**(من النافذة المحظونة Child)**
	حول...	**(من النافذة الحاضنة MDI)**

PDF

الإنشاء الديناميكي للقوائم

عملية الإنشاء الديناميكي للقوائم لحظة التنفيذ لا اعتقد انها ستكون غريبة، فكل ما في الامر اننا نتعامل مع فئات تتطلب إنشاء كائنات منها، ومن ثم اسناد خصائص وقنص أحداثها، وكل هذه المواضيع قد تطرقت لها في الفصل الثالث **الفئات والكائنات** من هذا الكتاب. الشيفرة التالية ستنشئ قائمة منبثقة ContextMenu وقت التصميم وقنص جميع أحداث عناصرها باستخدام AddHandler ليتم تنفيذ الإجراء MenuClicked():

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    Dim X As New ContextMenu()
    Dim sub1 As New MenuItem()
    Dim sub2 As New MenuItem()

    sub1.Text = "الامر 1"
    sub2.Text = "الامر 2"

    X.MenuItems.Add(sub1)
    X.MenuItems.Add(sub2)

    AddHandler sub1.Click, AddressOf MenuCliked
    AddHandler sub2.Click, AddressOf MenuCliked

    Button1.ContextMenu = X
End Sub

Sub MenuCliked(ByVal send As Object, ByVal e As EventArgs)
    MsgBox("تم اختيار الامر")
End Sub
```

مواضيع متقدمة

اختم معك الفصل الثالث عشر **نماذج Windows Forms** بهذا القسم، والذي أتطرق فيه إلى مواضيع متعددة هي : التفاعل مع نوافذ Modal، وراثة النماذج Form Inheritance، والنماذج المحلية Localized Forms.

التفاعل مع نوافذ Modeless

ذكرت في الفقرة **طرق النموذج** سابقا في هذا الفصل، انك تستدعي الطريقة Show() لفتح نافذة نموذج من النوع Modeless:

PDF Factory Pro trial version www.fastio.com

```
Dim MyForm As New Form2

MyForm.Show ()

' الشيفرة التالية سيتم تنفيذها أيضا
…
…
```

وكما في التعليق السابق، سيستمر تنفيذ شيفرة الإجراء المستدعي بشكل متزامن مع إجراءات النافذة التي تم فتحها. في هذه الفقرة سنجيب على السؤال التالي: كيف يمكن للإجراء المستدعي من معرفة انه تم اغلاق النافذة التي فتحها؟

اجابة هذا السؤال نستنتجها من الفصل الثالث **الفئات والكائنات**، فكما أخبرتك مرارا ان نوافذ النماذج ما هي إلا فئات، وأحداثها هي أعضاء لتلك الفئات، لذلك يمكنك قنص الحدث Closing (أو أي حدث اخر) للتفاعل مع النوافذ من النوع Modeless:

```
Sub ShowForm ()
    Dim mdless As New Form1()

    AddHandler mdless.Closing, AddressOf FormHasBeenClosed

    mdless.Show()
End Sub

Sub FormHasBeenClosed(ByVal sender As Object, ByVal e As  _
    System.ComponentModel.CancelEventArgs)

    MsgBox("تم اغلاق النافذة " & sender.text)
End Sub
```

وراثة النماذج Form Inheritance

تصميم النماذج بمصمم النماذج امر ممتع مبدئيا، ولكنه يصبح ممل جدا ان تكررت نفس التصاميم والشيفرات البرمجية الأولية، فمعظم صناديق الحوار تتكرر في أكثر من مشروع، مما يتطلب منك إعادة تصاميمها وكتابة شفراتها من جديد.

بما ان نوافذ النماذج ما هي إلا فئات تقليدية، فيعني ذلك إمكانية اشتقاقها وراثيا وتطبيق الوراثة Inheritance عليها، وبذلك توفر على نفسك الكثير من الوقت المستخدم في التصاميم المكررة لنوافذ النماذج. ليس هذا فقط، بل حتى عند قيامك بتطوير نافذة النموذج القاعدية Base Form (اقصد فئة النموذج القاعدية)، سيشمل هذا التعديل كافة النماذج الوارثة منها، لذلك عند

اكتشافك لأحد الأخطاء في نافذة النموذج القاعدية، فلست بحاجة إلى تعديل كافة النماذج الوارثة منها.

لا يوجد شيء جديد اخبرك به عند وراثة النماذج، فكل ما تعلمناه في الفصل الرابع **الوراثة** يطبق مع النماذج بمرونة كبيرة. مع ذلك، ضع في عين الاعتبار ان فئات النماذج تشمل نوافذ تمثلها، وليس مجرد بيانات كالفئات الاخرى التقليدية.

نقاط اضافية:

من المهم معرفة ماذا تمثل الأدوات Controls التي تضعها بمصمم النماذج بالنسبة لفئة النموذج، اذ عليك ان تعلم علم اليقين ان كل اداة تضعها تمثل حقل Field في فئة النموذج، فلو وضعت اداة TextBox سيقوم مصمم النماذج بتعريفها كحقل:

```
Class Form1
    Inherits System.Windows.Forms.Form
    …
    …
    Friend WithEvents TextBox1 As Forms.TextBox
    …
    …
End Class
```

ينصح دائما بتغيير محدد الوصول للحقول، وجعله Private حتى تحمي فئة النافذة من العبث بأدواتها من خارجها، أو Protected ان رغبت في السماح للنافذة المشتقة فقط من الوصول إلى اعضاء النافذة الحالية.

ملاحظة

بدلا من تعديل محدد الوصول والخاص بالأداة يدويا من خلال الشيفرة المصدرية، يمكنك استخدام خاصية الأداة Modifier من نافذة الخصائص.

استخدامك لمحدد الوصول Private سيحرم النافذة المشتقة من أشياء كثيرة منها: قنص أحداث الأدوات، اعادة تعريف طرقها وخصائص، والاهم من ذلك سيحرمها من الوصول إلى اعضاء الاداة ولن تستفيد منها في الحصول على أية معلومات.

PDF Converter

مع ذلك، ان استخدمت محدد الوصول Private يمكنك السماح للنافذة المشتقة من الوصول إلى بعض عناصر الاداة عن طريق كتابتها يدويا بنفس:

```
Class Form1
    Inherits System.Windows.Forms.Form
    …
    …
    Private WithEvents TextBox1 As Forms.TextBox

    ' يمكن للفئة المشتقة الوصول إلى هذه الخاصية
    Friend Property TextValue() As String
        Get
            Return TextBox1.Text
        End Get

        Set(ByVal Value As String)
            TextBox1.Text = Value
        End Set
    End Property
    …
    …
End Class
```

مثال تطبيقي:

قم بتصميم نافذة نموذج، وضع الأدوات عليها واجري كافة التعديلات على خصائصها وتنسيقاتها، في (**الشكل 13-10**) صممت صندوق حوار بسيط خاص لكلمات المرور، اسم فئته PasswordForm.

شكل 13-10: تصميم مبدئي لنافذة قاعدية.

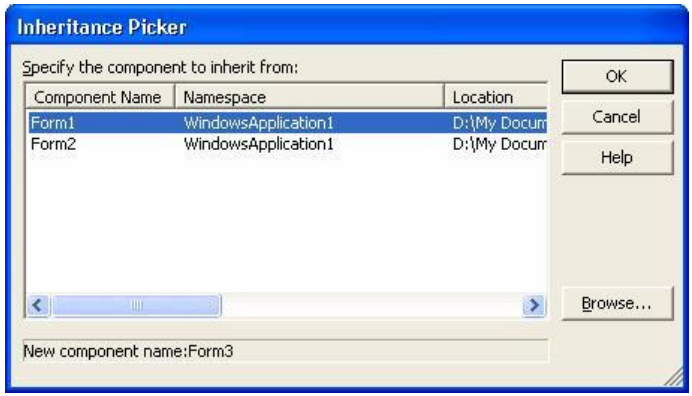
بعد تصميم صندوق الحوار، اختر الامر Build xxx (حيث xxx اسم المشروع) من قائمة Build، حيث لابد من ترجمة الملف حتى نتمكن من وراثة النافذة السابقة، أنشئ ملف جديد وقم فورا بوراثة النافذة PasswordForm بهذا الشكل:

```
Class TestForm
    Inherits PasswordForm
    …
    …
End Class
```

ملاحظة

يمكنك وراثة النافذة أيضا دون الحاجة لاختيار الامر Build xxx لترجمة المشروع، ولكنك في هذه الحالة لن تتمكن من مشاهدة التأثيرات الأولية لوراثة النموذج في مصمم النماذج.

لست بحاجة لكتابة الشيفرة يدويا بنفسك، اذ يمكنك الاعتماد على بيئة التطوير Visual Studio .NET في وراثة النماذج، اختر الامر Add Inherited Form من القائمة Project، اكتب اسم الملف الذي تريده ثم اضغط على الزر Open، سيظهر لك صندوق حوار بعنوان Inheritance Picker (**شكل 13-11**).



شكل 13-11: صندوق الحوار Inheritance Picker.

حدد النافذة التي تود وراثتها واضغط على الزر OK، ستلاحظ ان نافذة نموذج جديدة ظهرت ولكنها ليست خالية، بل تحتوي على جميع أدوات وخصائص النافذة القاعدية (**شكل 13-12**)، وان أمعنت النظر سترى الرمز في أعلى الزاوية اليسرى لكل اداة، ليعلمك ان هذه الاداة وارثة من النافذة القاعدية.

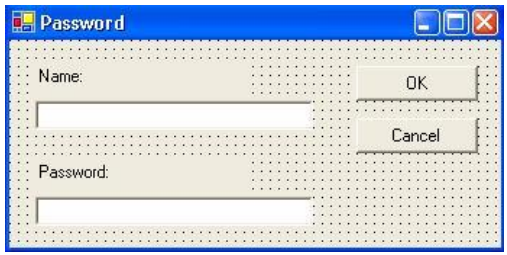
شكل 13-12: نافذة النموذج الجديدة قد ورثت النافذة القاعدية.

يمكنك تعديل خصائص الأدوات الذي سمح لك محدد الوصول المستخدم للأداة في النافذة القاعدية بذلك، فلو كان محدد وصول الأداة Private فلن تتمكن من عمل أي شيء، حتى لو حاولت فتح نافذة الخصائص لتغيير خصائص الاداة، ستلاحظ انها اصبحت باللون الخافت ولن تتمكن من تعديل أي خاصية فيها.

النماذج المحلية Localized Forms

حتى لو لم تكن تنوي تطوير تطبيقات متعددة اللغات، من الجيد إعطاء نكهة محلية لنماذج برنامجك، فقد يأتي يوم من الايام الذي تود تغيير واجهة نوافذ برنامجك ليغلب عليه الطابع المحلي للدولة المستخدمة، وبغض النظر عن مدى قبولك لهذه الفكرة، فالسهولة التي توفرها لك بيئة التطوير Visual Studio .NET لا تعطيك حجة لعدم فعل ذلك.

تخيل معي انني صممت نافذة لاستقبال كلمة المرور من المستخدم (**شكل 13-13**)، باللغة الإنجليزية، ووضعت في اعتباراتي انها لمستخدمين اجانب حيث محاذاة الأدوات وطريقة ترتيبها يغلب عليها طابع الاتجاه من اليسار إلى اليمين.

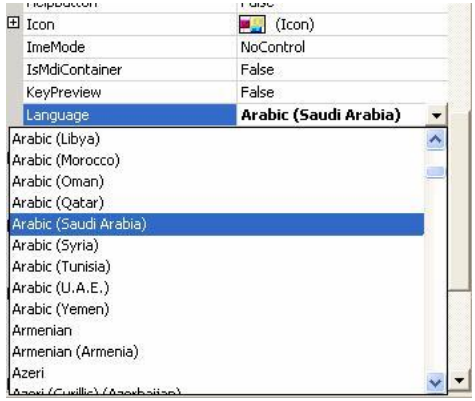


شكل 13-13: نافذة نموذج تقليدية.

هب مثلا اني بعد فترة اردت تغيير واجهة الاستخدام، وهذا الامر بحد ذاته سيكلف الكثير من الوقت والجهد، خاصة ان المسؤولية ستنحصر علي عند حصول أي خطأ، والسبب ان أي تعديل لمحتويات هذه النافذة ستتأثر فيه الشيفرة البرمجية، يبقى الحل الأسهل هو بجعل هذه النافذة نافذة محلية Localized Form.

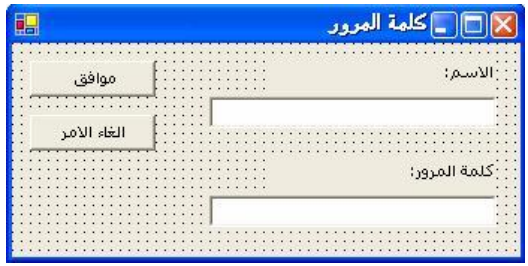
كل ما هو مطلوب منك لجعل النافذة محلية اسناد القيمة True إلى الخاصية Localizable من نافذة الخصائص (فهي ممكنة في وقت التصميم فقط)، وبمجرد إسنادك للقيمة True لهذه الخاصية فانك تطلب من مصمم النماذج تحويل جميع الشيفرات المولدة والتي تتعامل مع الخصائص إلى **ملف المصادر Resource File**، بحيث يمكن لبيئة التطوير من قراءة قيم الخصائص من هناك.

وحتى تفهم الهدف من الفكرة السابقة، حدد الدولة المراد دعمها في نافذة النموذج المحلية عن طريق الخاصية Language (**شكل 13-14**) من نافذة الخصائص (فهي أيضا خاصية وقت التصميم فقط).



شكل 13-14: اختيار Arabic (Saudi Arabia) كأحد لغات النافذة.

بمجرد تحديدك للدولة في الخاصية Language قم بتصميم النافذة ليغلب عليها الطابع المحلي بالشكل الذي تريده، فالنافذة السابقة (**شكل 13-13**) قم بتعديل خصائصها وكتابتها باللغة العربية، كما أعدت ترتيب محاذاة الأدوات ليكون من اليمين إلى اليسار (**شكل 13-15**).



شكل 13-15: نافذة النموذج باللغة Arabic (Saudi Arabia)

والان لدينا نافذة نموذج واحد ولكن بتصميمين مختلفين، يمكنك رؤية التصميم القديم (**شكل 13-13**) باختيار القيمة (Default) من الخاصية Language، وتستطيع العودة إلى التصميم الجديد (**شكل 13-15**) باختيار اللغة Arabic (Saudi Arabia) من نفس الخاصية.

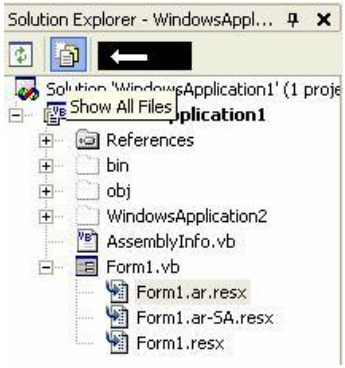
التعامل مع نوافذ النماذج المحلية هو نفس التعامل مع نوافذ النماذج التقليدية، ولا يوجد أي فرق جوهري يحتاج توضيحه، وان سألتني سؤال واخبرتني أي من النموذجين سيتم عرضه على المستخدم؟ فإجابتي ستكون بالاعتماد على الاعدادت الإقليمية في جهاز المستخدم، وان كانت

الاعدادات الإقليمية في جهاز المستخدم ليست مدعومة، فسيظهر النموذج (**شكل 13-13**) وذلك لان لغته هي الافتراضية (فقد صممته عندما كانت قيمة الخاصية Language هي (Default)).

نظرة خاطفة خلف الكواليس:

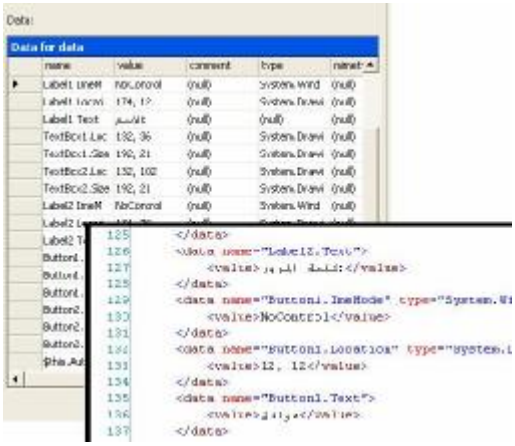
كما رأيت، عملية تدويل النماذج Internationalization Forms لا يتطلب إلا مجهود يسير جدا من طرف المبرمج، مع ذلك فهو يتطلب مجهود كبير من قبل بيئة التطوير Visual Studio .NET، حيث ان كل التصاميم مختلفة اللغات للنماذج، تحفظ قيم خصائصها في ملفات مصادر خاصة بنافذة النموذج، يحمل نفس اسم ملف النموذج بالإضافة إلى رمز الدولة مع الامتداد resx. في نفس المجلد.

يمكنك عرض ملف المصادر التابع لكل نافذة نموذج عن طريق نافذة مستكشف المشروع Solution Explorer، ولكن ذلك يتطلب منك الضغط على الزر العلوي لهذه النافذة والمسمى Show All Files (**شكل 13-16**).



شكل 13-16: عرضت ملفات المصادر بعد الضغط على الزر Show All Files.

لست هنا بصدد شرح ملفات المصادر، فيمكنك التجول فيها والعبث بمحتوياتها يدويا، ولكن دعني انوه هنا بان البيانات الموجودة في شكل جداول في ملفات المصدر، تحفظ في الخلفية بالصيغة XML (**شكل 13-17 بالصفحة التالية**). تستطيع تحرير الملفات بصيغة XML يدويا بالضغط على خانة التبويب XML في أسفل نافذة المصادر.



شكل 13-17: ملفات المصادر تحفظ بالصيغة XML.

ملاحظة

يمكنك مراجعة الشيفرة المولدة من قبل مصمم النماذج المحلية حتى ترى أمثلة وافية وشافية لطريقة قراءة البيانات من ملفات المصادر باستخدام الفئة Resources.

نصيحتي لك بان لا تطوي غلاف الكتاب الآن، بل تابع واستمر في قراءة الفصل التالي **الأدوات Controls** فهو مرتبط ارتباطا وثيقا بهذا الفصل، وفيه عرض من الخصائص التي لا تعتبر خاصة بالأدوات وحسب، بل تتبع أيضا للنماذج –حيث ان كلاهما مشتق من الفئة القاعدية Control.

الأدوات Controls

الاعتماد على نافذة النموذج وحدها لا يكفي لإنجاز تطبيقات Windows إنتاجية، حيث ان استخدام الأدوات Controls يعتبر جزء لا يتجزأ من مراحل تطوير برامجك المعتمدة على نماذج Windows Forms.

جميع الأدوات التي توضع في نماذج Windows Forms مشتقة وراثيا بشكل مباشر أو غير مباشر من الفئة Control (**شكل 13-4 صفحة 449**)، وبالتالي فجميع خصائص، طرق، وأحداث الفئة Control ستكون أيضا مدعومة في سائر الأدوات، لذلك وجدت انه من الأفضل – لي ولك- البدء بعرض الأعضاء المشتركة بين الأدوات (أعضاء الفئة Control) ومن ثم ذكر كل أداة على حده.

ملاحظة

لا تنسى ان هذه الخصائص، الطرق، والأحداث مدعومة أيضا في نافذة النموذج Form والتي تحدثت عنها في الفصل السابق، وذلك لان الفئة Form مشتقة وراثيا أيضا من الفئة Control.

الخصائص المشتركة

كما هو الحال مع نافذة النموذج، يمكن تعديل خصائص الأدوات وقت التصميم عن طريق نافذة الخصائص Windows Properties (**شكل 14-5**) وذلك بعد تحديد الأداة، ليقوم مصمم النماذج بتوليد الشيفرة الضرورية في الإجراء InitializeComponent() والخاص بصمم النماذج.

الشيفرة التي ولدها مصمم النماذج لحظة تعديل الخصائص، علمتني أشياء كثيرة وسرعت علي استكشاف عشرات الكائنات وطرق استخدامها لحظة كتابة هذه السطور التي تقرأها الآن.

PDF

لذلك، لا تفوت الفرصة على نفسك في استكشاف الشيفرات التي يولدها مصمم النماذج عندما تغير كل خاصية من خصائص الأدوات.

ملاحظة

لا اقصد بكلمة مشتركة (الخصائص المشتركة، الطرق المشتركة، والأحداث المشتركة) بالأعضاء المشتركة Shared Members. وإنما اقصد أعضاء مشتركة بين الأدوات Common Members فهي مشتقة من الفئة Control.

اسم الأداة Name

قد تجد الخاصية Name في أعلى نافذة الخصائص، مع ذلك من الخطأ اعتبارها خاصية، فالقيمة Name التي تضعها في هذه الخانة تمثل الاسم البرمجي للأداة، وعند تغيير الاسم سيقوم مصمم النماذج بتعديل كافة الشيفرات التي استخدمت اسم الأداة في الإجراء InitializeComponent().

نصيحة اخوية، لا تحاول تعديل اسم الأداة بعد كتابة الكثير من الشيفرات المصدرية، وذلك سيضطرك إلى تعديل جميع الشيفرات التي استخدمت الأداة بنفسك، حيث –كما قلت- ان مصمم النماذج يعدل الاسم البرمجي للأداة تلقائيا في الإجراء InitializeComponent() فقط.

نصيحة اخوية اخرى، لا تعتمد على الاسماء الابتدائية Text1، Text2، Text3 فهي ستسبب لك التشويش وإضعاف قدرة تذكرها، خاصة ان كثرت الأدوات على سطح النافذة.

لست بحاجة إلى تذكيرك ان شروط تسمية الأدوات يطبق عليها شروط تسمية باقي المعرفات الاخرى (فهي اسماء برمجية لكائنات)، ويمكنك العودة إلى الفصل الثاني **لغة البرمجة** لقراءة شروط التسمية.

خصائص المظهر

نبدأ بالخاصتين Visible و Enabled كلاهما يحمل قيمة منطقية Boolean تمثل الاول ظهور أو عدم ظهور الأداة، والثانية تمكين أو عدم تمكين الأداة، ان أسندت القيمة False للخاصية Enabled فستظهر الأداة بلون خافت يوحي بان المستخدم لن يستطيع استخدامها وبالتالي فلن تطلق الأداة اي أحداث وقت التنفيذ، مع ذلك لديك فرصة كبيرة في الوصول إلى الأداة برمجيا باستدعاء طرقها واسناد قيم لخصائصها.

PDF

الخاصية Text خاصية حرفية تمثل النص الظاهري على جبهة الأداة، نوع وحجم خط هذا النص يعتمد على الخاصية Font والتي سأتحدث عنها بعد قليل. بالنسبة للخاصية RightToLeft فتستطيع إسناد القيمة RightToLeft.Yes لها ان كنت تنوي إسناد حروف عربية على الأداة لتظهر في السياق **اليمين إلى اليسار Right-To-Left**. كما يفضل إسناد القيمة RightToLeft.Inherit لها إن أردت تغيير اتجاه السياق بشكل تلقائي بحيث يتوافق مع سياق نافذة النموذج أو الأداة الحاضنة للأداة. اما الخاصية TextAlign ففيها تحدد محاذاة النص، والذي يكون قيمة من تسع قيم (**شكل 14-1**).

TopLeft	TopCenter	TopRight
MiddleLeft	MiddleCenter	MiddleRight
BottomLeft	BottomCenter	BottomRight

شكل 14-1: القيم التسعة الممكنة للخاصية TextAlign.

ملاحظة

ان كان سياق الأداة في الخاصية RightToLeft من اليمين إلى اليسار، فسيتم عكس مواقع القيم xxxLeft و xxxRight للخاصية TextAlign. اي ستكون القيمة TopRight تمثل الزاوية العليا اليسري وليس اليمنى.

القيم التسع السابقة تابعة للتركيب ContentAlignment والذي يطبق على الأدوات Label، Button، CheckBox، و RadioButton فقط. اما الأدوات الاخرى فستتعامل مع ثلاث قيم هي: Right، Left، و Center تابعة للتركيب HorizontalAlignment.

اما خاصية الخط Font فهي تحمل قيمة تمثل كائن من النوع System.Drawing.Font يحتوي على مجموعة كبيرة من الخصائص والطرق والتي سأفصلها في الفصل القادم **GDI+**، يمكنك تعديل الخط عن طريق صندوق الحوار Font والذي يظهر زره على شكل ... في نافذة الخصائص، أو يمكنك انشاء كائن من الفئة System.Drawing.Font:

PDF

```
TextBox1.Font = New Font("Tahoma", 20, FontStyle.Bold Or _
                                        FontStyle.Italic)
```

مشكلة بسيطة حلها ابسط، وهي ان الكائن المسند إلى الخاصية Font لا يمكنك تعديل قيمه، فمعظم خصائص الفئة Font للقراءة فقط ReadOnly، لذلك عليك إسناد قيمة كائن جديدة باستخدام المعامل New:

```
' زيادة حجم الخط إلى الضعف
With TextBox1
    .Font = New Font(Font.Name, .Font.Size * 2, .Font.Style)
End With
```

ملاحظة

مشيد الفئة Font تم إعادة تعريفه Overloads بثلاث عشرة صيغة مختلفة. الصيغة التي استخدمتها في الشيفرتين السابقتين تستقبل 3 وسيطات الاولى اسم الخط، الثانية حجمه، والثالثة نمطه.

نقطة هامة علي توضيحها هي ان خاصية الخط Font لجميع الأدوات تشير إلى نفس كائن الخط التابع لنافذة النموذج أو الأداة الحاضنة ما لم تغير الخاصية Font لكل أداة محضونة، ماذا نستنتج من هذا الكلام؟ نستنتج ان اي تغيير لخاصية الخط التابعة لنافذة النموذج أو الأداة الحاضنة سيتم تغيير جميع الخطوط التابعة للأدوات المحضونة بها بشكل تلقائي. لذلك، إن أردت أن تكون الأداة محضونة مستقلة بخاصيتها Font (بحيث لا تتأثر ان تغيرت الأداة الحاضنة) وان تحمل نفس قيم الأداة الحاضنة بشكل ابتدائي، يمكنك **نسخ** الكائن باستخدام Clone():

```
TextBox1.Font = Form1.Font.Clone()
```

خصائص الموقع والحجم

قبل البدء في سرد خصائص الموقع والحجم، من الجيد تعريفك بمجموعة من الفئات الاولى System.Drawing.Point والتي تمثل نقطة تحتوي على خاصيتين للقراءة فقط ReadOnly هما الإحداثي السيني X (والذي يزيد كلما اتجهنا إلى اليسار) والإحداثي الصادي Y (الذي يزيد كلما اتجهنا إلى الاسفل)، والفئة الثانية System.Drawing.Size التي تحتوي على الخاصيتين للقراءة فقط أيضا هما Width و Height تمثلان عرض وارتفاع الأداة. اما الفئة الثالثة فهي

System.Drawing.Rectangle والتي **تمثل منطقة Client Region** تحتوي على مجموعة من الخصائص والطرق أيضا تشمل موقع، حجم، مساحة وغيرها من المعلومات والخاصة بالمنطقة.

ملاحظة

الوحدة المستخدمة بشكل افتراضي في القياسات هي البكسل Pixel والذي يمثل نقطة رسم واحد من الشاشة.

اسند قيمة من النوع Point إلى الخاصية Location لتحدد موقع الأداة بالنسبة لنافذة النموذج أو الأداة الحاضنة لها، واسند قيمة من النوع Size إلى الخاصية Size للتحكم في حجم الأداة، يمكن تعديل هذه الخصائص باستخدام الفأرة في مصمم النماذج أو كتابة الشيفرة التالية (والتي يولدها المصمم بشكل تلقائي):

```
' نقل الأداة إلى الزاوية العليا اليسرى
TextBox1.Location = New Point(0, 0)

' عرض الأداة 400 وارتفاعها 100 '
TextBox1.Size = New Size(400, 100)
```

ملاحظة

توجد خصائص قديمة هي Left، Top، Height، و Width صممت خصيصا للتوافقية مع الإصدارات (1->6) Visual Basic بل وأضيفت الخصائص Right و Bottom للتحكم في موقع وحجم الأداة. مع ذلك، لا أنصحك باستخدامها فهي أسلوب قديم ولن يقدم لك اي نتائج ايجابية عند الحديث عن تحسين الكفاءة Optimization.

من خصائص الموقع والحجم خصائص من النوع System.Drawing.Rectangle كالخاصية Bounds والتي تمثل المنقطة الكلية للأداة (شاملة حدودها وموقعها)، والخاصية ClientRectangle التي تمثل المنطقة الداخلية للأداة (متجاهلة حدودها وموقعها). الشيفرة التالية ستجعل الأداة تغطي كافة المنطقة الداخلية للنافذة:

```
TextBox1.Bounds = Form1.ClientRectangle
```

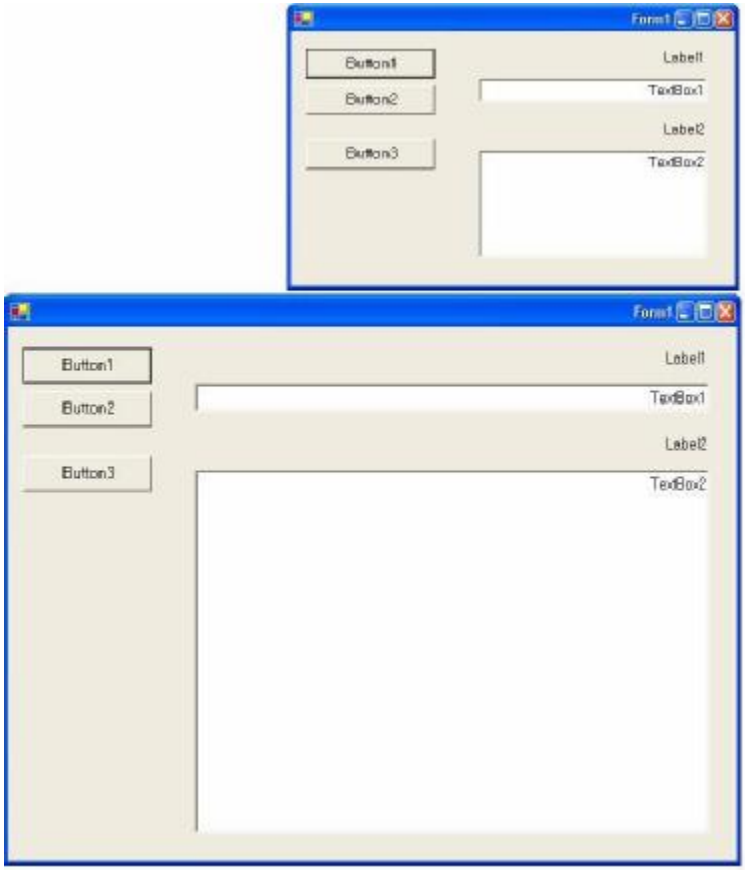
PDF

الخاصية Anchor:

بالنسبة للخاصية Anchor فهي خاصية مفيدة توفر لك كتابة عشرات الاسطر البرمجية لتنسيق موقع الأداة ان تغير حجم النافذة، حيث تثبت المسافة بين حدود الأداة وبين الحد الخارجي لنافذة النموذج أو الأداة الحاضنة في حالة ما تغير حجمها، يمكنك تثبيت المسافة من الاعلى، الاسفل، اليمين، أو اليسار باسناد القيم AnchorStyles.Top، AnchorStyles.Bottom، AnchorStyles.Right، أو AnchorStyles.Left. كما تستطيع الغاء تثبيت المسافة باسناد القيمة AnchorStyles.None.

ان قمت بتثبت المسافة لجهتين متضادتين (كفوق وتحت، أو يمين ويسار) فذلك سيضطر الأداة إلى تغيير حجمها حتى تثبت المسافة بين حدودها وبين الحد نافذة النموذج أو الأداة الحاضنة. وهذه فكرة رائعة جدا جدا لتمكين الأداة تحجيم نفسها بشكل تلقائي ان قام المستخدم بتغيير حجم النافذة (كما تفعل أداة الشجرة Tree في يسار مستكشف النظام Windows Explorer).

يمكنك الاستفادة من نتيجة التضادات السابقة بانشاء صناديق حوار قابلة للتمدد والتقلص كما ترى في (**الشكل 14-2 بالصفحة المقابلة**):



شكل 14-2: تأثير الخاصية Anchor.

حتى تعرف كيف استطعت انجاز صندوق الحوار السابق، فدعني اخبرك بان القيم كانت للأدوات كالتالي: TextBox1 هي: Top، Right، و Left، الأداة TextBox2 هـــي: Top، Bottom، Right، و Left، أدوات Label هي: Right و Top، اما الأدوات Button فكانت قيم خاصيتها Anchor هي: Left و Top.

الخاصية Dock:

اما الخاصية Dock فهي تمكنك من تغيير حجم وموقع الأداة بالنسبة لنافذة النموذج أو الأداة الحاضنة بحيث تحاذيها اما من اعلى DockStyle.Top كما تفعل اشرطة الأدوات ToolBars، من الاسف DockStyle.Bottom كما هو الحال مع سطر الحالة StatusBar (**شكل 14-3**)، من اليمين DockStyle.Right، من اليسار DockStyle.Left، أو تغطي كامل النافذة أو الأداة الحاضنة DockStyle.Fill.

شكل 14-3: تأثير الخاصية Dock.

تستطيع التحكم في المسافة بين الأداة التي تم محاذاتها والأداة الحاضنة لها عن طريق الخاصية DockPadding للأداة الحاضنة أو نافذة النموذج، يمكنك تحديد المسافة من الحد الايسر للأداة الحاضنة بهذه الطريقة:

```
Form1.DockPadding.Top = 10
```

وكافة الجهات الاخرى Right، Left، و Bottom بنفس الطريقة مع انك تستطيع إسناد قيم الجهات الاربع كلها بضربة واحدة:

```
' تعادل
' Form1.DockPadding.Top = 10
' Form1.DockPadding.Bottom = 10
' Form1.DockPadding.Left = 10
' Form1.DockPadding.Right = 10

Form1.DockPadding.All = 10
```

ملاحظة

الخاصية DockPadding مدعومة في الأدوات الحاضنة فقط، وهي الأدوات المشتقة من الفئة ContainerControl (**شكل** 13-4 **صفحة** 449).

خصائص الاحتضان

طريقة وخاصية الاحتضان ()FindForm و Parent تعودان بمرجع إلى نافذة النموذج الحاضنة للأداة في الطريقة ()FindForm (احذر فـ()FindForm طريقة Method وليست خاصية)، أو الأداة الحاضنة للأداة في الخاصية Parent. بالنسبة للخاصية Parent، فيمكن إسناد اي قيمة لها لتغيير الأداة الحاضنة للأداة الحالية (حتى لو اردت تغيير نافذة النموذج الحاضنة لها!):

```
Dim frmForm2 As New Form2

TextBox1.Parent = frmForm2
frmForm2.Show ()
```

مع ذلك، حتى وان نقلت الأداة إلى نافذة اخرى فلا تنسى ان الاسم البرمجي للأداة لا يزال يتبع إلى فئة نافذة النموذج الاصلية Form1.TextBox1 (حيث ان الأدوات قد عرفت في فئة النموذج لقنص أحداثها باستخدام WithEvents كما سترى لاحقا)، الا انك تستطيع إسناد مؤشر كائن اخر في النافذة الاخرى وتتبع الأداة).

لا تنسى ان الطريقة ()FindForm والخاصية Parent تمثل مرجع إلى نافذة النموذج الحاضنة أو الأداة الحاضنة لذلك يمكنك الوصول إلى طرقها وخصائصها بدون اي إشكالية:

```
TextBox1.FindForm().Test = "النافذة الحاضنة"
```

نافذة النموذج والأدوات الحاضنة الأخرى Panel و GroupBox مشتقة وراثيا من الفئة ContainerControl (**شكل 13-4 صفحة 449**)، لذلك تحتويان على خصائص إضافية تتعلق بالاحتضان، من هذه الخصائص الخاصية Controls والتي تمثل مرجع إلى جميع الأدوات المحضونة في الأداة الحالية.

ضع في عين الاعتبار ان الخاصية Controls عبارة عن مجموعة Collection تمثل الأدوات المحضونة في اعلى مستوى، والذي اقصده من العبارة اعلى مستوى اي أدوات

PDF

المحضونة بها فقط ولا تشمل الأدوات المحضونة بتلك الأدوات المحضونة في الأداة نفسها، وحتى أسهل لك استيعاب الفكرة افترض هذه الأدوات الحاضنة والمحضونة:

```
Form1
    GroupBox1
        TextBox1
        TextBox2
        TextBox3
    GroupBox2
        TextBox4
        TextBox5
```

الخاصية Form1.Controls ستعود بمرجع يمثل الأداتين GroupBox1 و GroupBox2 فقط، والخاصية GroupBox1.Controls ستعود بمرجع يمثل الأدوات TextBox1، TextBox2، و TextBox3، بينما الأداتين TextBox4 و TextBox5 هي المراجع التي ستعود بها الخاصية GroupBox2.Controls.

خصائص الألوان

الخاصيتان ForeColor و BackColor تمثلان لون الأمامية والخلفية للأداة، لون الامامية – بالتحديد- هو لون النص الظاهر على الأداة، ولون الخلفية هو لون سطح الأداة. الالوان في عالم Windows بشكل عام و NET. بشكل خاص تصنف إلى قسمين رئيسين هما: **الوان النظام System Colors، و الالوان الخاصة Custom Colors**.

الوان النظام هي الالوان المفضلة في معظم الاحوال، فهي الوان حددها المستخدم في لوحة التحكم Control Panel لتظهر به سائر نوافذ وأدوات تطبيقات Windows بها. لذلك فمن باب احترام وتقدير ذوق المستخدم استخدام نفس الالوان التي طلبها. تجد هذه الالوان في الفئة System.Drawing.SystemColors:

```
TextBox1.BackColor = SystemColors.Window
TextBox2.BackColor = SystemColors.ActiveBorder
```

اما الالوان الخاصة فهي ستاتيكية لا تتغير مهما كانت اعدادات نظام التشغيل، وان لم تكن مصمم راقي فأتمنى من صميم قلبي ان تتجاهل هذه الفقرة، اما ان كنت ممن اتخذ في فن الرسم سبيلا، فيمكنك الاعتماد على الفئة System.Drawing.Color وتحديد الالوان باسمها:

```
TextBox1.BackColor = Color.Black
TextBox1.ForeColor = Color.White
```

أو تحديد العمق اللوني للأحمر، الأخضر، والأزرق باستدعاء الطريقة FromArgb() والتابعة لنفس الفئة السابقة:

```
TextBox1.BackColor = Color.FromArgb (255, 0, 0)
```

ملاحظة

الخاصيتان BackColor و ForeColor تشيران إلى نفس خصائص نافذة النموذج أو الأداة الحاضنة (حالها كحال الخاصية Font). لذلك، اي تأثير على خصائص الأدوات الحاضنة سيشمل الأدوات المحضونة ما لم تسند قيم خاصة للأدوات المحضونة وقت التصميم.

خصائص التركيز

اقصد بكلمة التركيز Focus هي قدرة الأداة على ان تكون الأداة النشطة Active Control ويكون التركيز Focus عليها (كأداة TextBox عندما تبدأ الكتابة بها). يمكنك معرفة ما اذا كانت الأداة قابلة لاستقبال التركيز عليها عن طريق الخاصية CanFocus القابلة للقراءة فقط ReadOnly. ويمكنك معرفة ما اذا كان الأداة هي الأداة النشطة وعليها التركيز فعلا عن طريق الخاصية Focused وهي للقراءة فقط أيضا.

الأداة التي عليها التركيز تسمى الأداة النشطة، يمكن للأداة (والحديث هنا عن الأداة الحاضنة أو نافذة النموذج) من معرفة الأداة النشطة والمحضونة فيها عن طريق الخاصية ActiveControl والتي تعود بمرجع يمثل الأداة النشطة (الخاصية قابلة للكتابة أيضا).

خصائص الجدولة

معظم مستخدمي تطبيقات Windows يفضلون استخدام مفتاح الجدولة [TAB] لنقل التركيز بين الأدوات، معظم الأدوات التي لها قابلية انتقال التركيز Focus عليها تحتوي على الخاصيتين TabStop و TabIndex. أسند القيمة True إلى الخاصية TabStop إن أردت من الأداة ان تستقبل التركيز بمجرد ان يضغط المستخدم على المفتاح [TAB]، ثم حدد ترتيب وتسلسل الأدوات في الخاصية TabIndex مع العلم ان الترقيم يبدأ بصفر.

ملاحظة

حتى لو أسندت القيمة False إلى الخاصية TabStop، فان للمستخدم فرصة كبيرة لنقل التركيز على الأداة بالنقر عليها بزر الفأرة.

خصائص أخرى

من الخصائص الاخرى التي اود ذكر أسمائها فقط الخصائص Created، Disposing، و Disposed والتي تعود بالقيمة True في حال ما –على التوالي- تم انشاء الأداة فعلا، الأداة على وشك الموت، الأداة ماتت فعلا.

يمكنك إسناد قيمة تمثل قائمة Menu إلى الخاصية ContextMenu يتم عرضها على شكل قائمة منبثقة Pop-up menu.

هناك خصائص اخرى تفيدك في حالة قيامك بتطوير أدوات خاصة Custom Controls كالخاصيتين ProductName، ProduceVersion، و CompanyName.

لديك خاصية الاقفال Locked (والتي تستخدم لحظة التصميم فقط من قبل مصمم النماذج) حيث تثبت الأداة وتمنعك من تحريكها أو تغيير حجمها بطريق الخطأ.

اخيرا، مجموعة من الخصائص موجه لذوي الاحتياجات الخاصة (عافاني الله وإياك) هي AccessibleDescription، AccessibleName، AccessibleRole، و IsAccessible - راجع مكتبة MSDN لكيفية الاستفادة منها عسى ان لا تحتاجها يوما من الايام.

الطرق المشتركة

معظم الطرق ليست سوى نسخ مكررة من الخصائص، فمثلا الطريقة SetSize() ترسل مع وسيطاتها عرض وارتفاع الأداة، كذلك الطريقة SetBounds() والتي تشمل في وسيطاتها الأحداثي السيني والصادي للأداة وعرضها وارتفاعها بضربة واحدة.

لديك الطريقتان BringToFront() و SendToBack() التي تظهر الأداة فوق الأدوات الاخرى أو خلف الأدوات الاخرى، مع العلم ان الأدوات المحضونة تكون دائما فوق الأداة الحاضنة.

الطريقة Show() تظهر الأداة (تعادل القيمة True للخاصية Visible)، والطريقة Hide() تخفي الأداة (تعادل القيمة False للخاصية Visible).

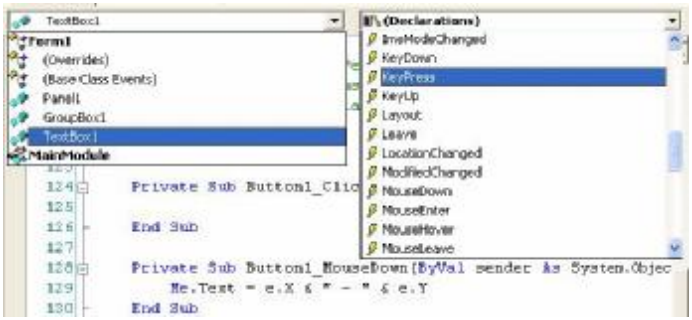
لو تذكر ان بعض خصائص الأدوات تتأثر بقيمة خصائص الأداة الحاضنة لها (كالخصائص ForeColor و BackColor) ما لم تتغير قيمة الخاصية للأداة، يمكنك استرجاع قيم الخصائص لتشير إلى كائنات الأدوات الحاضنة إما باسناد قيم الكائنات بنفسك، أو استدعاء الطرق ()ResetBackGround و ()ResetForeColor.

يمكنك توجيه التركيز إلى الأداة (القابلة لاستقبال التركيز) باستدعاء الطريقة ()Focus، كما تستطيع معرفة الأداة التالية والتي سيأتي دورها في الخاصية TabIndex لمفتاح الجدولة [TAB] باستدعاء الطريقة ()GetNextControl والتي تعود بمرجع للأداة التالية.

اخيرا، الطرق ()Invalidate، ()Refresh، و ()Update تقوم بإعادة رسم الأداة، بامكانك مراجع مكتبة MSDN لتوضيح الفروق بينها.

الأحداث المشتركة

في هذا القسم نعرض مجموعة كبيرة من الأحداث بشكل سريع. وكما هو الحال مع النماذج، يمكنك الاستعانة بمحرر الشيفرة لقنص الأحداث دون الحاجة لكتابتها بنفسك. ولكن هذه المرة باختيار اسم كائن الأداة من القائمة العلوية اليمنى، ومن ثم الحدث المراد قنصه من القائمة العلوية اليسرى **(شكل 14-4)**.



شكل 14-4: الاستعانة بمحرر الشيفرة لقنص أحداث الأدوات.

أحداث الفأرة

معظم شيفراتك المصدرية يتم تنفيذ نتيجة لأعمال درامية قام بها مستخدم البرنامج بالفأرة، من هذه الأحداث الحدث Click والذي يتم تنفيذه لحظة النقر على الأداة والحدث DblClick لحظة النقر المزدوج.

يمكنك معرفة المزيد من التفاصيل حول الفأرة بالاستعانة بالحدث MouseDown والذي يتم تنفيذه بمجرد الضغط بزر الفأرة على الأداة (النقر هي عملية الضغط بزر الفارة ومن ثم تحريره، بينما الضغط لا يشترط تحرير الزر لتنفيذ الحدث)، من هذه التفاصيل الأزرار التي تم ضغطها بالفأرة في الخاصية Button لوسيطة الحدث:

```
Private Sub Button1_MouseDown(ByVal sender As Object, _
    ByVal e As Forms.MouseEventArgs) Handles Button1.MouseDown

    If e.Button = MouseButtons.Left Then
        ' الزر الايسر
        …
        …
    ElseIf e.Button = MouseButtons.Middle Then
        ' الزر الاوسط
        …
        …
    ElseIf e.Button = MouseButtons.Right Then
        ' الزر الايمن
        …
        …
    End If
 End Sub
```

كما تستطيع معرفة احداثيات موقع الفأرة إما من الأداة (باستخدام الخصائص X و Y التابعة لوسيطة الحدث) أو بالنسبة للنافذة (باستخدام الخاصية المشتركة Control.MousePosition):

```
Private Sub Button1_MouseDown(ByVal sender As Object, _
    ByVal e As Forms.MouseEventArgs) Handles Button1.MouseDown

    Me.Text = e.X & " - " & e.Y
End Sub
```

من التفاصيل التي تحصل عليها من وسيطة الحدث MouseDown، عدد مرات النقر منذ اخر تنفيذ للحدث في الخاصية Click، والخاصية Delta التي تمكنك من معرفة مقدار تحرك عجلة الفأرة Mouse Wheel.

PDF

وسيطة الحدث MouseDown هي من النوع MouseEventArgs، وهي نفس الوسيطة التي يتم ارسالها إلى الأحداث MouseUp، MouseMove، و MouseWheel. يتم تنفيذ الحدث الاول في حالة تحرير زر الفأرة، الثاني عن تحريك الفارة فوق الأداة، والثالث عن تحريك عجلة الفأرة على الاداة في حل ما إذا كان التركيز عليها.

العدد الكلي لأحداث الفأرة هو تسع أحداث وقد ذكرنا ستا منها، اما الثلاثة الباقية MouseEnter، MouseLeave، MouseHover فيتم تفجيرها مع بداية دخول الفارة على الأداة، خروج الفارة عن حدود الأداة، أو عند تحريك الفأرة على الأداة (كما هو الحال مع الحدث MouseMove).

أحداث لوحة المفاتيح

ثلاثة أحداث مرنة توفرها لك الأدوات تتعلق بلوحة المفاتيح هي: KeyDown، KeyUp، و KeyPress. بالنسبة للحدث الثالث فيتم تنفيذه لحظة النقر على مفتاح من مفاتيح لوحة المفاتيح، ترسل وسيطة هذا الحدث خاطيتين هما KeyChar و Handled، الخاصية الاولى تمثل الحرف الذي تم النقر عليه، ويمكنك إسناد القيمة True إلى الخاصية الثانية إن أردت إخبار إطار عمل NET. انك قمت بعمل اللازم لردة الفعل على هذا المفتاح ولا تريد منه أي يقوم بأي عمل إضافي، بمعنى اخر اسنادك للقيمة True إلى هذه الخاصية يلغي عملية النقر وكأن شيئا لم يحدث. (تفيدك بكثرة مع الأداة TextBox إن أردت الغاء مدخلات المستخدم الغير مناسبة).

بالنسبة للحدثان KeyDown و KeyUp فيتم تنفيذهما بمجرد قيام المستخدم بالضغط على الزر و تحريره، ضع في عين الاعتبار ان الحدث KeyDown سيتم تنفيذه بشكل متكرر طالما كان الزر مضغوطا.

يرسل كلا الحدثين وسيطة من النوع KeyEventArgs تحتوي على الخصائص Alt، Shift، و Control لتعود بقيم تبين حالة ضغط المفاتيح [Alt]، [Shift]، و [Ctrl] على التوالي. كما تحتوي على الخاصية KeyCode التي تمثل المفتاح بقيمته الفيزيائية الذي تم ضغطه (نوع القيمة هذه تركيب Enum بالاسم Keys)، كما تحتوي الوسيطة على الخاصية Handled والتي مثل الخاصية Handled التابعة لوسيطة الحدث KeyPress.

فرق اخر بين الحدث KeyPress والحدثين KeyUp و KeyDown وهو ان الحدث KeyPress يتم تنفيذه ان ضغط المستخدم على الحروف أو الارقام المطبوعة، المفتاح [Enter]، المفتاح الجدولة [Tab]، والمفتاح [Esc] فقط، اما باقي المفاتيح كمفاتيح الوظائف [F1]، [F2]،

...الخ مفاتيح الاسهم، المفاتيح [Shift]، [Ctrl]، [Alt] ...الخ فالحدث KeyDown لها بالمرصاد.

يوجد حدث اضافي يماثل الحدث KeyPress السابق وهو HelpRequested والذي يتم تنفيذه في حال ما قام المستخدم بالنقر على المفتاح [F1].

اخيرا، لو تم كتابة شيفرات في أحداث لوحة المفاتيح لكلا الأداة ونافذة النموذج، واردت معرفة ايهما سيبدأ في ردة الفعل، فستكون نافذة النموذج هي الأولى ان كانت خاصيتها KeyPreview تساوي True، بينما تستجاب أحداث الأداة اولا ان كانت قيمة الخاصية False.

أحداث التركيز

عندما تستقبل الأداة التركيز، فسيتم تنفيذ حدثها الخاص وهو GotFocus، وعلى العكس ان فقدت الأداة التركيز سيتم تنفيذ حدثها LostFocus.

تفضل مستندات Microsoft .NET Documentation بالاعتماد على الحدثين Enter و Leave، يتم تنفيذ الأول بمجرد وقوع التركيز على الأداة (قبل الحدث GotFocus)، والثاني بعد فقد الأداة تركزيها (قبل الحدث LostFocus).

السبب الذي يجعلك تفضل استخدام الحدثين Enter و Leave عن GotFocus و LostFocus تجنب التشويش المنطقي في ترتيب وقوعها في سلسلة أحداث التركيز، حيث ترتيب السلسلة يكون كالتالي:

Enter -> GotFocus -> Leave -> Validating -> Validated.

بالنسبة للحدثين Validating و Validated فيفيدانك كثيرا عند التعامل مع الأداة TextBox، لذلك فضلت تأجيل مثال لاستخدامهما عند فقرة **الأداة TextBox** لاحقا في هذا الفصل.

أحداث أخرى

من الأحداث أيضا، حدث الرسم Paint -الذي ذكرته في الفصل السابق **نماذج Windows Forms**- والذي يتم تفجيرها كلما دعت الحاجة إلى اعادة رسم الأداة، والحدث Resize الذي يتم تنفيذه بمجرد تغيير حجم الأداة، والحدث Move الذي يتم تنفيذه ان تم تحريك الأداة (يفيدك الحدث الاخير كثيرا مع نوافذ النماذج).

PDF Converter

يوجد حدثان اضافيان يتعلقان بالأدوات الحاضنة هما ControlAdded و ControlRemoved حيث يتم تنفيذ الاول في كل مرة تضيف أداة محضونة إلى الأداة الحاضنة، والثاني إن أزلت الأداة المحضونة من الأداة الحاضنة.

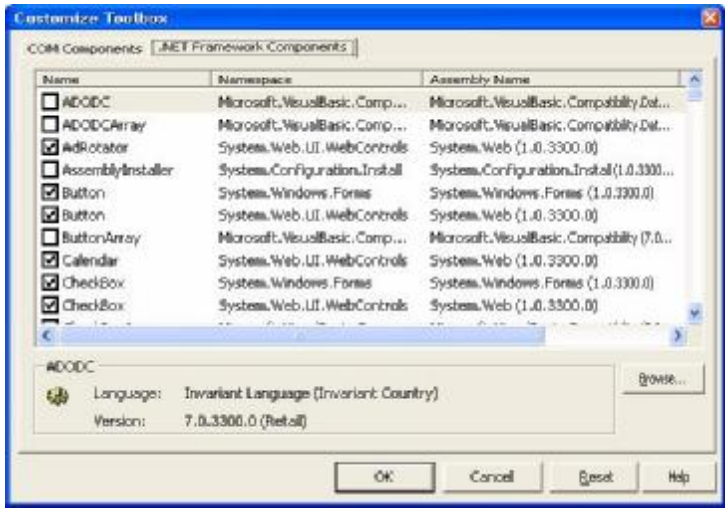
اخيرا، الحدث PropertyChanged يتم تنفيذه في كل مرة تقوم بتغيير احد خصائص الأداة، يرسل هذا الحدث مع وسيطته الخاصية PropertyName والتي تمثل اسم الخاصية التي تم تعديلها.

عرض سريع للأدوات

توجد العشرات من الأدوات التي تأتي بشكل ابتدائي مع رزمة Visual Studio .NET تجدها في نافذة صندوق الأدوات ToolBox (**شكل 15-5**) والذي تصل اليه باختيار الامر ToolBox من قائمة View، اعلم ان الأدوات في شريط الأدوات لا تظهر الا ان كانت نافذة مصمم النماذج Form Desinger مفتوحة وظاهرة امام عينك.

شكل 14-5: نافذة صندوق الأدوات ToolBox.

يمكنك تحديد هذه الأدوات والبدء في رسمها على جبهة نافذة مصمم النماذج، كما تستطيع حذف أو إضافة أدوات جديدة بالضغط بزر الفأرة الايمن على صندوق الأدوات، واختيار الامر Customize ToolBox من القائمة المنبثقة، ليظهر لك صندوق الحوار Customize Toolbox **(شكل 14-6)**.



شكل 14-6: صندوق الحوار Customize ToolBox.

يحتوي صندوق الحوار Customize ToolBox على خانتي تبويب Tab، الاول تعرض لك كافة أدوات التحكم ActiveX Controls والتي تعتمد في بنيتها التحتية على تقنية COM القديمة، ومعظم الأدوات المعتمدة على تقنية COM لا تعمل بشكل صحيح مع تقنيتنا الجديدة NET..

بالنسبة لخانة التبويب الثانية NET Framework Component. تعرض لك أدوات التحكم الخاصة Custom Controls والذي قد يكون احد مصمميها انت –كما سترى لاحقا.

في هذا القسم من الفصل سنحتك مع معظم الأدوات التي ظهرت لك بشكل ابتدائي في صندوق الأدوات ToolBox لحظة انشاء مشاريع من النوع Windows Application، لنعرض لك ابرز خصائصها، طرقها، وأحداثها. وكالعادة، يمكنك العودة إلى وثائق ومستندات NET. Documentation إن أردت المزيد من التفاصيل حول أعضاء هذه الأدوات، حيث انني لن اعرضها هنا بالا بشكل سريع ومختصر.

الأداة Label

تستخدم هذه الأداة البسيطة في عرض النصوص على النوافذ، يمكنك محاذاة النص باستخدام الخاصية TextAlignment باسناد قيمة لها من 6 قيم (**شكل 14-1**)، كما تحتوي على الخاصيتين PreferredWidth و PreferredHeight التي تعود بأفضل حجم يناسب الأداة اعتمادا على نوع وحجم الخط المستخدم.

توجد الخاصية FlatStyle التي تمكنك من تغيير شكل الحد الخارجي للأداة، كما يمكنك عرض صورة في الأداة عن طريق الخاصية Image ويمكنك التحكم في موقع الصورة عن طريق الخاصية ImageAlign.

أسند القيمة True إلى الخاصية UseMnemonic لتمكن المستخدم من نقل التركيز إلى الأداة التي تلي أداة Label الحالية في ترتيب TabIndex عند قيامه بالضغط على المفتاح [Alt] و الحرف الذي يتبع للحرف **&** في الخاصية Text، مع العلم انه سيظهر خط سفلي تحت الحرف الذي يلي حرف &. وإن أردت عرض الحرف & على جبهة الأداة في هذه الحالة، اكتبه مرتين متتاليتين &&.

الأداة LinkLabel

الأداة LinkLabel نسخة مطورة من الأداة Label السابقة، تمكنك من وضع روابط كالروابط الموجودة في صفحات HTML، بحيث تمكنك من كتابة شيفرات ردة فعل على هذه الروابط.

توجد طرقتين لوضع الأداة، الطريقة السريعة ممكنة في وقت التصميم لتعرض لك رابط واحد في كامل النص تحدد في الخاصية LinkArea، فلو كان قيمة الخاصية Text هي "مرحبا بكم في شبكة المطورون العرب" يمكنك وضع قيمة البداية 13 والحجم 19 في الخصائص Start و Length التابعة للخاصية LinkArea.

اما الطريقة الثانية فهي ممكن وقت التنفيذ فقط، بحيث تمكنك من وضع مجموعة من الروابط في نفس نص الأداة عن طريق الخاصية Links والتي تمثل مجموعة Collection:

```
LinkLabel1.Text = "يمكنك الضغط هنا أو الضغط هنا"
LinkLabel1.Links.Add(12, 3, "Link1")
LinkLabel1.Links.Add(25, 3, "Link2")
```

سواء استخدمت الاولى أو الثانية، تستطيع الاستفادة من الحدث LinkClicked لكتابة شيفرات ردة الفعل عند النقر على احد روابط الأداة:

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
    Handles LinkLabel1.LinkClicked

    Select Case e.Link.LinkData()
        Case "Link1"
            MsgBox("تم نقر الرابط الاول")
        Case "Link2"
            MsgBox("تم نقر الرابط الثاني")
    End Select

End Sub
```

الأداة TextBox

تعتبر الأداة TextBox الوسيلة المثلى لقنص المدخلات من المستخدمين، واستخدامها يعتبر جزئا لا يتجزأ من اي نافذة نموذج موجه لاستقبال المعلومات والبيانات، النص الظاهر في وسط الأداة هو نفس النص الموجود في الخاصية Text، وعند اي تغيير لهذه الخاصية سيتم تنفيذ الحدث TextChanged. كما يمكنك منع المستخدم من تحرير الأداة باسناد القيمة True إلى الخاصية ReadOnly –رغم ان فرصة تغيير محتوياتها برمجيا ممكنة.

يمكنك تحديد نص معين من النص الظاهر وسط الأداة عن طريقة الخاصيتين SelectionStart و SelectionLength، الاولى تحدد فيها نقطة البداية والثانية عدد الحروف، الشيفرة التالية تحدد جميع الحروف في أداة النص:

```
TextBox1.SelectionStart = 0
TextBox1.SelectionLength = TextBox1.Text.Length
```

ان كان النية تحديد جميع الحروف في أداة النص، فالشيفرة التالية يمكنك تقليصها إلى سطر واحد وذلك باستدعاء الطريقة SelectAll() التي تحدد كامل النص، كما يمكنك معرفة النص المحدد عن طريق الخاصية SelectedText.

المزيد أيضا، ارسل قيمة مع الطريقة AppendText لإضافة نص في نهاية النص الحالي، كما توجد الخاصية AutoSize لتحجيم الأداة لتتناسب مع حجم ونوع الخط بشكل تلقائي. ويمكنك استدعاء الطريقة Undo() لاعادة القيمة الاخيرة في أداة النص، تشترط هذه الطريقة ان تكون قيمة الخاصية CanUndo هي True، وعليك التحقق من ان الاداة قد تم تعديلها فعلا عن طريقة الخاصية Modified.

يمكنك تحديد العدد الأقصى للحروف الممكن إدخالها في أداة النص عن طريقة الخاصية MaxLength، كم تستطيع إسناد حرف إلى الخاصية PasswordChar إن أردت استخدامه ليتم عرضه مهما كانت الحروف (يفيدك حرف النجمة * لمحاكاة كلمات المرور).

ملاحظة

ان استخدمت الخاصية PasswordChar، فإن ذاكرة أداة النص TextBox كفيلة بإلغاء أوامر النسخ Copy والقص Cut التابعة للقائمة المنبثقة والتي تظهر ان قام المستخدم بالضغط بزر الفأرة الأيمن على الأداة. اما ان أنشأت قوائم نسخ ولصق بنفسك، فذاكرتك هي المسئول الاول والاخير عن إلغائها.

عند الحديث عن الحروف الإنجليزية، يمكنك إسناد القيمة Upper للخاصية CharacterCasing حتى تجعل جميع الظهور كبيرة Capital واسناد القيمة Lower لتتحول الحروف إلى صغيرة.

اما الحديث عن الحروف العربية، فدعني اذكرك بان الحروف المتداخلة (مثل: لا، لأ، لآ ...الخ) هي عرفين مستقلين، كذلك الحال مع علامات التشكيل والتنوين (َ ً ِ ٍ ُ ٌ ْ)، لذلك ضع استقلالية الحروف في الاعتبار عند التعامل مع خصائص الأداة.

أداة متعددة السطور:

إسناد القيمة True للخاصية Multiline يحول الأداة إلى أداة متعدد السطور Multi-Line Textbox بحيث تمكن المستخدم من كتابة سطور متعددة في نفس الأداة.

تستطيع إسناد القيمة True أيضا إلى الخاصيتين AcceptsReturn و AcceptsTab إن أردت التعامل مع المفاتيح [Enter] و [Tab] كحروف، فكما تعلم ان المفتاح [Enter] يقوم بتنفيذ زر الاوامر الافتراضي (ان وجد) في نافذة النموذج، والمفتاح Tab ينقل التركيز إلى الأداة التالية في الخاصية TabIndex.

PDF Factory Pro trial version www.fastio.com

ملاحظة

الزر الافتراضي Default Button هو الزر الذي عليه حد إضافي خارجي يتم تنفيذه بمجرد الضغط على المفتاح [Enter] من قبل المستخدم على نافذة النموذج –كما سترى في الفقرة القادة **الأداة** Button. مع ذلك، ان لم يكن هناك زر افتراضي في النافذة فلست بحاجة إلى استخدام الخاصية AcceptsReturn.

يمكنك إضافة اشرطة تمرير للأداة متعددة السطور عن طريق الخاصية ScrollBars، مع العلم ان شريط التمرير الأفقي H ScrollBar لن يظهر الا ان كانت قيمة خاصية الالتفاف WordWrap تساوي False .

يمكنك الاستفادة من الخاصية Lines والتي تعود بمصفوفة تمثل قيم السطور المختلفة في الأداة، كما تستطيع استدعاء الطريقة ()ScrollToCaret لتحريك أشرطة التمرير بحيث تظهر لك الجزء المحدد من النص.

التحقق من المدخلات:

الحدثين Validating و Validated سيفيدانك كثيرا عند التعامل مع الأداة TextBox، وذلك لانهما يعتبران انسب مكان للتحقق من المدخلات في أداة النص.

سيناريو تنفيذ الحدثان يكون كالتالي: عندما ينتقل التركيز من الأداة X إلى الأداة Y، سيتم التحقق من الخاصية CausesValidation لكلا الأداتين، ان كانت قيمة الخاصيتين False فلن يحدث شيء، وان كانت قيمة الخاصية True فسيتم تنفيذ الحدث Validating التابع للأداة X، وان قمت باسناد القيمة True إلى الخاصية Cancel التابعة لوسيطة الحدث Validating:

```
Private Sub X_Validating(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles X.Validating

    If X.Text = "" Then e.Cancel = True
End Sub
```

فسيعود التركيز إلى الأداة X ولن يتم تنفيذ الحدث Validated التابع لنفس الأداة (ولا حتى الحدث LostFocus أيضا)، اما ان لم تفعل شيئا للخاصية Cancel السابقة، فسيتم تنفيذ الحدث Validated ومن ثم نقل التركيز إلى الأداة Y.

الأداة Button

استخدام هذه الأداة معروف وسهل جدا حتى لمستخدمين Windows العاديين، وهو زر يتم ضغطه لتنفيذ أوامر معينة. لا يوجد الكثير لأخبرك به حول هذا الزر سواء وجود خاصيتين تابعة لنافذة النموذج تؤثران تاثير بسيط على هذا الزر هما AcceptButton و CancelButton، تحدد في الاولى الزر الذي تود رسم حوله حد اضافي يخبر المستخدم بان المفتاح [Enter] يؤدي إلى تنفيذ هذا الزر، والخاصية الثانية مرافقة للمفتاح [Esc]، يمكنك تعديل هذه الخصائص لنافذة النموذج وقت التصميم من نافذة الخصائص، أو وقت التنفيذ بهذه الشيفرة:

```
Me.AcceptButton = Button1
Me.CancelButton = Button2
```

الأداة CheckBox

تملأ هذه الأداة اغلب تطبيقات Windows، يمكنك تحديد ما اذا كانت الأداة مختارة باسناد القيمة True إلى الخاصية Checked و False لالغاء الاختيار، عند اسنادك للقيمة True للخاصية السابقة، يمكنك تحديد القيمة Indeterminate للخاصية CheckState والتي تماثل ما بين نعم و لا.

عندما يقوم المستخدم بالنقر على الأداة سيتم عكس قيمة خاصيتها Checked بشكل تلقائي، مع ذلك تستطيع منع هذا التغيير باسناد القيمة False إلى الخاصية AutoCheck، لتحصر المسؤولية عليك في كتابة الشيفرات البرمجية واللازمة لتغيير قيمة الخاصية Checked.

الخاصية CheckAlign مثل الخاصية TextAlign (**شكل 14-1**) تماما، ويكمن الفرق في ان الاولى خاصة بموقع رمز المربع ☑ فقط، بينما الثانية فخاصة بالنص المرافق لرمز المربع.

كما ذكرت قبل قليل، عندما يقوم المستخدم بالنقر على الأداة فسيتم عكس قيمة الخاصية Checked، ولكن عن إسناد القيمة True للخاصية ThreeState فلن يتم عكس قيمة الخاصية Checked إلى كل نقرتين، النقرة الاولى تجعل الخاصية Checked هي True والثانية تضيف القيمة القيمة Indeterminate للخاصية CheckState، اما النقرة الثالثة فتعكس قيمة الخاصية Checked وهكذا ...

PDF

ملاحظة

لن تعمل الخاصية ThreeState إلا إن كانت قيمة الخاصية AutoCheck هي True.

اخيرا، تحتوي الأداة CheckBox على الحدث CheckedChanged والذي يتم تنفيذ ان تم تغيير قيمة الخاصية Checked أو CheckState.

الأداة RadioButton

وجه الشبه بين هذه الأداة والأداتين التي قبلها (CheckBox و Button) هو ان كلاهم مشتق وراثيا من الفئة ButtonBase والتي تحتوي على خصائص اضافية كـ FlatStyle لتحديد شكل ثلاثي الأبعاد 3D والخاصية Appearance التي تمكنك من استخدام شكل الزر Button مع الأداتين CheckBox و RadioButton.

يمكنك إسناد القيمة True للخاصية Checked التابعة لهذه الأداة لاختيارها، مع العلم ان باقي الأدوات في نفس المجموعة (نفس الأدوات المحضونة في الأداة الحاضنة) سيتم إسناد القيمة False لخصائصها Checked,

الأداة ListBox

إن أردت التعامل مع الأداة بشكل عام، فانك تستخدم كائنها المنشئ من الفئة ListBox، اما إن أردت التعامل مع **العناصر** الموجودة في الأداة، فوجه أنظارك إلى الخاصية Items والتي عبارة عن مجموعة Collection تمثل عناصر الأداة ListBox.

بما ان الخاصية Item عبارة عن مجموعة Collection، فهي تحتوي على الواجهة ICollection فيمكنك استخدام الطرق التقليدية لإضافة، حذف، والاستعلام عن العناصر (كـ Add()، Insert()، Clear()، Count() ...الخ):

```
' إضافة عناصر
Dim counter As Integer

For counter = 1 To 10
    ListBox1.Items.Add(counter)
Next

' حذف جميع العناصر
ListBox1.Items.Clear()
```

```
' الاستعلام عن العناصر
Dim item As Object
For Each item In ListBox1.Items
    …
    …
Next
```

انظر أيضا

يمكنك مراجعة الفصل السادس **الفئات الأساسية** لمزيد من التفاصيل حول الواجهة ICollection.

العناصر التي تضيفها لا يشترط ان تكون بيانات ابتدائية Primitave Types (كـ String، Integer، Double ...الخ) بل يمكن ان تشكل كائنات لفئات تعرفها بنفسك:

```
Class Person
    Public Name As String
    Public Age As Integer

    Sub New(ByVal name As String, ByVal age As Integer)
        Me.Name = name
        Me.Age = age
    End Sub
End Class

…
…

With ListBox1.Items
    .Add(New Person("تركي", 99))
    .Add(New Person("عباس", 3000))
End With
```

السؤال الذي يطرح نفسه بقوة الان، ما هو النص الذي سيظهر في عناصر الأداة ListBox، والجواب هو اسم الفئة الذي تعود به الطريقة ()ToString والتابعة للكائن. نستنتج من هذا، أننا نستطيع تحديد القيمة التي نريدها بفضل اعادة القيادة Overriding:

```
Class Person
    …
    Overrides Function ToString() As String
        Return Me.Name
    End Function
End Class
```

PDF

يستطيع المستخدم تحديد اكثر من عنصر من عناصر أداة ListBox عن طريقة الخاصية SelectionMode والتي تسند لها اما القيمة MultiSimple أو القيمة MultiExtended (على المستخدم استخدام المفاتيح [Ctrl] أو [Shift] في حالة إسناد القيمة الثانية)، كما يمكن للمبرمج من الاستعلام عن جميع العناصر المحددة عن طريقة الخاصية SelectedItems:

```
Dim x As String

For Each x In ListBox1.SelectedItems
    …
    …
Next
```

وعلى ذكر التحديد، يمكن للمبرمج أيضا من تحديد/الغاء تحديد العنصر برمجيا باستدعاء الطريقة SetSelected()، والتي ترسل معها القيمة True لتحديد العنصر أو False لالغاء التحديد:

```
ListBox1.SetSelected(0, True)
ListBox1.SetSelected(1, False)
```

اخيرا، عند إضافة مجموعة كبيرة من العناصر، فينصح دائما باستدعاء الطريقة BeginUpdate() قبل إضافة العناصر والطريقة EndUpdate() بعد إضافتها، وذلك تمنع الأداة من اعادة رسمها في كل مرة تضيف عنصر جديد مما يزيد سرعة الإضافة أضعاف المرات:

```
Dim counter As Integer

ListBox1.BeginUpdate()
For counter = 0 To 10000
    ListBox1.Items.Add(counter)
Next
ListBox1.EndUpdate()
```

الأداة CheckedListBox

الأداة CheckedListBox هي نسخة محسنة من الأداة ListBox وهي مشتقة وراثيا منها، فكل ما ذكرته في السطور السابقة يطبق على هذه الأداة باستثناء الخاصية SelectionMode حيث لن تتمكن من استخدامها كما فعلت مع الأداة ListBox، وذلك ان طريقة تحديد العناصر تعتمد على أدوات شبيهه بالأداة CheckBox، اما إن أردت معرفة العناصر التي تم تحديدها فاستخدم الخاصية SelectedItems عوضا عن الخاصية SelectedItems.

PDF

الأداة ComboBox

الأداة ComboBox ما هي الا أيضا نسخة محسنة من الأداة ListBox السابقة وتحتوي على أداة TextBox، ولكنها ليست مشتقة منها، مع ذلك معظم الطرق والخصائص التابعة للأداة ListBox والأداة TextBox مدعومة في الأداة ComboBox أيضا.

يمكنك تغيير شكل الأداة ComboBox عن طريق الخاصية DropDownStyle والتي تكون قيمة من ثلاث قيم هي: DropDown، Simple، و DropDownList، في الاولى تمكنك المستخدم من تحرير النص في خانة النص اما الثانية فلا، وبالنسبة للقيمة الثالثة فهي تمكن المستخدم من تحرير النص ولكنها تظهر عناصر الأداة بشكل مبدئي.

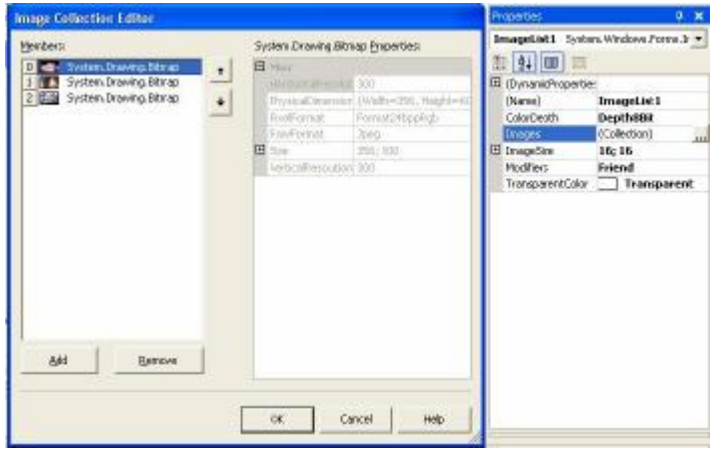
ان اخترت القيمة الاولى أو الثانية، يمكنك عرض قائمة عناصر الأداة في اي وقت برمجيا باسناد القيمة True إلى الخاصية DroppedDown.

الأداة ImageList

تستخدم هذه الأداة كمحفظة أو حاوية للصور التي تود عرضها على الأدوات الاخرى، صحيح ان معظم الأدوات يمكنك وضع قيم صور لها مباشرة عن طريق خاصيتها Image، الا ان استخدام الأداة ImageList سيوفر عليك مساحة عند حفظ الصور المتكررة، ليس هذا فقط بل ان بعض الأدوات (كـ TreeView و ListView) لن تتمكن من عرض رموز على عناصرها الا ان وجدت أداة ImageList على جبهة نافذة النموذج.

يمكنك إضافة وحذف الصور في الأداة ImageList وقت التصميم، وذلك عن طريق الخاصية Images، انتقل إلى نافذة الخصائص واضغط على الزر ... المقابل للخاصية ليظهر لك صندوق حوار بعنوان Image Collection Editor (**شكل 14-7**) يمكنك من إضافة الصور.

PDF



شكل 14-7: إضافة صور للأداة ImageList

ملاحظة

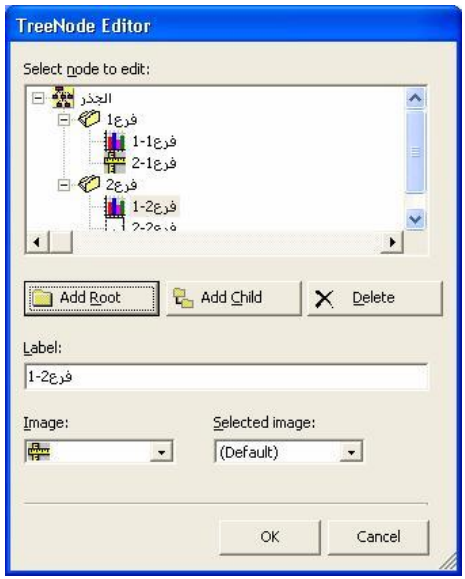
الخاصية Item عبارة عن مجموعة Collection، فهي تحتوي على الواجهة ICollection لذلك، يمكنك استخدام الطرق التقليدية لإضافة، حذف، والاستعلام عن الصور (كـ Add()، Insert()، Clear()، Count() ...الخ) وقت التنفيذ.

الأداة TreeView

تمكنك الأداة TreeView من عرض العناصر على شكل شجري كما تعرض مجلدات مستكشف النظام Windows Explorer، تتطلب هذه الأداة أداة ImageList إن أردت عرض صور ورموز على عناصر الأداة، يمكنك تحديد ارفاق أداة ImageList إلى الأداة TreeView عن طريق الخاصية ImageList والتابعة للأداة TreeView.

العناصر التي تضيفها إلى الأداة تسمى Nodes، يمكنك تحريرها وقت التصميم عن طريق الخاصية Nodes والتي تمثل مجموعة Collection للعناصر، اضغط على الزر المرافق للخاصية في نافذة الخصائص ليظهر لك صندوق الحوار Tree Node Editor **(شكل 14-8)**.

PDF Factory Pro trial version www.fastio.com



شكل 14-8: تحرير عناصر Node الأداة TreeView.

ستستخدم الخاصية Nodes دائما في شيفراتك البرمجية إن أردت التعامل مع العناصر (كما تفعل مع الخاصية Items التابعة للأداة ListBox) وقت التنفيذ، الا انك ستنشئ كائن من النـــوع TreeNode عند إضافة العنصر بالطريقة Add() أو Insert():

```
TreeView1.Nodes.Insert(0, New TreeNode("عنصر جذري"))
TreeView1.Nodes(0).Nodes.Add(New TreeNode("عنصر فرعي"))
```

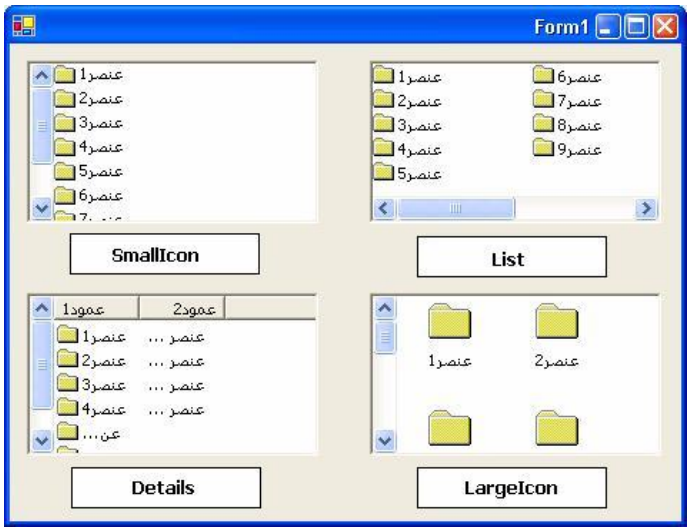
من خصائص الأداة TreeView الخاصية ShowLines التي تظهر خطوط تربط العناصر الفرعية بالعناصر الجذرية، الخاصية ShowPlusMinus التي تظهـــر علامـــــات الزائـــد (+) والناقص (-) للعناصر الجذرية، والخاصية Indent التي تحدد فيها المسافة بين العنصر الجذري وحد الأداة الأيسر.

ملاحظة

الخاصية RightToLeft تحاذي أشرطة تمرير الأداة بالاتجاه الصحيح ولكنها لا تشمل عناصر الأداة Nodes، حيث ستضل من اليسار إلى اليمين. إن أردت تحويلها إلى الاتجاه العربي عليك استخدام تقنية المرآة Mirroring كما سترى لاحقا.

الأداة ListView

تمكنك الأداة ListView من عرض عناصر على شكل ايقونات كما يفعل سطح المكتب Desktop ومستكشف النظام Windows Explorer لعرض الملفات، حدد في الخاصية View اسلوب من اربعة اساليب لعرض عناصر الأداة هي: SmallIcon، List، Details، و LargeIcon (**شكل 14-9**).



شكل 14-9: الاساليب المختلفة لعرض عناصر الأداة في الخاصية View.

تتطلب الأداة ListView اداتين من النوع ImageList لعرض صور في عناصر الأداة (يمكنك الاعتماد على أداة ImageList إن أردت)، حددها في الخاصية SmallImageList الأداة ImageList التي تود عرض رموزها في الحالات الثلاث للخاصية View، وحدد في الخاصية

LargeImageList الأداة ImageList التي تستخدم لعرض الرموز الكبيرة (الحالة LargeIcon للخاصية View).

التعامل مع الأداة ListView هو مثل التعامل مع الأداة TreeView السابقة، ولا يوجد داعي للتكرار، فان كانت الأداة TreeView تعتمد على المجموعة Nodes لعرض عناصرها، فان الأداة ListView تعتمد على المجموعة Items لعرض عناصرها.

ان كان اسلوب العرض للأداة ListView في الخاصية View هو Details، فتستطيع الاعتماد على المجموعة Columns لتحديد الأعمدة، وفي هذه الحالة ستستخدم المجموعة SubItems والتابعة لكل عنصر من عناصر المجموعة Items لتوزيع بيانات العنصر في الحقول (الأعمدة) المختلفة للأداة.

الأداتان ToolBar و StatusBar

في اغلب الاحوال، تستخدم كلا الاداتين ToolBar و StatusBar في نافذة النموذج، الأداة الاولى تعرض لك شريط أدوات تستطيع تحرير ازراره وقت التصميم عن طريق الخاصية Buttons، والأداة الثانية تعرض لك سطر حالة يمكنك تحرير المربعات التي فيه عن طريق خاصيته Panels.

كلا الاداتين تتطلبان كائن من النوع ImageList لعرض الرموز على عناصرها، وكلا الخاصيتين Buttons و Panels تحتويان على الواجهة ICollection، لذلك تستطيع التعامل مع عناصرهما برمجيا كما تفعل مع المجموعة Nodes (للأداة TreeView) والمجموعة Items (للاداتين ListBox و ListView)، والاختلاف لا يتعدى نوعية واسماء الخصائص والطرق التي تناسب كل أداة.

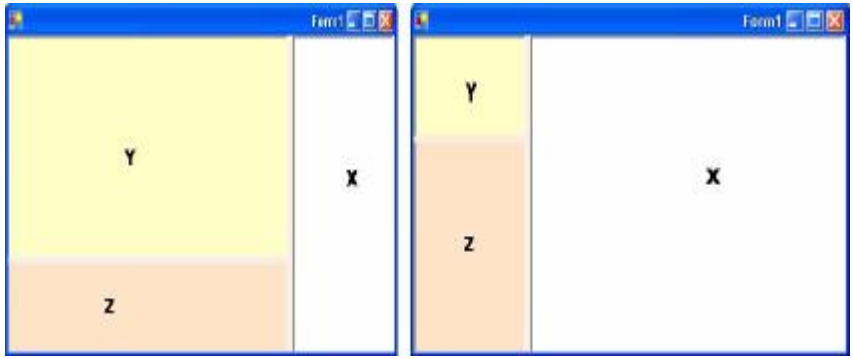
ملاحظة

حتى تتمكن من رؤية مربعات Panels والخاصة بسطر الحالة في الأداة StatusBar، عليك إسناد القيمة True للخاصية ShowPanels.

PDF

الأداة Splitter

توفر الأداة Splitter على نفسك كتابة عشرات الأسطر من الشيفرات المصدرية والتي تتعلق باعطاء قابلية للمستخدم بتحجيم الأدوات (كما تفعل نوافذ مستكشف النظام Windows Explorer)، فكل ما تطلبه منك الأداة خطوات بسيطة بالفارة وستجعل أدواتك قابلة للتحجيم من قبل المستخدم (**شكل 14-10**).



شكل 14-10: تحجيم الأدوات وقت التنفيذ باستخدام الأداة Splitter.

تتعامل الأداة Splitter مع الأدوات المُحاذة بالخاصية Dock ولن تستطيع تعلم استخدام الأداة Splitter الا بتطبيق عملي عليها، اضف أداة X في نافذة النموذج وقم باسناد القيم Right لخاصيتها Dock (ليتم محاذاة الأداة في الجزء الايمن من النافذة)، اضف الان أداة Splitter وغير خاصيتها Dock إلى Right، اضف أداة Y واسند القيمة Top لخاصيتها Dock، اضف أداة Splitter اخرى بنفس القيمة Top لخاصيتها Dock، واخيرا اضف أداة Z واسند القيم Fill لخاصيتها Dock.

نفذ البرنامج وحاول تغيير حجم الأدوات يمينا ويسارا، فوق وتحت (**شكل 14-10**).

أدوات صناديق الحوار الشائعة

توجد ست أدوات يمكنك استخدامها لعرض مجموعة من صناديق حوار نظام Windows الشائعة، كصندوق حوار فتح، حفظ، الطابعة ...الخ. التعامل مع هذه الأدوات يتشابه إلى حد كبير. يكفي إضافة نسخة واحد من كل أداة على نافذة النموذج حتى تتمكن من فتح صندوق الحوار اكثر من مرة مع تغيير خصائص العرض وبياناته.

الأداة OpenFileDialog:

تستخدم هذه الأداة لعرض صندوق حوار **فتح Open**، ابدأ بالخاصية Filter لتحديد نوع الملفات التي تود عرضها، كما يمكنك إسناد القيمة True إلى الخاصية MultiSelect لتمكن المستخدم من اختيار اكثر من ملف، الخاصية ShowReadOnly تظهر أداة من النوع CheckBox على صندوق الحوار بعنوان ReadOnly (يمكنك معرفة ما ان قام المستخدم بتحديد هذا الاختيار عن طريق الخاصية ReadOnlyChecked)، الخاصية CheckFileExists التي تجبر المستخدم على اختيار ملف موجود، والخاصية InitialDirectory التي تحدد المسار الابتدائي لصندوق الحوار.

بعد إسناد قيم للخصائص الآنفة، يمكنك استدعاء الطريقة ()ShowDialog لفتح صندوق الحوار، ستعود الطريقة بالقيمة DialogResult.OK ان تم الضغط على زر Open والقيمة DialogResult.Cancel ان كائن الضغط على الزر Cancel:

```
With OpenFileDialog1
     .CheckFileExists = True
     .Filter = "كل الملفات|*.*|" & _
         "برامج تطبيقية|*.EXE|ملفات صور|*.BMP;*.GIF;*.JPG"
     .FilterIndex = 2
     .InitialDirectory = "C:\"

     If .ShowDialog = DialogResult.OK Then
         …
         …
     Else
         …
         …
     End If
 End With
```

الأداة SaveFileDialog:

تعرض لك هذه الأداة صندوق حوار **حفظ** Save وهي تشابه إلى حد كبير الأداة OpenFileDialog السابقة، باستثناء بعض الخصائص التي لا تتناسب مع غرض ومهمة صندوق الحوار (كالخاصية ShowReadOnly). المزيد أيضا، أسند القيمة True إلى الخاصية OverwritePrompt إن أردت اظهار رسالة تنبيه في حال ما تم اختيار ملف موجود.

الأداة ColorDialog:

تعرض هذه الأداة صندوق حوار الألوان لتمكن المستخدم من اختيار اللون بطريقة افضل، ويعتبر استخدام هذه الأداة سهل جدا حيث سيكون جل تركيزك على الخاصية Color:

```
With ColorDialog1
    If .ShowDialog = DialogResult.OK Then
        Me.BackColor = .Color
    End If
End With
```

الأداة FontDialog:

اما الأداة FontDialog فتعرض لك صندوق حوار الخطوط Fonts، تحتوي على مجموعة من الخصائص الاضافية كالخاصية ShowColor لتظهر قائمة الالوان (يمكنك الاستعلام عن اللون المختار عن طريق الخاصية Color). يمكنك فتح صندوق الحوار هذا بنفس الطرق السابقة:

```
With FontDialog1
     .ShowColor = True
     If .ShowDialog = DialogResult.OK Then
         TextBox1.Font = .Font
         TextBox1.ForeColor = .Color
     End If
 End With
```

لإعطاء مرونة لمستخدمي برامجك، لما لا تمكنهم من تحديد الخطوط ورؤية النتائج دون اغلاق صندوق الحوار وذلك بالضغط على الزر Apply –عوضا عن OK، يتم ذلك باسناد القيمة True إلى الخاصية ShowApply، ويمكن كتابة ردة الفعل عند الضغط على هذا الرز في الحدث Apply:

```
Private Sub FontDialog1_Apply(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles FontDialog1.Apply

    TextBox1.Font = FontDialog1.Font
    TextBox1.ForeColor = FontDialog1.Color
End Sub
```

الأداة PrintDialog:

هذه الأداة تعرض لك صندوق حوار اختيار الطابعة، من أهم خصائصها الخاصية PrinterSettings والتي عبارة عن كائن من النوع PrinterSettings يحتوي على جميع الاعدادات الموجودة في صندوق حوار الطابعة، الشيفرة التالية توضح لك طريقة استخدامه (الفئة PrinterSettings مشمولة في مجال الاسماء System.Drawing.Printing):

```
With PrintDialog1
    .AllowSomePages = True

    .PrinterSettings = New System.Drawing.Printing.PrinterSettings()

    If .ShowDialog = DialogResult.OK Then
        With .PrinterSettings
            MsgBox(.PrinterName)
            MsgBox(.FromPage)
            MsgBox(.ToPage)
        End With
    End If
End With
```

الأداة PageSetupDialog:

ان كان الأداة PrintDialog تعرض لك صندوق حوار اختيار الطابعة، فان الأداة PageSetupDialog تعرض لك صندوق حوار اعدادات صفحة الطباعة Page Setup، والتي يحدد فيها المستخدم معلومات تفصيلية (كالهوامش، حجم الورق، الطباعة الأفقية أو العمودية ...الخ)، يمكنك الحصول على هذه المعلومات عن طريق الخاصية PageSettings والتي تعود بكائن من النوع System.Drawing.Printing.PageSettings:

```
With PageSetupDialog1
    .AllowPaper = True

    .PageSettings = New System.Drawing.Printing.PageSettings()

    If .ShowDialog = DialogResult.OK Then
        With .PageSettings
            MsgBox(.PaperSize.Height & "x" & .PaperSize.Width)
        End With
    End If
End With
```

أدوات المزودات

أدوات المزودات Provider Controls هي مجموعة من الأدوات التي تضيف خصائص جديدة على جميع الأدوات الموجودة في نافذة النموذج لتطويرها وإضافة امكانية جديدة عليها. في هذه الفقرة ساعرض لك اداتين من هذا النوع –مع العلم انه يمكنك تطوير أدوات مزودات خاصة بك.

الأداة ToolTip:

تمكنك الأداة ToolTip من عرض مستطيل التلميح على الأداة (**شكل 14-11**)، واستخدامها سهل جدا، فكل ما هو مطلوب منك إضافة نسخة من الأداة ToolTip على النموذج، وبذلك تضيف الخاصية ToolTip on ToolTip1 لكل أداة موجودة على نافذة النموذج، أسند قيمة حرفية لهذه الخاصية الجديدة في كل أداة ليتم عرضها كتلميح ان ظل مؤشر الفأرة فترة من الوقت دون تحريك على سطح الأداة.

شكل 14-11: مستطيل التلميح ToolTip.

ملاحظة

جميع أدوات المزودات Provider Controls تضيف خصائص جديدة على الأدوات بالصيغة "**اسم الأداة** on **اسم الخاصية**".

المزيد أيضا، تستطيع التحكم في الفترة من الوقت المطلوبة لاظهار التلميح عن طريق مجموعة من الخصائص AutoPopDelay، ReshowDelay، و AutomaticDelay –رغم اني لا احبذ لك تعديلها.

الأداة HelpProvider:

الأداة HelpProvider تعمل كحلقة وصل بين الفئة Help، بحيث تمكنك من الاتصال بملف التعليمات وعرض ومحتوياته عندما يقوم المستخدم بالضغط على المفتاح [F1] عندما يكون التركيز على أداة معينة.

انظر أيضا

الفئة Help فئة خاصة بالتعامل مع ملفات التعليمات Help Files، الفصل القادم **مواضيع متقدمة** يعرض أمثلة لاستخدام هذه الفئة.

تحتوي الأداة HelpProvider على خاصية وحيدة هي HelpNamespace والتي تسند لها اسم ملف التعليمات MyHelpFile.CHM، وبعد وضعها على نافذة النموذج، ستضيف ثلاث خصائص على كل أداة من الأدوات هي: HelpNavigator، HelpKeyword، و HelpString.

الخاصية الاولى تحدد فيها الصفحة التي تود عرضها في ملف التعليمات، أو صفحة المحتويات Contents، أو الفهرس Index ...الخ، والخاصية الثانية تحدد فيها الكلمة التي تود البحث عنها في قسم البحث من ملف التعليمات، اما الخاصية الأخيرة HelpString فتظهر مستطيل تلميح (**شكل 14-11**) يتم عرضه بمجرد الضغط على المفتاح [F1] على الأداة (لابد مسح قيمة الخاصية HelpKeyword لتفعيل هذه الخاصية).

أدوات أخرى

في السطور السابقة عرضت ثلاث وعشرين أداة تستخدمها في معظم مشاريعك المبنية على Windows Forms بشكل مختصر، تبقى مجموعة اضافية من الأدوات التي يكفي ذكر اسمها والوظيفة التي تقوم بها.

الأداة PictureBox أداة بسيطة تمكنك من وضع صور عليها في خاصيتها Image، كما تستطيع تحجيم الصورة لتغطي كامل الأداة أو تحجيم الأداة لتغطي كامل الصورة عن طريق الخاصية SizeMode.

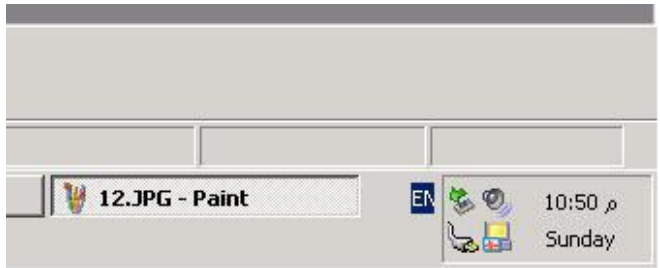
الاداتان Panel و GroupBox كلاهما من النوع الحاضن Container مثل نافذة النموذج، ابرز الفروق بينهما في دعم الخاصية AutoScroll حيث الثانية تفتقر هذه الخاصية كما انك لا

PDF

تستطيع اخفاء حدودها، مع ذلك توجد ميزة ليست موجودة في الأداة الاولى وهي النص الظاهر في احد زوايا الأداة تحدده عن طريق الخاصية Text.

بالنسبة للاداتين HScrollBar و VScrollBar فلا اعتقد انك ستكثر من استخدامها الا ان كنت تنوي تطوير أدوات Custom Controls خاصة بك، وذلك لان معظم الأدوات –التي ذكرناها في هذا الفصل- تحتوي على اشرطة تمرير. حدد اعلى واصغر قيمة لشريط التمرير في الخاصيتين Maximum و Minimum، وانتظر وقوع الحدث Scroll ان تم تحريك اشرطة التمرير لتختبر موقع مربع شريط التمرير في الخاصية Value.

لديك الأداة NotifyIcon والتي تمكنك من وضع رموز في **صينية النظام System Tray** كما تفعل اغلب برامج Windows الخدمية (**شكل 14-12**). حدد الرمز الذي تود عرضه في الخاصية Icon وقد تضيف مستطيل تلميح في الخاصية ToolTip، كما تستطيع عرض واخفاء الرمز من صينية النظام في اي وقت عن طريق الخاصية Visible. اما إن أردت ارفاق قائمة منبثقة مع ذلك الرمز، فحدد كائن القائمة المنبثقة في الخاصية ContextMenu.



شكل 14-12: الرموز في خانة صينية النظام System Tray.

الأداة TabControl تمكنك من وضع خانات التبويب Tab كما تفعل اغلب صناديق الحوار في تطبيقات Windows (**شكل 14-13**)، كل خانة تبويب من هذه الخانات تمثل كائن من النوع TabPage وهو كائن حاضن Container. يمكن تحرير خانا التبويب وقت التصميم عن طريق الخاصية TabPages التي تحتوي على الواجهة ICollection أيضا، كما تتطلب الأداة ImageList حتى تضع الرموز في اعلى خانات التبويب. اخيرا، عليك استخدام تقنية المرآة حتى توجه رؤوس خانات التبويب إلى الاتجاه العربي.

شكل 14-13: ثلاث خانات تبويب للأداة TabControl.

إن أردت أداة مثل الأداة TextBox ولكنها تدمج تنسيقات مختلفة من الخطوط، الألوان، الاحجام، وحتى الصور، فقد تحتاج إلى الأداة RichTextBox والتي تحتوي على خصائص وطرق كثيرة جدا جدا تجد تفاصيلها في مستندات NET. Documantation، مبدئيا لست بحاجة إلى معرفة كل هذه الخصائص، حيث ان اغلبها مشابه إلى حد كبير خصائص الأداة TextBox العادية.

يمكنك تسهيل اختيار القيم العددية بدلا من كتابة الارقام على المستخدم باستخدام الأداة NumericUpDown، كما تستطيع التسهيل عليه اكثر باختيار قيم التاريخ عن طريق الاداتين DateTimePicker و MonthCalendar. وان كانت المهام المنجزة طويلة، فيفضل عرض شريط نسبة مئوية للمستخدم باستخدام الأداة ProgressBar.

اخيرا، أداة المؤقت Timer تمكنك من تنفيذ شيفرات في حدثها الوحيد Tick والذي يتم تنفيذه كل فترة معينة تحددها في خاصية الأداة Interval (وحدتها 0.001 ثانية)، يمكنك بدء تنفيذ الحدث باسناد القيمة True إلى الخاصية Enabled أو القيمة False لايقافه (لا تسند القيمة 0 إلى الخاصية Interval حتى تتفادى وقوع استثناء Exception وقت التنفيذ).

تقنية المرآة

على مر العقود الأخيرة، واجه المبرمجون العرب معاناة كبيرة في تطوير واجهات استخدام تحمل معايير وسمات عربية خالصة، وان كنت من المبرمجين المخضرمين، فستذكر ان بدايات معاناتنا كانت تحت انظمة MS-DOS، حيث اضطررنا إلى استخدام أدوات تعريب كنافذة Nafitha، المساعد العربي Arabic Helper، وغيرها من انظمة التعريب التي لا اذكر اسمائها.

لست بصدد العودة إلى الوراء وذكر المصاعب المختلفة التي وقفت حاجزا للمطورين العرب، وكيف كانوا يتلهفون على العبارة "Right To Left" –فهو موضوع طويل، ولكن دعني أتسارع في الايام لنصل إلى نماذج Windows Forms (التابعة لتقنية NET. الحالية).

الخاصية RightToLeft

الفئة Control تحتوي على الخاصية RightToLeft، وبالتالي فان جميع الأدوات –بما فيها نافذة النموذج- تشمل هذه الخاصية والتي تسند لها قيمة من ثلاث قيم (تابعة لتركيب من النوع Enum باسم RightToLeft):

القيمة	الوظيفة
Yes	قلب اتجاه الأداة إلى الاتجاه العربي (من اليمين إلى اليسار).
No	قلب اتجاه الأداة إلى الاتجاه الغربي (من اليسار إلى اليمين).
Inherit	قلب اتجاه الأداة إلى نفس اتجاه الأداة الحاضنة.

عند تغيير اتجاه الأداة إلى الاتجاه العربي، فلا يوجد اي تغيير تطلبه شيفراتك المصدرية باستثناء الشيفرات التي تستخدم الخاصية TextAlign، حيث ان قيمة Right ستحاذي النص إلى جهة اليسار، والقيمة Left إلى جهة اليمين، لذلك يتحتم عليك وضع ذلك في عين الاعتبار عن تغيير اتجاه الأداة إلى الاتجاه العربي.

بالنسبة لأشرطة التمرير ScrollBars، فسيتم نقل شريط التمرير العمودي إلى الجهة المقابلة -اي يسار الأداة- ان كانت تحتوي على شريط تمرير عامودي، اما شريط التمرير الأفقي فسيضل مكانه ولكن عملية تحريك ستكون معاكسة.

المزيد أيضا، عند تغيير قيمة الخاصية RightToLeft سيتم تنفيذ الحدث RightToLeftChanged التابعة للفئة Control، وبالتي فهو مدعوم من قبل جميع الأدوات.

عند تصميم نافذة نموذج بالشكل الغربي، فان تنسيقات ومواقع الأدوات تكون مرتبة بأسلوب يتماشى مع طبيعة الاتجاه المستخدم. فمثلا، الازرار Buttons توضع في جهة اليمين، وأدوات Label توضع يسار الأدوات TextBox ...الخ، وان فكرت في جعل اتجاه هذه النافذة من اليمين إلى اليسار، عليك اعادة تحريك جميع الأدوات، فالأزرار ستضعها جهة اليسار، وأدوات Label ستكون يمين أدوات TextBox وغيرها، مع ذلك يمكنك جعل المسالة تتم تلقائيا وذلك بتعريف إجراء MirrorLocations:

```
Private Sub MirrorLocations(ByVal ctl As Control)
   Dim C As Control

   For Each C In ctl.Controls
      C.Location = New Point(C.Parent.ClientRectangle.Width _
         - C.Size.Width - C.Location.X, C.Location.Y)

      If C.Controls.Count > 0 Then
         MirrorLocations(C)
      End If
   Next
 End Sub
```

كما اعتقد انه سيكون افضل مكان مناسب نستدعي الإجراء MirrorLocations منه هو الحدث RightToLeftChanged:

```
Private Sub Form1_RightToLeftChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.RightToLeftChanged

    MirrorLocations(Me)
End Sub
```

ذكرت قبل بضعة سطور، ان قيم الخاصية TextAlign سيتم عكسها ان تم تغيير قيم الخاصية RightToLeft، مع ذلك لست بحاجة إلى تغييرها في الإجراء MirrorLocation السابق، حيث ان Windows Forms ذكية بما فيه الكفاية لتغيير قيمها.

إلا أن ذكاء Windows Forms لا يشمل الخاصيتين Anchor و Dock، فهما لا يتغيران من ناحية تنسيقية –كالخاصية TextAlign، ولكن المستخدم قام بوضع قيمهم على اساس اتجاه التصميم، لذلك يفضل بعكس قيم Right و الـ Left، ليكون الشكل النهائي للإجراء MirrorLocation:

```
Private Sub MirrorLocations(ByVal ctl As Control)
    Dim C As Control

    For Each C In ctl.Controls
        C.Location = New Point(C.Parent.ClientRectangle.Width _
            - C.Size.Width - C.Location.X, C.Location.Y)
        If CBool(C.Anchor And AnchorStyles.Left) Then
            C.Anchor = (C.Anchor Or AnchorStyles.Right) Xor
AnchorStyles.Left
        ElseIf CBool(C.Anchor And AnchorStyles.Right) Then
            C.Anchor = (C.Anchor Or AnchorStyles.Left) Xor
AnchorStyles.Right
        End If
        If C.Dock = DockStyle.Right Then

            C.Dock = DockStyle.Left
        ElseIf C.Dock = DockStyle.Left Then
            C.Dock = DockStyle.Right
        End If
        If C.Controls.Count > 0 Then
            MirrorLocations(C)
        End If
    Next
End Sub
```

قصور الخاصية RightToLeft

مع المزايا التي توفرها الخاصية RightToLeft لتطوير تطبيقات ذات واجهات عربية حقيقية، الا انها به بعض القصور الذي يظهر جليا على بعض الأدوات وأولها نافذة النموذج Form، حيث ان استخدام الخاصية RightToLeft لا يؤدي الا تغيير محاذاة عنوان النافذة (**شـــكل 14-14**)، ولا يعطي ان انطباع عربي حقيقة لاتجاه النافذة.

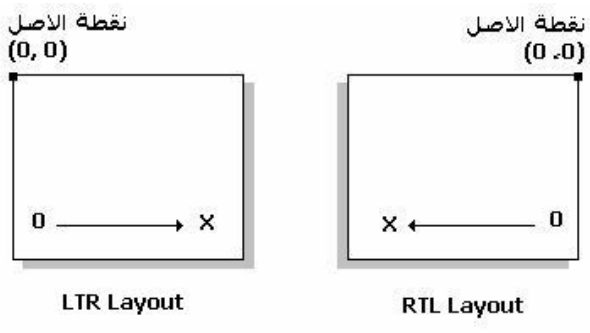
شكل 14-14: الخاصية RightToLeft لا تكفي لنافذة النموذج.

المزيـــــــد أيـــــــضا، الأدوات ListView، Panel، ProgressBar، StatusBar، TabControl، ToolBar، وTreeView لا تؤثر فيها قيمة الخاصية RightToLeft ولاتقـوم بقلب اتجاه عناصرها إلى الاتجاه العربي. اما باقي الأدوات فتعمل الخاصية RightToLeft بكفائة ولا توجد حاجة لاستخدام تقنية المرآة عليها.

مدخلك إلى تقنية المرآة

تقنية المرآة Mirroring مصطلح يستخدم لوصف اتجاه المخرجات إلى الاتجاه العربـي (مـن اليمين إلى اليسار Right to Left Layout)، كلمة المخرجات في هذا السياق كلمة عامة بحيـث تشمل اي شيء تراه عيناك على الشاشة (صور، أدوات، نصوص، ...الخ). ظهرت التقنيـة مـع بدايات نظم التشغيل Windows 98، ولكنها كانت مدعومة في النسخ العربية منها Enabled و Local فقط، اما مع الاصدارات Windows 2000 وما بعده، فتقنية المرآة أصبحت مدعومة في كافة النسخ والإصدارات.

إن أخذت مرآة ووضعتها بجانب يدك، ستلاحظ ان صورة يدك في المرآة قد انعكست، سبب الانعكاس هو ان الإحداثي السيني قد تم عكسه فقط، كذلك الحال مع تقنية المرآة، فكل ما تقوم بـه هذه التقنية هو عكس الإحداثي الأفقي **(شكل 14-15)**.



شكل 14-15: عكس الإحداثي السيني عن تطبيق تقنية المرآة.

لا تقتضي تقنية المرآة عكس الإحداثي السيني فقط، بل تتطلب عكس أفكارك ونظرتك إلـى الأمور في كل شيء، **فالإحداثي السيني سيزيد كلما اتجهنا يسارا ويقل كلما اتجهنا يمينا**. الجملة

السابقة تشمل كل شيء يتعلق بالإحداثيات تراه امام عينك، الأدوات وخصائص الموقع، إحـــداثيات النقاط التي ترسلها الفأرة، الصور والرسوم باستخدام GDI+، وكل شيء اخر.

اهم نقطة في هذا القسم من الفصل: **عملية العكس التي تنتجها تقنية المرآة تقع على الأداة والأدوات المحضونة بها فقط**. فمثلا، عند تطبيق تقنية المرآة على نافذة النموذج، وحاولت تحريك نافذة النموذج على سطح المكتب، استمر في التعامل معه كما تفعـــل ســـابقا، ولكـــن ان تعلقـــت الشيفرات المصدرية داخل حدود النموذج، فالوضع سيبدأ بالانعكاس.

ملاحظة

عند تطبيق تقنية المرآة على النماذج والأدوات، فانسى كل شيء يتعلق بالخاصية RightToLeft، حيث استخدامك لهذه الخاصية سيعمي الأمور بدلا من تكحيلها!

تطبيق تقنية المرآة بـ Visual Basic .NET

دعنا الان نأخذ مثالا يطبق تقنية المرآة، أنشئ نافذة نمـــوذج وضـــع بهـــا ثـــلاث أدوات هـــي: PictureBox، Button، و TabControl (**شكل 14-16**).

شكل 14-16: نافذة نموذج بها أداة PictureBox، Button، و TabControl.

لتطبيق تقنية المرآة على نافذة النموذج، عليك ارسال القيمة WS_EX_LAYOUTRTL إلى **النمط الموسع Extended Style** إلى النافذة، يمكنك تغيير قيمة النمط الموسع عن طريق الإجراء ()SetWindowLong وهو احد إجراءات API الشهيرة، وبما انني –للمرة الثانية- لست من المبرمجين الشجعان، فلن أتجرأ وانوي استخدام احد إجراءات API.

ملاحظة

النمط الموسع Extended Style تركيب خاص بنظام التشغيل Windows ونوافذه تحدد من خلاله بعض الصفات الظاهرية للنافذة كحدودها، الأزرار الظاهرة عليها، نمط النافذة ...الخ. (راجع مكتبة MSDN والخاصة بمراجع إجراءات API لمزيد من التفاصيل حول النمط الموسع والإجراء ()SetWindowLong).

سؤال اخر هام جدا، أين ومتى سنغير قيم النمط الموسع؟ والاجابة ستكون لحظة انشاء النافذة **من قبل نظام التشغيل وليس من قبل Windows Forms**، وفي الحقيقة Windows Forms لا تنشئ النوافذ من نفسها وانما تستخدم إجراء ()CreateWindow (من إجراءات API) في بنيتها التحتية لانشاء النافذة، وحتى تتمكن من الوصول إلى لحظة انشاء النافذة من قبل نظام التشغيل، ستقوم باعادة قيادة Overrides الخاصية CreateParams.

الخاصية CreateParams هي كائن من النوع CreateParams، يحتوي على الخاصية ExStyle التي تمكنك من تغيير النمط الموسع للنافذة، الشيفرة التالية ستتطبق تقنية المرآة على النافذة (**الشكل 14-17**):

```
Protected Overrides ReadOnly Property CreateParams() As _
    System.Windows.Forms.CreateParams

    Get
        Const WS_EX_LAYOUTRTL As Integer = &H400000

        Dim MirrorExStyle As System.Windows.Forms.CreateParams
        MirrorExStyle = MyBase.CreateParams

        ' تطبيق تقنية المرآة
        MirrorExStyle.ExStyle = MirrorExStyle.ExStyle Or
WS_EX_LAYOUTRTL

        Return MirrorExStyle
    End Get
End Property
```

شكل 14-17: تم عكس كل شيء حتى النص في الأداة Button والصورة في الأداة PictureBox.

تلاحظ في (**شكل 14-17**)، ان تغيير النمط الموسع على نافذة النموذج لم يكتفي فقط بتغيير وعكس مواقع الأدوات المحضونة بها، بل قام بتغيير النمط الموسع على الأدوات المحضونة نفسها، مما أدى إلى عكس النص الموجودة في الأداة Button وعكس الصورة الموجودة في الأداة PictureBox. لذلك، عليك ارسال القيمة WS_EX_NOINHERITLAYOUT أيضا إلى النمط الموسع حتى نطلب من نظام التشغيل عدم تغيير النمط الموسع للأدوات المحضونة. صحح النقص الموجود في الشيفرة السابقة واضف بيانات القيمة WS_EX_NOINHERITLAYOUT:

```
Const WS_EX_NOINHERITLAYOUT As Integer = &H100000

MirrorExStyle.ExStyle = MirrorExStyle.ExStyle Or _
    WS_EX_LAYOUTRTL Or WS_EX_NOINHERITLAYOUT
```

شكل 14-18: تم عكس الأدوات بالشكل الصحيح دون تغيير أنماطها الموسعة.

ملاحظة

لو عدنا إلى (**الشكل** 14-17) ستلاحظ ان الأداة TabControl قد عكست بالشكل الصحيح ولم تتأثر النصوص التي عليها، السبب في ذلك يتعلق باسلوب الرسم الذي يتبعه سياق الجهاز Context Device والخاص بالأداة TabControl، فمعظم الأدوات الاخرى (Labe، Button، CheckBox، RadioButton ...الخ) تتبع اسلوب رسم يعرف بـ GM_ADVANCED (راجع مكتبة MSDN لمزيد من التفاصيل)، اما الأدوات المعقدة الاخرى كـ ToolBar، TreeView، ListView لا تتبع هذا الأسلوب، لذلك لن تتأثر مخرجاتها النصية Textual.

متى ترسل القيمة WS_EX_NOINHERITLAYOUT:

لا يشترط دائما ارسال القيمة WS_EX_NOINHERITLAYOUT إلى النمط الموسع، حيث ان بعض الأدوات تحتوي على عناصر وأدوات اخرى بداخلها عليك تغيير نمطها الموسع حتى يتم تطبيق تقنية المرآة عليها بالشكل الصحيح، الجدول التالي يعرض لك الأدوات التي تتطلب استخدام القيمة WS_EX_NOINHERITLAYOUT (كما تقول مراجع MSDN):

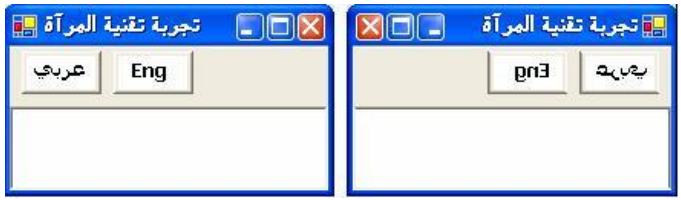
الأداة	ضرورة استخدام WS_EX_NOINHERITLAYOUT
Form	نعم.
ListView	لا.
Panel	نعم.
StatusBar	نعم.
TabControl	نعم.
TabPage	نعم.
ToolBar	لا.
TreeView	لا.

من الجدول السابق، يتضح لنا ان الأداة TreeView لا تتطلب منع العناصر المحضونة فيها من تغيير نمطها الموسع، لذلك لن نرسل القيمة WS_EX_NOINHERITLAYOUT لهـــا. لعمل ذلك، اعد قيادة الخاصية CreateParams التابعة للأداة TreeView، وحتى تتمكن من فعل ذلك، عليك تعريف فئة جديدة مشتقة من الأداة TreeView:

```
Public Class ArabicTreeView
    Inherits System.Windows.Forms.TreeView


    Protected Overrides ReadOnly Property CreateParams() As _
        System.Windows.Forms.CreateParams

        Get
            Const WS_EX_LAYOUTRTL As Integer = &H400000
            Dim MirrorExStyle As System.Windows.Forms.CreateParams

            MirrorExStyle = MyBase.CreateParams

            MirrorExStyle.ExStyle = MirrorExStyle.ExStyle Or
WS_EX_LAYOUTRTL

            Return MirrorExStyle
        End Get
    End Property
End Class
```

مشاكل إضافية

توجد مشكلة بسيطة عند تطبيق تقنية المرآة تتمحور حول الصور المعروضة على الأداة، فعند تطبيق تقنية المرآة على الأدوات التي تعرض صور في عناصرها (كالأداة TreeView، ListView، و ToolBar)، سيتم عكس اتجاه الصور أيضا (**شكل 14-19**).



شكل 14-19: صور موجودة في الأداة ToolBar تم عكسها

يمكنك حل المشكلة السابقة بأسلوبين، الاول في عكس الصور والرموز عند تصميمها ببرامج تحرير ومعالجة الصور، يعيب هذا الاسلوب ان الصور ستكون معكوسة ان طبقتها على أدوات لا تستخدم تقنية المرآة، اما الاسلوب الثاني هو عكس الصور برمجيا في شيفراتك المصدرية باستخدام طرق وخصائص الرسم ان كانت الصورة ستظهر على أداة تستخدم تقنية المرآة.

مشكلة اخرى تظهر مع الأدوات التي لايمكنك اشتقاقها وراثية NotInheritable (كالأداة Imagelist)، حيث لن تتمكن من إعادة قيادة خاصيتها CreateParams لتغيير نمطها الموسع، ويبقى الحل الفاشل هو بتغيير النمط الموسع لنافذة النموذج دون ارسال القيمة WS_EX_NOINHERITLAYOUT مما يؤدي إلى عكس كافة الأدوات المحضونة فيها (**شكل 14-17**).

أدوات صناديق الحوار الشائعة

بالنسبة لأدوات صناديق الحوار الشائعة (OpenFileDialog، SaveFileDialog، FontDialog، ColorDialog، PageSetupDialog، و PrintDialog فستظهر على نوع نسخة نظام التشغيل ولغة واجهة المحلية Local.

PDF

صناديق الرسائل

صناديق الرسائل التي تعرضها باستخدام الدالة MsgBox، فيمكن تغيير اتجاهها إلى الاتجاه العربي بارسال القيمتين MsgBoxRight و MsgBoxRtlReading مع الوسيطة الثانية للدالة:

```
MsgBox("هل تريد حذف الملف؟", MsgBoxStyle.MsgBoxRight _
    Or MsgBoxStyle.MsgBoxRtlReading _
    Or MsgBoxStyle.YesNo Or MsgBoxStyle.Question _
    Or MsgBoxStyle.DefaultButton2, "حذف ملف")
```

شكل 14-20: صندوق رسالة Message Box بالاتجاه العربي.

بالنسبة للغة الأزرار (كـ Yes و No) تعتمد على لغة نظام التشغيل المحلية Local، يمكنك تعديلها باستخدام مجموعة من إجراءات API ابحث عنها في مكتبة MSDN. اخيرا، ان استخدمت الإجراء MessageBox، فأرسل القيمتين RtlReading و RightAlign للوسيطة الأخيرة:

```
MessageBox.Show("هل تريد حذف الملف؟", "حذف ملف", _
    MessageBoxButtons.YesNo, _
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button2, _
    MessageBoxOptions.RightAlign Or MessageBoxOptions.RtlReading)
```

ملاحظة

الدالة MessageBox تعمل مثل عمل الدالة MsgBox ولكن بصيغة مختلفة.

PDF Factory Pro trial version www.fineprint.com

حاولت في هذا الفصل والفصل السابق تعريفك بنماذج وأدوات Windows Forms. وكما اتضح لك في السطور السابقة، تحتوي النماذج والأدوات على مئات الطرق، الخصائص، والأحداث التي تحتاج إلى وقت طويل للتعرف عليها واستخدامها بالشكل الامثل. ودعني أخبرك، بأنه ستأتي إصدارات أخرى من إطار عمل NET Framework. قبل الاستفادة من معظم هذه الأعضاء لأغلب المبرمجين. الفصل التالي يعرفك على تقنية **GDI+** والتي تمكنك من تصميم واجهات استخدام رسومية متقدمة.

PDF

مبادئ GDI+

إن الله جميل يحب الجمال، والجمال لا تكتشفه سوى لغة العيون، وان كان المبـــرمجين يحكمـــون على جودة البرامج من شيفراتها المصدرية، فان المستخدمين (وهم هدفنا الأول) نظرتهم محصورة على واجهاتها –اي من خارجها.

الفصل الخامس عشر **GDI+** يتعلق باستخدام تقنية GDI+ الموجه بشكل مباشر للتعامل مع الصور والرسوم والمخرجات النصية. قسمت هذا الفصل إلى ثلاث أقسام يمثل كل قـــسم زاويـــة ومنظور لاستيعاب واستخدام تقنية GDI+.

ملاحظة

لاختـصار كتابـة الاسـماء الطويلـة لفئـات GDI+، سـأفترض انـك قمـت باستيراد مجالات الاسماء التالية:

```
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging
Imports System.Drawing.Text
```

الرسم المتقدم

ان كان الرسامون يعتمدون على الريشة في التعبير ما بي داخلهم، فان مبرمجي .NET يستخدمون مجموعة من الفئات معظمها في مجال الاسماء System.Drawing.Drawing2D للتعبير عن خواطرهم الجياشة. هذا القسم مخصص بعمليات الرسم.

الكائن Graphics

قبل ان تبدأ باستخدام الريشة احم احم اقصد فئات الرسم، عليك تحضير ورقة الرسم اقصد **سياق الجهاز Device Context**. سياق الجهاز هو تركيب خاص بنظام التشغيل يحمل كل شيء يتعلق بمنطقة الرسم التي تنوي الرسم فيها، حتى تحصل على سياق رسم لابد ان تمتلك كائن من الفئة Graphics، وللاسف الشديد لن تستطيع إنشاء هذا الكائن مباشرة باستخدام New حتى تحصل عليه، بل عليك الحصول عليها بطريقة من طريقتين: الاولى عن طريق وسيطة بعض احداث الرسم -كالحدث Paint التابع للاداة أو نافذة النموذج:

```
Private Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As PaintEventArgs) Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics
    …
    …
End Sub
```

والثانية باستخدام الطريقة CreateGraphics() التابعة لنافذة النموذج والادوات:

```
Dim MainForm As New frmMain
…
…
Dim gr As Graphics = MainForm.CreateGraphics()
```

مع ذلك، يوجد اختلاف كبير بين الطريقتين من ناحية تفريغ المصادر، حيث ان استخدام سياق رسم جاهز لا يتطلب منك تحريره يدويا من الذاكرة عند عدم الحاجة له، وذلك ان الأداة أو نافذة النموذج (التي أخذت منها سياق الرسم) ستقتل هذا السياق لحظة موتها، اما إنشائك لسياق رسم خاص باستدعاء الطريقة CreateGraphics() يحصر المسؤولية عليك كمبرمج من تحرير هذا السياق عند عدم الحاجة له:

```
gr.Dispose()
gr = Nothing
```

بعد إنشائك لسياق الرسم، فانك جاهز الان لاستخدام هذا السياق وبدء الرسم الفعلي باستدعاء عدد كبير من الطرق التابعة للفئة Graphics نفسها أو بعض الفئات الاخرى –كما ستريك الفقرات التالية.

رسم الخطوط، المستطيلات، والدوائر

توفر الفئة Graphics مجموعة من الطرق التي تمكنك من رسم اشكال مبسطة (كالخطوط، المستطيلات، الدوائر، الاقواس وغيرها). معظم هذه الطرق تم اعادة تعريفها Overloads باشكال عديدة، الا ان الوسيطة الاولى تكون دائما نوع القلم وهو كائن من الفئة Pen (سأتحدث عنه لاحقا في الفقرة **الكائن Pen**)، كما يمكنك تحديد احداثيات النقاط على شكل وسيطات أو ارسالها دفعة واحدة على شكل كائن من النوع Rectangle. استدعي الطريقة DrawLine()، الطريقة DrawRectangle()، أو الطريقة DrawEllipse() لرسم –على التوالي- الخطوط، المستطيلات، الدوائر (يمكن ان تكون دوائر بيضاوية ايضا):

```
Private Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As PaintEventArgs) Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics
    gr.DrawLine(Pens.Black, 0, 0, 200, 200)
    gr.DrawRectangle(Pens.Red, New Rectangle(0, 0, 200, 200))
    gr.DrawEllipse(Pens.Blue, 0, 0, 200, 200)
End Sub
```

انظر ايضا

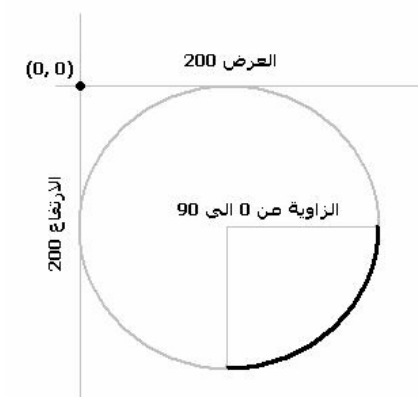
اذكر اني لمحت سابقا إلى الكائنات من النوع Rectangle في الفصل السابق **الأدوات** Controls عندما عرضت الخصائص المشتركة للموقع والحجم.

بالنسبة للاقواس فيمكنك رسمها باستدعاء الطريقة DrawArc() والتي تتطلب نفس وسيطات الطريقة DrawEllipse() السابقة بالاضافة إلى وسيطتين تحدد فيهما زاوية البداية والنهاية للقوس **بنفس اتجاه عقارب الساعة** (وحدة القياس الدرجة وليس الراديان):

```
gr.DrawArc(Pens.Black, 0, 0, 200, 200, 0, 90)
```

(**شكل 15-1 بأعلى الصفحة التالية**) يوضح احداثيات الشيفرة السابقة.

PDF



شكل 15-1: احداثيات الاستدعاء DrawArc (…, 0, 0, 200, 200, 0, 90).

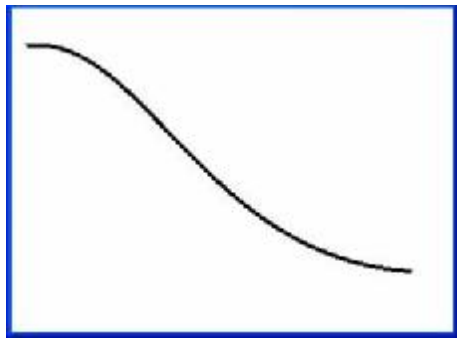
المزيد ايضا، لديك الطريقتين DrawLines() و DrawRectangles() والتي تستقبلان وسيطة مصفوفة من النوع Point و Rectangle على التوالي، هذا مثال لاستخدام الاولى:

```
Dim gr As Graphics = e.Graphics

Dim points() As Point = {New Point(10, 10), _
                         New Point(200, 200), New Point(10, 100),_
                         New Point(10, 10)}

gr.DrawLines(Pens.Green, points)
```

رسم المنحنيات المعقدة

يمكنك رسم منحيات معقدة باستخدام الطريقتين DrawCurve() و DrawBezier()، راجع مكتبة MSDN للحصول على كافة التفاصيل حول وسيطات هذه الطرق ومدى تأثيرها، حيث سأكتفي هنا بعرض مثال لاستخدام الطريقة الثانية DrawBezier()، يبينه لك (**الشكل 15-2**):

```
Dim gr As Graphics = e.Graphics

gr.DrawBezier(Pens.Black, 10, 30, 100, 20, 140, 190, 300, 200)
```



شكل 15-2: منحنى معقد باستخدام الطريقة ()DrawBezier.

كائن القلم Pen

في الفقرات السابقة كنا نستخدم مجموعة معرفة من كائنات الاقلام في اطار عمل NET. Framework باسماء مثل: Pens.Black، Pens.Red، Pens.Green ...الخ. يمكنك استخدام مجموعة اخرى من الاقلام تحمل الوان النظام System Colors في الفئة SystemPens (مثل: SystemPens.ControlDark).

اما في هذه الفقرة فسنقوم بإنشاء اقلام خاصة بنا تحمل الوان على أمزجتنا وسمكها اكثر من نقطة واحدة وأنماط خط مختلفة ...الخ. نستطيع عمل ذلك بفضل الفئة Pen والتي تمكنك من إنشاء كائنات تمثل اقلام جديدة.

معظم مواصفات القلم يمكنك تحديدها في مشيد الفئة Pen، الشيفرة التالية ستنشئ قلما جديدا لونه اسود وحجمه 4 بكسل:

```
Dim myPen As New Pen(Color.Black, 4)
```

يمكنك استخدام الكائن myPen مع اي طريقة من طرق الرسم التابعة للفئة Graphics السابقة، وذلك بارساله محل الوسيطة الاولى:

```
Dim gr As Graphics = e.Graphics
gr.DrawLine(myPen, 0, 0, 200, 200)
```

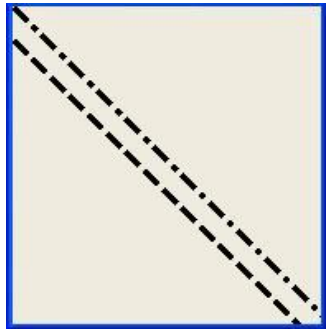
من الضروري التنبيه هنا بأنك انت المسئول الأول والأخير عن تحرير كائن القلم من الذاكرة ان لم تعد هناك حاجة لاستخدامه:

PDF

```
myPen.Dispose()
myPen = Nothing
```

المزيد ايضا، تستطيع تغير نمط الخط (**شكل 15-2**) وذلك في ان تجعله منقطة . . . . . أو مقطع - - - - - - أو مجموعة من الاشكال الاخرى تجدها في التركيب DashStyle والذي تسنده إلى الخاصية التي تحمل نفس الاسم DashStyle:

```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.Black, 4)

myPen.DashStyle = DashStyle.DashDot
gr.DrawLine(myPen, 0, 0, 200, 200)
myPen.DashStyle = DashStyle.Dash
gr.DrawLine(myPen, 0, 20, 200, 220)

myPen.Dispose()
```



شكل 15-2: تغيير نمط الخط DashStyle.

بل الأعظم من ذلك، يمكنك تخصيص شكل نمط الخط بنفسك عن طريق اسناد مصفوفة من النوع Single إلى الخاصية DashPattern، كل عدد من هذه المصفوفة يمثل عرض الشرطة – (ضع القيمة 1 ان اردت نقطة):

```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.Black, 7)
Dim customDash() As Single = {2, 1, 5, 1}
myPen.DashPattern = customDash
gr.DrawLine(myPen, 10, 10, 300, 10)
```

شيء ظريف جدا أعجبني في الكائن Pen –ومتأكد من انه سيعجبك انت ايضا- وهو قدرتك على تخصيص نقطة البداية والنهاية للخط عن طريق الخاصيتين StartCap و EndCap:

```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.Black, 7)
myPen.StartCap = LineCap.ArrowAnchor
myPen.EndCap = LineCap.ArrowAnchor
gr.DrawLine(myPen, 10, 40, 300, 40)
```

مخرجات الشيفرة السابقة وقبل السابقة يوضحها لك (**الشكل 15-3**):

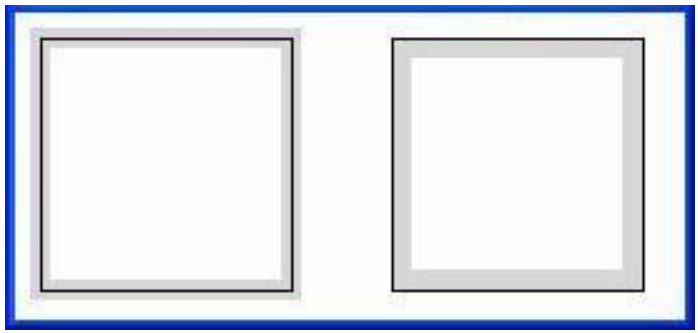
شكل 15-3: تأثير الخصائص DashPattern، StartCap، و EndCap.

من ناحية اخرى، عند تعريف اقلام سمكها يزيد عن 1 بكسل، فينصح بشدة تحديد محاذاتها للاحداثيات، فكما تعلم ان الاحداثي للموقع (0, 0) يكون في نفس النقطة (0, 0) ان كان سمك الخط 1 بكسل، اما ان زاد السمك عن واحد بكسل فقد تكون النقطة فوق أو اسفل الاحداثي المحدد. لعمل ذلك، استخدم الخاصية Alignment لتحديد نوع المحاذاة:

```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.LightGray, 8)

myPen.Alignment = PenAlignment.Center
gr.DrawRectangle(myPen, New Rectangle(10, 10, 100, 100))
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 100, 100))

myPen.Alignment = PenAlignment.Inset
gr.DrawRectangle(myPen, New Rectangle(150, 10, 100, 100))
gr.DrawRectangle(Pens.Black, New Rectangle(150, 10, 100, 100))

myPen.Dispose()
```

الشيفرة السابقة ترسم اربعة مستطيلات بنفس الحجم توضح لك مدى تأثير الخاصية Alignment على مواقع رسم اقلامك الخاصية (**شكل 15-4**).

PDF



شكل 15-4: تأثير الخاصية Alignment على مواقع رسم الاقلام التي تزيد سمكها عن 1 بكسل.

كائن مسار الرسم GraphicsPath

كائن مسار الرسم ما هو إلا كائن يحتوي على جميع عمليات الرسم التي نودها، يمكنك اعتبار كائن مسار الرسم على انه وعاء نضع فيه كافة طرق الرسم السابقة (بالصيغة AddLine()، AddRectangle()، AddCirlce() ...الخ) ومن ثم نرسمها دفعة واحدة.

الخطوة الاولى هي إنشاء كائن مسار رسم من الفئة GraphicsPath بالكلمة المحجوزة New:

```
Dim myPath As New GraphicsPath()
```

دعنا نضيف ثلاث خطوط تمثل شكل مثلث:

```
myPath.AddLine(10, 10, 30, 60)
myPath.AddLine(30, 60, 60, 10)
myPath.AddLine(60, 10, 10, 10)
```

ان اردت اضافة المزيد من الخطوط، فعليك الانتباه بانه سيتم ايصال النقطة الاخيرة من اخر خط رسمناها (10, 10) بالنقطة الاولى لاول خط سنرسمه، لذلك سنستدعي الطريقة StartFigure() لمنع حدوث ذلك:

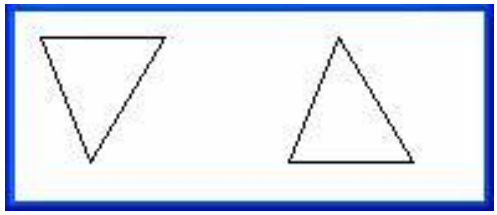
```
myPath.StartFigure()
myPath.AddLine(110, 60, 130, 10)
myPath.AddLine(130, 10, 160, 60)
```

لست بحاجة إلى اغلاق المثلث الاخير، حيث استدعاء الطريقة CloseFigure() يفي بالغرض:

```
myPath.CloseFigure()
```

والان كل ما تحتاجه كائن سياق رسم Graphics واستدعاء الطريقه DrawPath() (**شكل 16-6**):

```
Dim gr As Graphics = e.Graphics
…
…
…
gr.DrawPath(Pens.Black, myPath)
myPath.Dispose()
```



شكل 16-6: مخرجات مسار الرسم GraphicsPath في الشيفرات السابقة.

ملاحظة

من الضروري قتل كائن مسار الرسم باستدعاء الطريقة Dispose() عند عدم الحاجة إليه.

التعبئة

تحتوي الفئة Graphics على ثمانية طرق معاد تعريفها Overloads بصيغ متعددة تمكنك من رسم اشكال معبئة Filled (يعني ملونة) هي: FillRectangle()، FillRectangles()، FillEllipse()، FillPie()، FillClosedCurve()، FillPath()، و FillRegion(). راجع مستندات NET Documantation. لمزيد من التفاصيل حول هذه الطرق.

PDF

الوسيطة الاولى لجميع هذه الطرق عبارة عن كائن من النوع Brush يمثل الفرشاة، يمكنك استخدام مجموعة جاهزة ومعرفة من الفرش أو تطوير فرش خاصة بك –كما سنفعل في الفقرة التالية **كائن الفرشاة**.

الفرق بين كائن الفرشاة وكائن القلم، هو ان كائن الفرش يستخدم لتحديد نقش ونمط التعبئة، بينما القلم خاص بحدود الخطوط والاشكال الاخرى التي رسمناها، يمكنك استخدام الفئة Brushes (أو استخدام SystemBrushes لالوان النظام System Color) لاستخدام انواع معرفة وجاهزة من فرش التعبئة:

```
Dim gr As Graphics = e.Graphics

' رسم دائرة سوداء
gr.FillEllipse(Brushes.Black, 0, 0, 200, 200)
```

يمكنك استخدام الطريقة FillPath() لتلوين مسارات رسم (كائنات من النوع GraphicsPath) والتي تحدثنا عنها في الفقرة السابقة، اما الطريقة FillRegoin() فتمكنك من تعبئة مناطق لكائنات من النوع Region (لم اتحدث عنها في هذا الكتاب).

كائن الفرشاة Brush

في الفقرة السابقة استخدمنا فرش معرفة وجاهزة للون التعبئة تابعة للفئات Brushes و SystemBrushes. وفي هذه الفقرة سنقوم بتطوير فرش خاصة بنا، وقبل ان نبدأ عليك معرفة ان جميع الفرش –سواء الجاهزة أو الخاصة بنا- مشتقة وراثيا من الفئة Brush.

توجد مجموعة كبيرة من الفئات التي تمكنك من إنشاء فرش خاصة بك، وأسهلها الفئة SolidBrush التي تستقبل في مشيدها لون الفرشاة بحيث يكون هو نفسه لون التعبئة:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New SolidBrush(Color.Red)

gr.FillEllipse(myBrush, 0, 0, 200, 200)
myBrush.Dispose()
```

ملاحظة

لا تنسى قتل كائن الفرشاة باستدعاء الطريقة Dispose() عند عدم الحاجة إليه.

PDF

اما الفرش المنشئة من الفئة HatchBrush فانسب ما توصف به فئة النقش، حيث تمكنك من تحديد نقش من بين 56 نقش يدعمه التركيب HatchStyle (لا اعتقد انك ستطلب مني عرضها كلها، اليس كذلك؟!)، بالاضافة إلى تحديد لون الامامية والخلفية للنقش:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New HatchBrush(HatchStyle.BackwardDiagonal _
    , Color.Green, Color.White)

gr.FillRectangle(myBrush, 0, 0, 100, 100)

myBrush.Dispose()
```

ملاحظة

خاصية النقش HashStyle للقراءة فقط، ولا يمكن تعديلها على كائن الفرشاة الا بإنشاء نسخة كائن جديدة.

ولمن يبحث عن فنون التدرج اللوني، فيسرني تعريفه بالفئة LinearGradientBrush والتي تمكنه من تحقيق ما يصبو اليه، ارسل حجم المقطع ولونا الامامية والخلفية واتجاه التدرج اللوني مع مشيد الفئة:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New LinearGradientBrush(New Rectangle(0, _
    0, 200, 200), Color.Black, Color.Blue, _
    LinearGradientMode.BackwardDiagonal)


gr.FillRectangle(myBrush, 0, 0, 300, 300)
myBrush.Dispose()
```

يمكنك تحديد الاختيار Tile عندما تنوي تغيير خلفية سطح مكتبك خاصة ان كان حجم الصورة صغير، وذلك ليتم تكرار عرض الصورة حتى تغطي كافة سطح المكتب. هذا ما تفعله بالضبط الفرش من النوع TextureBrush، حيث تتطلب في مشيدها الصورة التي تود ان تمثل التعبئة في الاشكال المستخدمة لهذه الفرشاة:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New TextureBrush(PictureBox1.Image)

gr.FillRectangle(myBrush, 0, 0, 300, 300)
myBrush.Dispose()
```

PDF

توجد فئة اخرى متقدمة كثيرا لإنجاز فرش التعبئة، وموجهة للاستخدام بشكل خاص مع كائنات مسارات الرسم من النوع GraphicsPath، ابحث عنها في مكتبة MSDN تحت العنوان PathGradientBrush.

أنظمة القياس

في هذه الفقرة سأعرض عليك كيف يمكنك التحكم في انظمة القياس Coordinate system والخاصة بكائن سياق الرسم Graphics لتتمكن من تحريك الاشكال من مواقعها ، تدوير العناصر وقلبها، وتحجيمها (تكبير/تصغير).

التحريك:

في الحقيقة، ان الذي سنقوم به هو عملية تغيير نظام القياس Coordinate system لكائن سياق الرسم Graphics، مما يعني تغيير موقع نقطة الاصل (0, 0)، فكما تعلم ان نقطة الاصل تكون في الزاوية العليا اليسرى من الاداة، مع ذلك قد تحتاج إلى توسيطها في وسط الأداة لتسهل عليك عملية رسم الدوال الرياضية.

لتغيير نظام القياس، استخدم الطريقة TranslateTransform() وارسل معها موقع نقطة الاصل:

```
Dim gr As Graphics = e.Graphics

' جعل نقطة الاصل وسط الاداة
gr.TranslateTransform(Picture1.ClientRectangle.Width \ 2, _
    Picture1.ClientRectangle.Height \ 2)
```

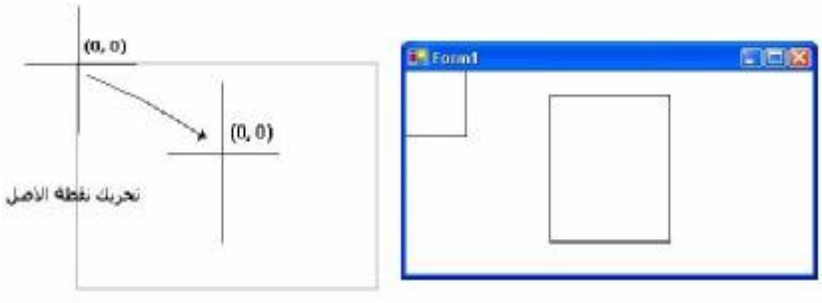
عند استدعاء الطريقة TranslateTransform() مرة اخرى، ستتأثر بآخر تعديل لنقطة الاصل وقع على سياق الرسم، لذلك يفضل دائما استدعاء الطريقة ResetTransform() قبلها:

```
Dim gr As Graphics = e.Graphics

Gr.ResetTransform()
' جعل نقطة الاصل وسط الاداة
gr.TranslateTransform(Picture1.ClientRectangle.Width \ 2, _
    Picture1.ClientRectangle.Height \ 2)
```

PDF Factory Pro trial version www.fastio.com

الشيفرة التالية تحاول رسم مربع بالطريقة ()DrawRectangle، وبعد رسمه تغير نظام القياس وتنقل موقع احداثيات نقطة الاصل إلى وسط النافذة بالطريقة ()TranslateTransform، ومن ثم تحاول رسم المربع بنفس الاحداثيات المرسلة قبل تعديل موقع نقطة الاصل (**شكل 15-7**):

```
Private Sub Form1_Paint(ByVal sender As Object, ByVal e _
    As Forms.PaintEventArgs) Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics

    gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
    gr.TranslateTransform(Me.ClientRectangle.Width \ 2, _
        Me.ClientRectangle.Height \ 2)

    gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
End Sub
```



شكل 15-7: تأثير الطريقة ()TranslateTransform على كائن سياق الرسم.

الدوران:

ان كانت الطريقة ()TranslateTransform تغير موقع نقطة الاصل، فان الطريقة ()RotateTransform تقلب المحور السيني والصادي x and y axis لنظام قياس الرسم والخاص بالكائن Graphics، ارسل مقدار زاوية القلب كوسيطة للطريقة لتيم تحريك المحاور باتجاه عقارب الساعة ان كانت القيمة موجبه، وعكس عقارب الساعة ان كانت القيمة سالبة. مخرجات الشيفرة التالية يظهرها لك (**الشكل 15-8**):

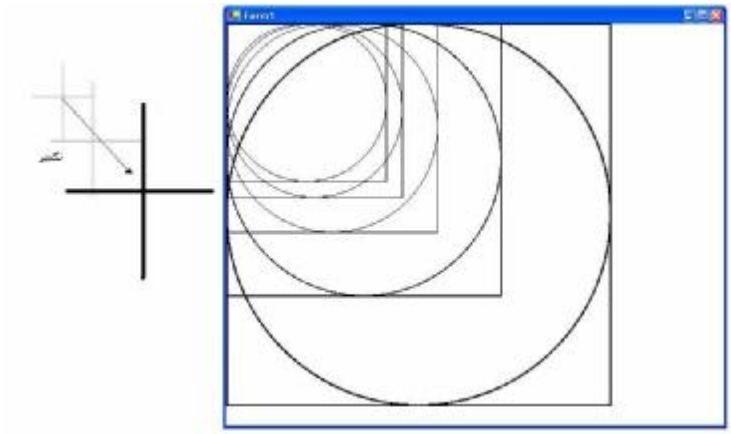
```
Dim gr As Graphics = e.Graphics

gr.TranslateTransform(Me.ClientRectangle.Width \ 2, _
    Me.ClientRectangle.Height \ 2)

gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
gr.RotateTransform(-45)
gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
```

شكل 15-8: تأثير الطريقة RotateTransform() على كائن سياق الرسم Graphics.

التحجيم:

يمكنك تحجيم نظام القياس بتكبيره أو تصغيره باستخدام الطريقة ScaleTransform() (**شكل 15-9**):

```
Dim gr As Graphics = e.Graphics
Dim myPath As New GraphicsPath()
Dim counter As Single

myPath.AddRectangle(New Rectangle(0, 0, 200, 200))
myPath.AddEllipse(New Rectangle(0, 0, 200, 200))

For counter = 1 To 1.6 Step 0.1
    gr.DrawPath(Pens.Black, myPath)
    gr.ScaleTransform(counter, counter)
Next
```



شكل 15-9: تاثير الطريقة ()ScaleTransform على كائن سياق الرسم Graphics.

وحدة القياس:

بشكل افتراضي، وحدة القياس لكائنات سياقات الرسم Graphics على الشاشة هي البسكل Pixel، يمكنك استخدام مجموعة من وحدات القياس الاخرى تسندها إلى الخاصية PageUnit هي: Inch انش، Millimeter ملم، Point نقطة (75/1 انش)، Display عرض (72/1 انش)، أو Document مستند (300/1 انش) –تفيدك الوحدة الاخيرة كثيرا مع طابعات الليزر:

```
Gr.PageUnit = GraphicsUnit.Inch
```

التعامل مع الصور

المنظور الثاني الذي يمكننا من استيعاب تقنية GDI+ يتعلق بالصور والتعامل معها. في هذا القسم من الفصل سنحاول ان نلقي الضوء على مجموعة من الفئات في مجال الاسماء Imports System.Drawing.Imaging والتي تتعلق بالصورة الجاهزة.

تحميل وحفظ الصور

عند التعامل مع الصور، استخدم الفئة Image أو الفئة Bitmap، الاولى تحتوي على طرق وخصائص لفتح وحفظ الصور، والثانية مشتقة وراثيا من الاولى وتحتوي على مجموعة اضافية من الطرق والخصائص التي تتعامل مع محتويات الصورة نفسها.

يمكنك تحميل صورة من ملف باستدعاء الطريقة LoadFromFile()، أو من وحدة تخزين Stream باستدعاء الطريقة LoadFromStream() –كلا الطريقتين مدعومتان لكلا الفئتين:

```
Dim JPG As Image = Image.FromFile("C:\Ibrahim.JPG")
```

انظر ايضا

لمزيد من التفاصيل حول وحدات التخزين Streams، راجع الفصل الثامن **الملفات والمجلدات**.

اسلوب اخر أسهل لفتح الملف وذلك بارسال مساره إلى مشيد الفئة:

```
Dim JPG As New Bitmap("C:\Ibrahim.JPG")
```

ملاحظة

تدعم GDI+ مجموعة كبيرة من هيئات الصور JPGE، GIF، BMP، PNG، TIFF ...الخ.

يمكنك في اي وقت من حفظ صورة في الكائنات النوع Bitmap أو Image باستدعاء طريقته Save() لحفظ ملف الصورة، تتطلب الطريقة اسم الملف والهيئة المراد حفظها:

```
Dim JPG As New Bitmap("C:\Ibrahim.JPG")
JPG.Save("C:\Ibrahim.GIF", ImageFormat.Gif)
```

عند التعامل مع هيئات صور معقدة، ستتطلب الطريقة Save() معلومات اضافية كالعمق اللوني، نسبة الضغط ...الخ.

عرض الصور

بمجرد حصولك على كائن من النوع Bitmap أو Image، يمكنك عرض صورته على اي سياق رسم باستدعاء الطريقة DrawImage() التابعة لسياق الرسم:

```
Private Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics
    Dim JPG As New Bitmap("C:\Ibrahim.JPG")


    gr.DrawImage(JPG, 0, 0)
End Sub
```

ملاحظة

لا تحاول ابدا تحميل الملف من خلال الحدث Paint، وذلك بسبب كثر عدد مرات تنفيذ الحدث، مما يضعف كفاءة التنفيذ. ولكن كان غرضي في الأمثلة السابقة التوضيح فقط.

المزيد ايضا، الطريقة ()DrawImage السابقة تم اعادة تعريفها Overloads باكثر من 30 صيغة، الصيغة السابقة تتطلب احداثي يمثل موقع رسم الصورة. صيغة اخرى مثل الصيغة السابقة، ولكن تضيف اليها المنطقة التي تود قطعها من الصورة الاصلية ورسمها (بارسال كائن من النوع Rectangle):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Ibrahim.JPG")

gr.DrawImage(JPG, 0, 0, New Rectangle(10, 10, 50, 50),
GraphicsUnit.Pixel)
```

صيغة ثالثة تمكنك من تحديد موقع وحجم المنطقة على سياق الرسم:

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Ibrahim.JPG")

gr.DrawImage(JPG, 0, 0, 200, 300)
```

كما يمكنك دمج الصيغتين الأخيرتين في خطوة واحدة، لتتمكن من قطع جزء من الرسمة و تحجيمها، الوسيطة الثانية تمثل المنطقة الهدف، والوسيطة الثالثة المنطقة المصدر (كلا الوسيطتين من النوع Rectangle):

```
Dim gr As Graphics = e.Graphics
Dim destRect As New Rectangle(0, 0, 200, 200)
Dim sourceRect As New Rectangle(30, 30, 50, 50)
Dim JPG As New Bitmap("C:\Ibrahim.JPG")

gr.DrawImage(JPG, destRect, sourceRect, GraphicsUnit.Pixel)
```

ملاحظة

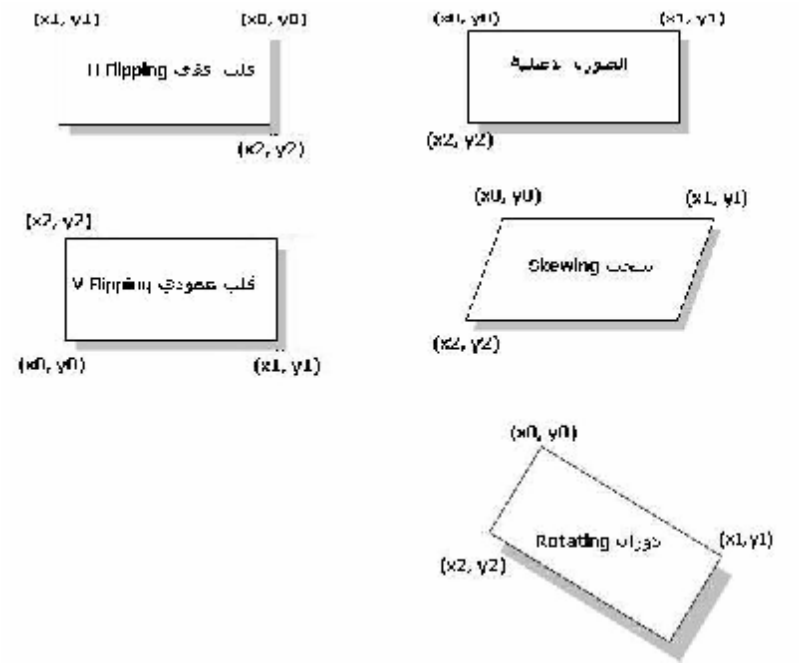
حتى تفرق بين الصيغ المختلفة، استعن بمحرر الشيفرة أو قراءة مستندات NET Documentation..

(**شكل 15-10**) يظهر لك مخرجات متعددة لصيغ الطريقة ()DrawImage.

شكل 15-10: مقاطع وأحجام مختلفة من الصورة.

عكس، قلب، وسحب الصور

من احد صيغ الطريقة ()DrawImage المعاد تعريفها صيغة تقبل مصفوفة من النـــوع Point، حجم هذه المصفوف ثلاث عناصر، العنصر الاول (x0, y0) يمثل النقطـــة العلويـــة اليـــسرى، العنصر الثاني (x1, y1) يمثل النقطة العلوية اليمنى، اما العنصر الثالث (x2, y2) فيمثل السفلية اليسرى راقب المربع "الصورة الاصلية" في (**الشكل 15-11**).



شكل 15-11: تحديد الاحداثيات لقلب، سحب، وتدوير الصورة.

كل ماهو مطلوب منك الان اسناد القيم المناسب في المصفوفة Point وارسالها إلى الطريقة ()DrawImage، لنبدأ مثلا بالقلب الأفقي H Flipping والذي يتطلب عكـــس النقـــاط اليمنـــى باليسرى فقط (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0)):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffel.JPG")
Dim points() As Point = {New Point(JPG.Width, 0), _
    New Point(0, 0), New Point(JPG.Width, JPG.Height)}

gr.DrawImage(JPG, points)
```

اما القلب العمودي V Flipping، فيتطلب عكس النقاط العلوية بالسفلية (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0) ايضا):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffl.JPG")
Dim points() As Point = {New Point(0, JPG.Height), _
    New Point(JPG.Width, JPG.Height), New Point(0, 0)}

gr.DrawImage(JPG, points)
```

بالنسبة للسحب Skewing، فيعتمد اعتماد كلي على مقدار السحب، وذلك وضعت المتغير L في الشيفرة التالية ليمكنك من زيادة/انقاص مسافة السحب (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0) ايضا):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffl.JPG")
Dim L As Integer = 50
Dim points() As Point = {New Point(L, 0), New Point(JPG.Width + L,
0),_
    New Point(0, JPG.Height)}

gr.DrawImage(JPG, points)
```

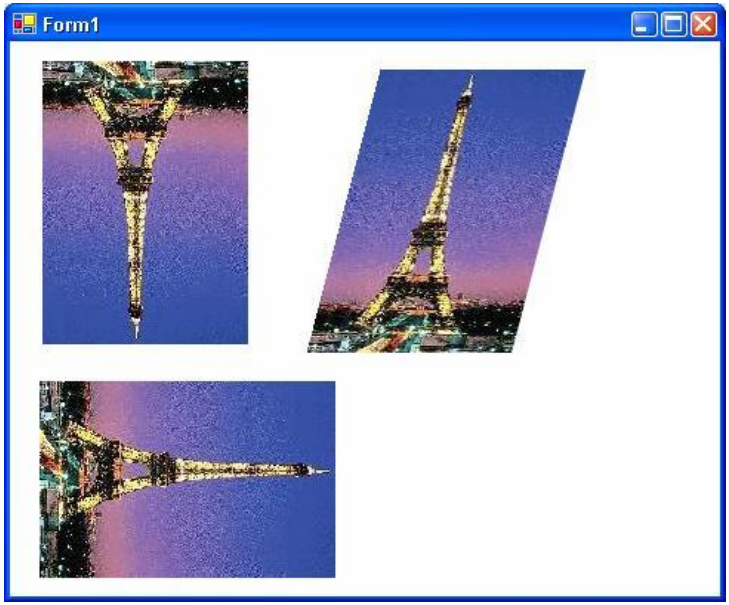
اخيرا، بالنسبة للقلب Rotating، فكان يمكنني وضع قيم ابتدائية للاحداثيات مباشرة، ولكن تعمدت إلى تعقيد الشيفرة اكثر وذلك للتسهيل عليك ووضع قيمة للزاوية بالدرجة في المتغير Angle بالشيفرة التالية (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0) ايضا، لذلك اضطررت إلى استخدام قيمة سالبة في العنصر الثالث للمصفوفة points):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffl.JPG")
Dim Angle As Integer = 90
Dim AngleInRad As Single = CSng(Angle / (180 / Math.PI))
```

```
Dim a As Single = CSng(Math.Cos(AngleInRad))
Dim b As Single = CSng(Math.Sin(AngleInRad))
Dim points() As PointF = {New PointF(0, 0), _
    New PointF(JPG.Width * a, JPG.Width * b), _
    New PointF(-JPG.Height * b, JPG.Height * a)}

gr.DrawImage(JPG, points)
```

ملاحظة

الفئة PointF هي تماما مثل الفئة Point، ويكمن الفرق ان الاولى تحمل خاصيتين X و Y من النوع Single، بينما الثانية من النوع Integer.

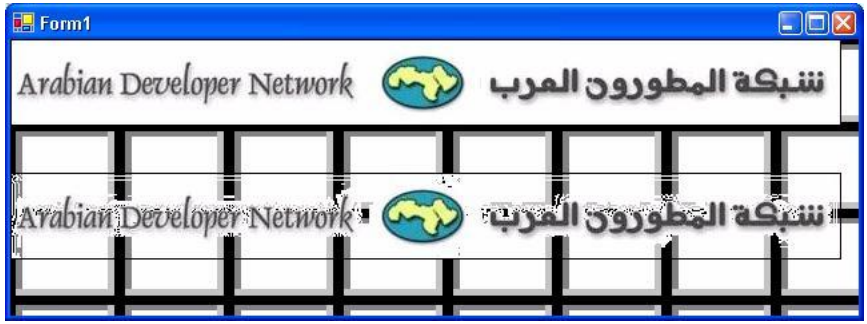


شكل 15-12: قلب عمودي للصورة، سحبها بمقدار 50 بكسل، وتدويرها بمقدار 90 درجة.

تحديد الألوان

يمكنك تحديد اللون الخفي Color Transparent في الصورة بإرسال قيمة اللون كوسيطة مع الطريقة MakeTransparent() (**شكل 15-13**):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\dev4arabs.JPG")

gr.DrawImage(JPG, 0, 0)
JPG.MakeTransparent(Color.White)
gr.DrawImage(JPG, 0, 100)
```



شكل 15-13: اختيار اللون الأبيض كلون مخفي.

المزيد ايضا، تستطيع تحديد درجة إخفاء اللون المخفي من الصورة (كما تفعل مع الخاصية Opacity لنوافذ النماذج) وذلك بارسال كائن من النوع ImageAttribute إلى الطريقة DrawImage() (التابعة كائن سياق الرسم Graphics)، ولو كان الامر بهذه السهولة لعرضت عليك مثال، اذ يتطلب الكائن ImageAttribute مصفوفة ثنائية البعد Two Dimensional Array ترسلها كوسيطة إلى طريقته SetColorMatrix، لذلك انصحك بالرجوع إلى مستندات NET Documentation. للاستخدام الامثل لها.

الرموز Icons

الفئة Icon تمثل رمز ايقونة من رموز نظام التشغيل Windows، يمكنك استعمال هذه الفئة لتحميل ورسم الرموز Icons، مع ذلك هذه الفئة ليست مشتقة من الفئة Image، فمعظم الطرق والخصائص التي تطرقت لها سابقا ليست مدعومة فيها.

يمكنك تحميل ملف الرمز بارساله كمشيد للفئة Icon (تستطيع استخدام وحدة تخزين Stream ايضا):

```
Dim icon As New Icon ("C:\test.ico")
```

طريقة اخرى تمكنك من الحصول على رمز هي باستخدام كائن رموز تابعة للفئة SystemIcons، تعود بمجموعة من الرموز الجاهزة تعتبر شائعة بين تطبيقات Windows:

```
Dim icon As Icon = SystemIcons.WinLogo()
```

بعد حصولك على مرجع للرمز في الكائن Icon، تستطيع ارساله كوسيطة لكائن سياق الرسم Graphics وعرضه فورا:

```
Dim gr As Graphics = e.Graphics
Dim icon As New Icon("C:\dev4arabs.ico")

gr.DrawImage(icon, 0, 0)
```

لا تنسى قتل كائن الرمز عند عدم الحاجة اليه باستدعاء الطريقة Dispose():

```
Icon.Dispose()
```

كما أخبرتك سابقا، الفئات من النوع Icon محدودة الإمكانيات، فهي ليست مشتقة من الفئة Image، وان اردت تطبيق الطرق والخصائص التي ذكرتها في الفقرات السابقة، يمكنك تحويل الرمز إلى صورة من النوع Bitmap باستخدام الطريقة ToBitmap():

```
Dim gr As Graphics = e.Graphics
Dim icon As New Icon("C:\dev4arabs.ico")
Dim BMP As Bitmap = icon.ToBitmap()

JPG.MakeTransparent(Color.White)
gr.DrawImage(BMP, 0, 100)

icon.Dispose()
BMP.Dispose()
```

PDF

اخيرا، كائن سياق الرسم Graphics يحتوي على الطريقة ()DrawIconUnstretched، والتي تستقبل كائن رمز Icon، بحيث ترسل معها كائن منطقة من النوع Rectangle يتم تكرار عرض الرمز في هذه المنطقة.

المخرجات النصية

الجزء الثالث والاخير من GDI+ يتعلق بالمخرجات الحرفية النصية والتي تستخدم الخطوط Fonts بكافة اشكاله، الوانه، احجامه، وانماطه لعرض الحروف. ومعظم الفئات التي سنتعامل معها في هذا القسم مشمولة في مجال الاسماء الخاص بالمخرجات النصية System.Drawing.Text.

قبل البدء في عرض المخرجات النصية، وتنسيق خطوطها ومحاذاة حروفها، من الجيد التعرف على عوائل الخطوط واخذ فكرة عامة حولها، حيث ستحتاج فئاتها كثيرا ان رغبت في معرفة الخطوط وأنماطها المدعومة في الجهاز.

عوائل الخطوط

عائلة الخط Font Family هي مجموعة مترابطة من انماط مختلفة للخط ولكن بشكل واحد، فمثلا افراد عائلة الخط Tahoma الموقرة تتكون من: Tahoma Regular (العادي)، Tahoma Bold (سميك)، Tahoma Italic (مائل)، و Tahoma Bold Italic (سميك ومائل).

يمكنك معرفة جميع عوائل الخطوط المثبتة في الجهاز عن طريق الخاصية Families والتابعة للكائن InstalledFontCollection والتي تعود بمصفوفة كائنات من النوع FontFamily:

```
Dim fonts As New InstalledFontCollection()
Dim fontFamilies() As FontFamily = fonts.Families

Dim fontFamily As FontFamily

For Each fontFamily In fontFamilies
    …
    …
    …
Next
```

ملاحظة

الخاصية Families لا تعود إلا بعوائل الخطوط من النوع TrueType و OpenType فقط.

طريقة اخرى تمكنك من معرفة عوائل الخطوط هي الطريقة المشتركة FontFamily.GetFamilies() والتي تتطلب كائت سياق جهاز من النوع Graphics:

```
Dim gr As Graphics = Me.CreateGraphics
Dim fontFamilies() As FontFamily = FontFamily.GetFamilies(gr)

gr.Dispose()
```

الفرق بين هذه الطريقة والخاصية Families السابقة، ان هذه الطريقة تعود بعوائل الخطوط التي يمكن عرضها واستخدامها على سياق الجهاز Graphics المرسل (فلا تنسى انه توجد عوائل خطوط موجه للاستخدام على الشاشة أو على الطابعة بشكل حصري).

كما يمكنك إنشاء كائن من النوع FontFamily مباشرة بارسال قيمة حرفية تمثل اسم عائلة الخط مع مشيده:

```
Dim fontFamily As New FontFamily("Tahoma")
```

بمجرد حصولك على مرجع لعائلة الخط في الكائن FontFamily، يمكنك الاستعلام عن الانماط المختلفة للخط باستخدام الطريقة IsAvailable()، والتي ترسل معها النمط المطلوب لتعود بالقيمة True ان كان مدعوما في الجهاز الحالي:

```
Dim fontFamily As New FontFamily("Tahoma")

If fontFamily.IsStyleAvailable(FontStyle.Bold) Then
    …
    …
End If
```

PDF Factory Pro trial version www.fastio.com

رسم النصوص

يقصد بعبارة رسم النصوص اي كتابة المخرجات الحرفية على سياقات الرسم Graphics، وان نظرتها من جانب تقني، عملية كتابة النصوص ما هي الا رسم تلك لمجموعة من الاشكال تمثل الحروف. على العموم، تحتاج إلى إنشاء كائن من النوع Font حتى تتمكن من الكتابة، يحتوي مشيد الكائن على 13 صيغة معاد تعريفها Overloads، الاولى تتطلب اسم وحجم الخط:

```
Dim font1 As New Font("Tahoma", 12)
```

صيغة اخرى تضيف لها نمط Style الخط:

```
Dim font2 As New Font("Tahoma", 20, FontStyle.Bold)
Dim font3 As New Font("Tahoma", 20, FontStyle.Bold Or
FontStyle.Italic)
```

وصيغة ثالثة تمكنك من استخدام كائن عائلة خط FontFamily:

```
Dim fontFamily As New FontFamily("Arial")
Dim font4 As New Font(fontFamily, 20, FontStyle.Bold)
```

ملاحظة

خصائص الكائنات من النوع Font للقراءة فقط ReadOnly ولن تتمكن من تعديلها بعد إرسالها إلى المشيد. إن أردت تغيير قيمة احد الخصائص، أنشئ الكائن من جديد.

حجم الخط الذي تختاره تكون وحدته خاصة بنظام التشغيل (وحدة مترية وليس بكسلية)، مع ذلك تستطيع تغيير الوحدة بارسال تركيب Enum بالاسم GraphicsUnit:

```
Dim myFont As New Font("Tahoma", 20, FontStyle.Bold,
GraphicsUnit.Pixel)
```

بعد إنشائك لكائن الخط وتنسيق كافه خصائصه، تستطيع البدء باستخدام فورا على اي كائن سياق جهاز يدعم ذلك الخط، استدعي طريقة سياق الجهاز ()DrawString لرسم النصوص، تتطلب وسيطتها الاولى النص String المراد رسمه، الثانية كائن الخط Font، الثالثة كائن الفرشاة Brush، والبقية احداثيات موقع الرسم:

```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 20, FontStyle.Bold)

gr.DrawString("شبكة المطورون العرب", myFont, Brushes.Black, 0, 0)
```

اخيرا، يمكنك الاستعلام عن المساحة المطلوبة لعرض النص بخط معين باستخدام الطريقة MeasureString() والتي تعود بكائن من النوع SizeF بعد ارسالك لوسيطتين الاولى تمثل النص والثاني كائن الخط Font:

```
Dim size As SizeF
Dim myText As String = "..."
Dim myFont As New Font (...)

size = gr.MeasureString(myText, myFont)
```

الصيغة السابقة للطريقة تفرض بان النص سيعرض في سطر واحد، يمكنك معرفة الارتفاع المطلوب ان حددت العرض Width عندما تنوي التفاف النص Wrap:

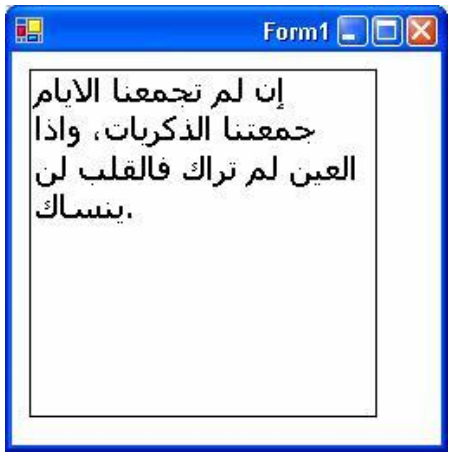
```
gr.MeasureString(myText, myFont, 200)
```

التفاف النص

الطريقة DrawString() يمكن ان تستقبل وسيطة من النوع RectangleF تحدد بها المنطقة التي تود من النص ان يظهر عليها، وبحيث يتم التفافه تلقائيا ان تجاوز حدودها:

```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 14)
Dim text As String = "ان لم تجمعنا الايام همعتنا الذكريات" & _
    "، واذا العين لم تراك فالقلب لن ينساك"

gr.DrawString(text, myFont, Brushes.Black, New _
    RectangleF(10, 10, 200, 200))
' ارسم حد اضافية
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 200, 200))
```

PDF Factory Pro trial version www.fastio.com



شكل 15-14: حجر النص في منطقة من النوع RectangleF.

ملاحظة

الفئة RectangleF تماثل الفئة من النوع Rectangle، والفرق الوحيد بينهما ان الاولى تستخدم قيم من النوع Single لخصائصها، بينما الثانية من النوع Integer.

اعلم انك ستتساءل عن طريقة ما تمكننا من محاذاة النص (**شكل 15-14**) بحيث يكون في الاتجاه العربي من اليمين إلى اليسار Right-to-Left، والإجابة ستكون شافية ووافية عن طريق الكائن StringFormat والذي يستحق فقرة كاملة لقوته وجبروته.

الكائن StringFormat

يمكنك ارسال الكائن StringFormat إلى طريقة سياق الجهاز ()DrawString، يحتوي هذا الكائن على مجموعة إضافية من الطرق والخصائص الموجه للتحكم في المخرجات الحرفية، لديك مثلا الخاصية Alignment والتي تتحكم في محاذاة النص والتي تكون إما Near (لليسار)، Center (الوسط)، أو Far (اليمين):

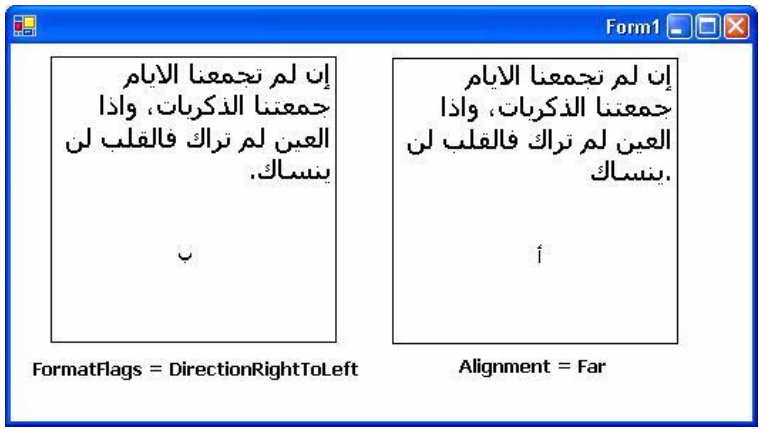
```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 14)
Dim sf As New StringFormat()
Dim text As String = "ان لم تجمعنا الايام جمعتنا الذكريات" & _
    "، واذا العين لم تراك فالقلب لن ينساك"

sf.Alignment = StringAlignment.Far
gr.DrawString(text, myFont, Brushes.Black, _
    New RectangleF(10, 10, 200, 200) , sf)

' ارسم حد اضافية
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 200, 200))
```

عند الحديث عن مخرجات الحروف العربية فلا تستخدم خاصية المحـــاذاة Alignment، وانما اعتمد على الخاصية FormatFlags لتغيير اتجاه النص Direction Text بارسال القيمة DirectionRightToLeft لها (**شكل 15-15 بالصفحة التالية**):

```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 14)
Dim sf As New StringFormat()
Dim text As String = "ان لم تجمعنا الايام جمعتنا الذكريات" & _
    "، واذا العين لم تراك فالقلب لن ينساك"

sf.FormatFlags = StringFormatFlags.DirectionRightToLeft
gr.DrawString(text, myFont, Brushes.Black, _
    New RectangleF(10, 10, 200, 200) , sf)
' ارسم حد اضافية
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 200, 200))
```

ملاحظة

عند تغيير اتجاه النص عن طريق الخاصية FormatFlags، فان اتجاه النص في الخاصية Alignment سيصبح معاكس، اي ان القيمة Near تحاذي النص إلى اليمين والقيمة Far إلى اليسار.



شكل 15-15: الفرق بين المحاذاة واتجاه النص.

قد لا تلاحظ الفرق بين تأثير المحاذاة واتجاه النص في (**الشكل 15-15**)، ولكن لو أعدت التدقيق ستلاحظ اختلاف موقع نقطة نهاية الجملة. ولكن كان هذا الفرق طفيفا، فصدقني ان الاختلاف اكبر بكثير من كونه تغيير موقع النقطة، جرب مثلا تعديل قيمة المتغير Text بحيث يحتوي على جملة تشمل كلمات إنجليزية وعربية، سيظهر الفرق عظيما هذه المرة (**شكل 15-16**).

شكل 15-16: الفرق بين المحاذاة واتجاه النص.

ملاحظة

ان كنت من مستخدمي Windows المتمرسين، فدعني أخبرك ان تغيير اتجاه النص هو تماما ما تحدثه المفاتيح [Shift] + [Ctrl] اليمنى.

المزيد ايضا، يمكنك محاذاة النص بالنسبة للمنطقة Rectangle من منظور عامودي عوضا عن افقي عن طريق الخاصية LineAlignment التي إما تكون Near (فوق)، Center (وسط)، أو Far (اسفل).

المحاذاة الكلية Justify:

تعتمد حروف لغتنا الجميلة على الكشيدة للمحاذاة الكلية Justify للفقرة، حاولت البحث في مكتبة MSDN على اي وسيلة أو قيمة تمكننا من تطبيق الكشيدة للمحاذاة الكلية للفقرة، ولكن محاولتي – مع الاسف- لم تجد الا دعم لهذه المحاذاة مع ادوات Web Forms والخاصة بصفحات HTML فقط. لذلك، كان علي تطوير الفئة ArJustify التي تستخدم الكشيدة لمحاذاة النص (**شكل 15-17**).

شكل 15-17: المحاذاة الكلية باستخدام الكشيدة عن طريق الفئة ArJustify.

تتطلب الفئة ArJustify خمس وسيطات في مشيدها هي: سياق الرسم أو الجهاز، النص المراد محاذاته، كائن الخط Font، عرض المنطقة، واسند القيمة True ان اردت محاذاة نص السطر الاخير. تحتوي الفئة على الخاصية JustifiedText والتي تعود بالنص بعد إضافة الكشيدات له:

```
Dim myText As String

myText = New ArJustify(gr, myText, myFont, 200, True).JustifiedText

gr.DrawString(myText, myFont, Brushes.Black, _
    New RectangleF(250, 10, 200, 200), sf)
```

ان نظرنا في داخل شيفرة الفئة، فسنجد أهم اجرائين بهما هما JustifiedText() و AddKashidas()، الاول يقوم بتوزيع الكلمات وفصلها لمعرفة أقصى عدد من الكلمات يمكن للسطر ان يحتويه، والثاني يقوم بالاضافة الفعلية للكشيدة:

```
Class ArJustify
    …
    …
    …
    Public ReadOnly Property JustifiedText() As String
        Get
            Return Me.text
        End Get
    End Property

    Private Sub JusitfyLines()
        Dim counter As Integer = 0

        Me.textLines = Me.text.Split(" "c)

        Do While counter <= UBound(Me.textLines)
            If gr.MeasureString(Me.textLines(counter), Me.font).Width
_
                = Me.width Then
                ' لا تفعل شئ

            ElseIf gr.MeasureString(Me.textLines(counter), _
                Me.font).Width > Me.width Then

                ' عليك فصل الكلمة هنا
            Else
                ' ادمج الكلمات
                counter += 1
                Do While counter <= UBound(Me.textLines)
                    If gr.MeasureString(Me.textLines(counter - 1) & _
```

```
                        " "c & Me.textLines(counter), _
                            Me.font).Width = Me.width Then

                    Me.textLines(counter - 1) &= " "c & _
                        Me.textLines(counter)
                    RemoveWord(counter)
                    counter -= 1
                    Exit Do
                ElseIf gr.MeasureString(Me.textLines(counter - 1)_
                    & " "c & Me.textLines(counter), Me.font).Width
                        > Me.width Then

                    counter -= 1
                    Me.AddKashidas(counter)
                    Exit Do
                Else
                    Me.textLines(counter - 1) &= " "c & _
                        Me.textLines(counter)
                    RemoveWord(counter)
                End If
            Loop
            If counter >= UBound(Me.textLines) Then
                Me.AddKashidas(counter - 1)
            End If
        End If
        counter += 1
    Loop
    Me.text = Me.text.Join(" "c, Me.textLines)
End Sub



Private Sub AddKashidas(ByVal index As Integer)
    Dim counter As Integer
    Dim canAddKashida As Boolean
    Dim allowedAfterLetters As String = "ضصثقفغعهخحجشسيبلتنمكطئ"
    Dim allowedBeforeLetters As String = _
    "ذضصثقفغعهخحجدشسيبلاأتنمكطئؤرىةوزظ_"

    If Me.JustifyLastLine = False AndAlso index = _
        UBound(Me.textLines) Then
        Exit Sub
    End If

    If Me.textLines(index).Length <= 1 Then
        Exit Sub
    Else
        counter = Me.textLines(index).Length - 2
    End If


    Do While gr.MeasureString(Me.textLines(index), Me.font).Width
_
```

PDF Factory Pro trial version www.fastio.com

```
            < Me.width

            If allowedAfterLetters.IndexOf(Me.textLines(index). _
                Chars(counter)) <> -1 AndAlso _
                allowedBeforeLetters.IndexOf(Me.textLines(index). _
                Chars(counter + 1)) <> -1 Then

                canAddKashida = True
                If gr.MeasureString(Me.textLines(index).Insert( _
                    counter + 1, "_"c), Me.font).Width > Me.width Then

                    Exit Do
                Else
                    Me.textLines(index) = Me.textLines(index).Insert(
_
                        counter + 1, "_"c)
                    Dim counter2 As Integer
                    For counter2 = counter To 0 Step -1
                        If Me.textLines(index).Chars(counter2) = " "c
Then
                            counter = counter2 - 1
                            Exit For
                        End If
                    Next
                    If counter2 <= 0 Then counter = _
                        Me.textLines(index).Length - 2
                End If
            Else
                counter -= 1
                If counter <= 0 Then
                    If canAddKashida Then
                        counter = Me.textLines(index).Length - 2
                    Else
                        Exit Do
                    End If
                End If
            End If
        Loop
    End Sub
End Class
```

كما تلاحظ في الشيفرة السابقة، تعتمد الفئة ArJustify على المصفوفات من النوع String وهذا هو احد اكبر أسباب بطئها، مع ذلك يمكنك الاعتماد على الفئة StringBuilder لزيادة الـــسرعة، كما يمكن لك تغيير الخوارزميات المتبعة بها.

انظر ايضا

قد تحتاج إلى العودة إلى الفصل السادس **الفئات الأساسية** ان اردت معرفة الغرض من الفئة StringBuilder.

المزيد ايضا، يمكنك الاستفادة من الفئة ArJustify لتنسيق محاذاة حروف وكلمات الشعر العربي (**الشكل 15-18**).

شكل 15-18: الاستفادة من الفئة ArJustify لتنسيق الشعر العربي.

ملاحظة

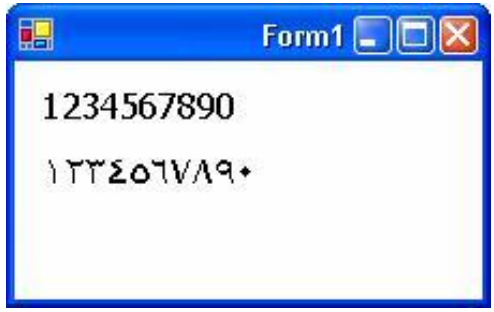
كان غرضي من الفئة ArJustify توضيح فكرة تطبيق المحاذاة الكلية فقط، فلا تعتمدها في مشاريعك لاني لم اجري اي اختبارات إضافية عليها ولم اصل إلى الدقة المطلوبة التي توفرها معظم برامج معالجة النصوص.

الارقام الهندية:

لا استخدم الارقام الهندية في حياتي اليومية وذلك لعدم قناعتي بها، مع ذلك تتبع هذه الارقام الاعدادات الإقليمية لمجموعة كبيرة من الدول العربية، وقد يفضلها العديد من مستخدمين نظم التشغيل Windows.

بشكل مبدئي، الارقام ستظهر بالاعتماد على اعدادات البيئة الحالية في جهازك الشخصي، مع ذلك يمكن تعديلها عن باستدعاء الطريقة SetDigitSubstitution() التي تتطلب منك وسيطتين الاولى **معرفة LCID** للدولة والثانية ونظام الاعداد، الشيفرة التالية تستخدم الاعدادات الإقليمية **العربية (المملكة العربية السعودية)**، وترى مخرجاتها في **(الشكل 15-19)**:

```
Dim gr As Graphics = e.Graphics
Dim sf As New StringFormat()
Dim myFont As New Font("Tahoma", 13)
Dim myText As String = "1234567890"

gr.DrawString(myText, myFont, Brushes.Black, _
    New RectangleF(10, 10, 280, 200), sf)

sf.SetDigitSubstitution(&H401, StringDigitSubstitute.Traditional)

gr.DrawString(myText, myFont, Brushes.Black, _
    New RectangleF(10, 40, 280, 200), sf)
```



شكل 15-19: الارقام الهندية والعربية مدعومة في اعدادات **العربية (المملكة العربية السعودية)**.

انظر ايضا

للحصول على معرفات LCID لباقي الدول العربية، راجع الفصل السادس **الفئات الأساسية** وبالتحديد الجدول الموجود صفحة 245.

عودة إلى الأدوات Controls

قبل ان اختم هذا الفصل بودي عرض هذا القسم والتي ستفتح لك آفاق واسعة لتحسين التصميم الخارجي والهيكلة الظاهرية للأدوات دون الحاجة لاعادة بنائها من جديد.

تمكنك بعض الادوات (كالاداة ListBox والقائمة MenuBox) من الوصول إلى سياق الجهاز Device Context الخاص بها، ماذا يعني هذا؟ يفيدك كرم الاداة بإعطائك مرجع لسياق رسمها من اضافة بعض اللمسات الفنية لمخرجات الاداة نفسها، مثلا هب انك تريد اضافة عناصر لاداة ListBox تمكن المستخدم من اختيار لون معين، يستحسن في هذه الحالة ان يظهر اللون كخلفية لكل عنصر من عناصرها (**شكل 15-20**).

شكل 15-20: الاستفادة من سياق رسم الاداة لتغيير مظهر عناصرها.

حتى اريك مثالا يستفيد من سياق جهاز الاداة، اضف اداة من النوع ListBox واكتب هذه الشيفرة في حدث Load الخاص بنافذة النموذج:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...

    Structure ColorName
        Dim ArabicName As String
        Dim Brush As Brush
    End Structure
    Dim colorItems(6) As ColorName

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        Dim ci As ColorName

        colorItems(0).ArabicName = "ابيض"
        colorItems(0).Brush = Brushes.White
        colorItems(1).ArabicName = "اصفر"
        colorItems(1).Brush = Brushes.Yellow
        colorItems(2).ArabicName = "فضي"
        colorItems(2).Brush = Brushes.Silver
        colorItems(3).ArabicName = "ازرق"
        colorItems(3).Brush = Brushes.Blue
        colorItems(4).ArabicName = "برتقالي"
        colorItems(4).Brush = Brushes.Orange
        colorItems(5).ArabicName = "احمر"
        colorItems(5).Brush = Brushes.Red
        colorItems(6).ArabicName = "اخضر"
        colorItems(6).Brush = Brushes.Green

        For Each ci In colorItems
            ListBox1.Items.Add(ci.ArabicName)
        Next

    End Sub
End Class
```

بالنسبة للاداة ListBox، فهي تحتوي على الخاصية DrawMode التي تسند لها القيمة OwnerDrawFixed لتسمح لك برسم عناصرها بنفسك، وبعد ذلك عليك اسناد قيمة مناسبة للخاصية ItemHeight لتحدد ارتفاع العنصر بالبكسل.

ستضع شيفرات الرسم بين فكي الحدث DrawItem والذي يتم تنفيذه بمجرد طلب الاداة منك رسم عنصر من عناصرها، يرسل هذا الحدث مع وسيطته مجموعة من الخصائص كالخاصية Bounds التي تحدد المنطقة المراد رسمها، الخاصية Index التي تمثل رقم العنصر المراد رسمه، والخاصية State التي تخبرك حالة العنصر (كان يكون محدد، ممكن، غير ممكن ...الخ)،

اضف الشيفرة التالية بين فكي حدث الاداة DrawItem لتصمم اداة ملونة العناصر كالموجودة في (الشكل 15-20):

```
Private Sub ListBox1_DrawItem(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.DrawItemEventArgs) _
        Handles ListBox1.DrawItem

    Dim gr As Graphics = e.Graphics
    Dim myFont As New Font("Tahoma", 13)
    Dim rect As Rectangle = e.Bounds
    rect.Inflate(-3, -3)

    gr.FillRectangle(colorItems(e.Index).Brush, rect)

    If CBool(e.State And DrawItemState.Selected) Then
        rect.Inflate(-1, -1)
        gr.DrawRectangle(SystemPens.Highlight, rect)
        rect.Inflate(-1, -1)
        gr.DrawRectangle(SystemPens.Highlight, rect)
    End If

    gr.DrawString(colorItems(e.Index).ArabicName, _
        myFont, Brushes.Black, _
        rect.Width - gr.MeasureString(colorItems(e.Index).ArabicName,_
        myFont).Width, rect.Y)

    myFont.Dispose()
End Sub
```

مكتبة +GDI غنية جدا وتحتوي على كثير من الفئات التي لم أتطرق لك، إن أردت التخصص في مجال الصور والرسوم بمكتبة +GDI، فلديك مستندات NET Documentation. لتبحر بين صفحاتها. والآن يمكنك الاستفادة من كل ما تعلمته في الفصول الثلاث السابقة، لتطور مشاريع من النوع Custom Controls أو Windows Services بالفصل التالي.

PDF

مواضيع متقدمة

عندما تتقدم في عمرك البرمجي مع Windows Forms، قد تحتاج إلى تطوير نوعية إضافية من المشاريع كالأدوات الخاصة Custom Controls أو خدمات Windows Services، والتي يمكنك انجازها بكافة الأساليب المتبعة في الفصول السابقة.

في هذا الفصل اختم معك الجزء الثالث **تطوير تطبيقات Windows** بالتحدث عن موضوع تطوير الأدوات الخاصة وخدمات Windows، كما سأختم الفصل بعرض سريع لمجموعة من الفئات الاضافية والتي قد تحتاجها يوما من الأيام لبرامجك الموجه للعمل تحت Windows.

تطوير أدوات خاصة

تأتي مع رزمة Visual Studio .NET مجموعة كبيرة من الأدوات جاهزة الاستخدام، وقد تطرقنا إلى معظمها في الفصل الرابع عشر **الأدوات Controls**، مع ذلك قد تحتاج إلى تطوير المزيد من الأدوات الخاصة Custom Controls والتي تستخدمها في مشاريعك الخاصة، أو مشاريع أخرى.

قبل الاستمرار في هذا القسم، علينا ان نتفق على ثلاث مصطلحات هي: **المؤلف Author** وهو الشخص الذي يقوم ببرمجة الأداة، **المبرمج Programmer** وهو الشخص الذي يستخدم هذه الأداة في مشاريعه، **والمستخدم User** وهو المستخدم النهائي للبرنامج الذي يعرض الأداة.

المزيد ايضا، **مرحلة التأليف Authoring Time** هي مرحلة تصميم وبرمجة الأداة، **ووقت التصميم Design Time** هو وقت التصميم الخاص بالمبرمج Programmer لتصميم برنامجه واستخدام الأداة، اما **وقت التنفيذ Run Time** فهو الوقت تنفيذ البرنامج النهائي والذي يستخدم الأداة.

ثلاثة أساليب يمكنك اتباعها لتطوير أدوات خاصة أسهلها وراثة أداة سابقة، استخدام مجموعة من الأدوات الجاهزة، أو إنشاء أداة مستقلة من الصفر. في الفقرات التالية اعرض لك هذه الاساليب الثلاث:

وراثة أداة

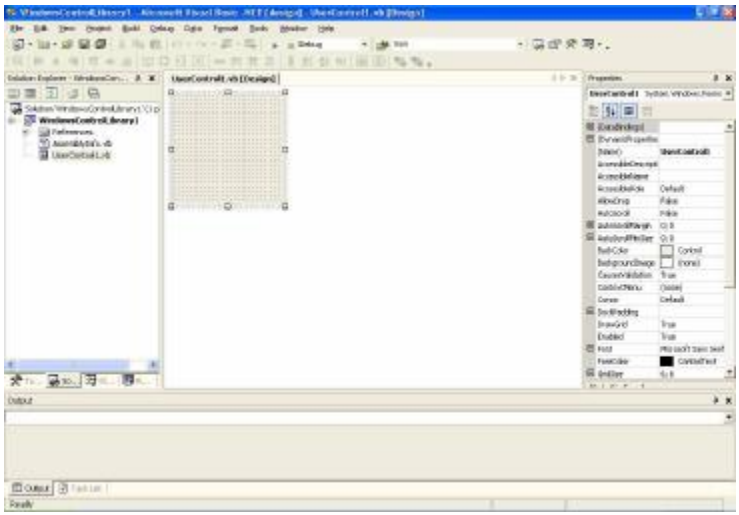
اسهل اسلوب تتبعه لانشاء أداة خاصة هو بوراثة أداة معينة لتطويرها وزيادة خصائصها وطرقها حتى تلائم احتياجاتك، واذكر أني في الفصل الرابع عشر **الأدوات Controls** اضطررت لوراثة مجموعة من الأدوات (كـ TreeView) حتى أتمكن من تطبيق تقنية المرآة Mirroring عليها بالشكل الصحيح.

يعيب هذا الاسلوب ان الأداة لابد من ان تكون قابلة للاشتقاق الوراثي حتى تتمكن من إتباعه، فبعض الأدوات (كالأداة ProgressBar أو ImageList)، لا يمكنك اشتقاقها وراثيا وبذلك لن تتمكن من اتباع هذا الاسلوب.

مع ذلك، اغلب الأدوات –المستخدمة بكثرة- قابلة للوراثة (كالأدوات Label، TextBox، Picture، ListView وغيرها) وبذلك يمكنك زيادة خصائصها وطرقها بحيث تناسب احتياجاتك الخاصة.

تأليف الأداة:

الخطوة الاولى لانشاء الأداة هو انشاء مشروع لها، اختر الامر New->Project من قائمة File وحدد الرمز Windows Control Library، ستنشئ لك بيئة التطوير Visual Studio .NET مشروع جديد من هذا النوع يحتوي على الملف UserControl1.vb (**شكل 16-1**). بالنسبة للنافذة المفتوحة امامك، فهي تمثل الأداة التي تود تصميمها والتعامل معها سيكون كما تتعامل مع نوافذ النماذج.



شكل 16 - 1: مشروع جديد من النوع Windows Control Library.

ملاحظة

المشاريع من النوع Windows Control Library ما هي إلا مشاريع من النوع Class Library تقليدية، ولا يوجد فرق جوهري بينما، فكل ما في الامر هو اختلاف الشيفرة البرمجية التي تولدها بيئة التطوير Visual Studio .NET بشكل مبدئي، واعدادات مختلفة في صندوق حوار خصائص المشروع Project Property Pages.

غير اسم المشروع إلى الاسم TextBoxExProject وافتح نافذة محرر الشيفرة التابعة للأداة UserControl1 وامسح كل شئ فيها، واكتب هذه الفئة التي تشتق الأداة TextBox:

```
Public Class TextBoxEx
    Inherits System.Windows.Forms.TextBox

    Sub New()
        MyBase.New()
    End Sub
End Class
```

بمجرد تعريفك للفئة TextBoxEx السابقة، فانك قد انتهيت من بتطوير أداة TextBox كاملـــة، تحتوي على جميع خصائص، طرق، واحداث الأداة TextBox. مع ذلك، لا توجد أي فائدة مـــن الأداة الجديدة ما لم تضيف أعضاء بها.

سنضيف الخاصية AutoSelect والتي تسند لها القيمة True ان اردت تحديد كافة الـــنص بمجرد حصول الأداة على تركيزها:

```
Private m_autoSelect As Boolean
 Public Property AutoSelect() As Boolean
     Get
         Return m_autoSelect
     End Get
     Set(ByVal Value As Boolean)
         m_autoSelect = Value
     End Set
 End Property
```

انسب حدث نضع في امر تحديد كافة النص في الأداة هو الحدث Enter، والذي سيتم تنفيذه بمجرد انتقال التركيز إلى الأداة:

```
Private Sub TextBoxEx_Enter(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Enter

    Me.SelectAll()
End Sub
```

وبذلك، تكون الشيفرة النهائية للأداة بهذا الشكل:

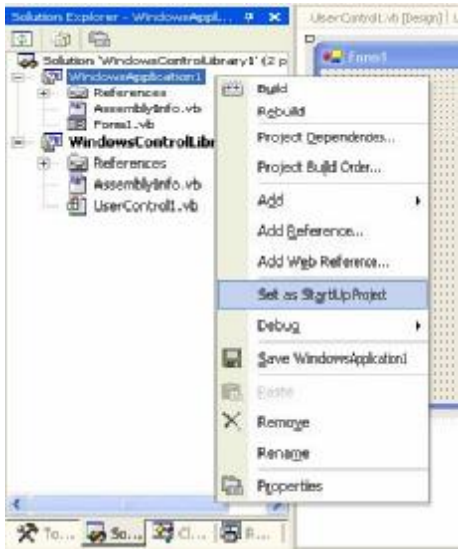
```
Public Class TextBoxEx
    Inherits System.Windows.Forms.TextBox


    Sub New()
        MyBase.New()
    End Sub

    Private m_autoSelect As Boolean
    Public Property AutoSelect() As Boolean
        Get
            Return m_autoSelect
        End Get
        Set(ByVal Value As Boolean)
            m_autoSelect = Value
        End Set
    End Property
```

```
    Private Sub TextBoxEx_Enter(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles MyBase.Enter

        Me.SelectAll()
    End Sub
End Class
```

اختر الامر Bulid TextExProject من قائمة Build لتترجم المشروع إلى ملف من النوع DLL لتتمكن من استخدام هذه الأداة في مشاريع اخرى.

برمجة الأداة:

اختر الامر الفرعي New Project من الامر Add Project من القائمة File حتى تنشئ مشروع من النوع Windows Application في نفس الحل Solution، ولا تنسى جعله **المشروع الابتدائي Startup Project** (حتى يتم تنفيذه بعد الضغط على المفتاح [F5])، وذلك بالضغط على رمزه في نافذة مستكشف الحل Solution Explorer واختيار الامر Set as Startup Project من القائمة المنبثقة (**شكل 16-2**).



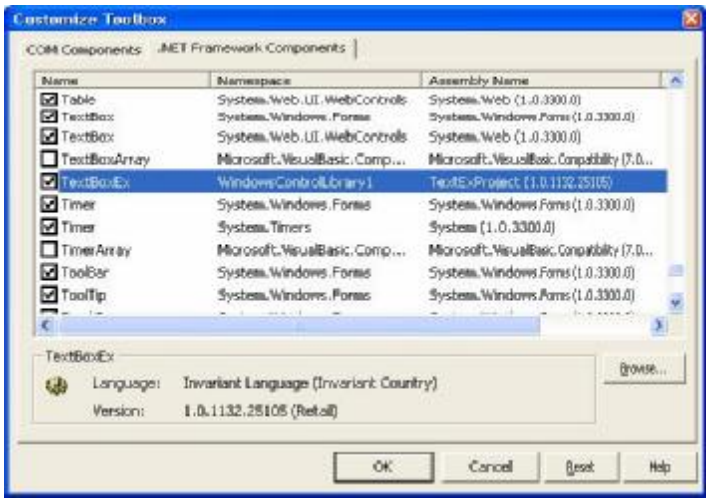
شكل 16-2: تحديد المشروع الابتدائي Startup Project.

PDF Factory Pro trial version www.fastio.com

ملاحظة

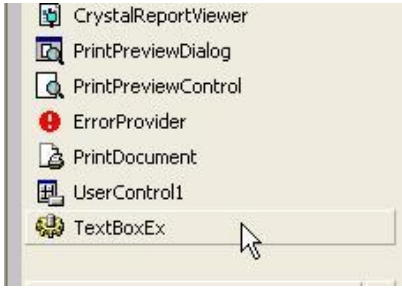
بامكانك إنشاء حل جديد New Solution ان اردت استخدام الأداة، ولكن يفضل إنشاء مشروع Windows Application في نفس الحل ان كنت لازلت تحت مرحلة التجربة واختبار الأداة، لتتمكن من الانتقال إلى شيفرة الأداة وتحريرها ومن ثم ترجمتها بسرعة.
ميزة اخرى في دمج المشاريع في نفس الحل يظهر جليا لحظة التنقيح Debugging، حيث ستتعامل مع أدوات التنقيح (والخاصة ببيئة Visual Studio .NET) وكأن كلا المشروعين مشروع واحد، فقيم المتغيرات ومسارات التنفيذ وغيرها من الامور، يمكنك تتبعها بين ملفات كلا المشروعين.
ميزة ثالثة، عندما تقوم باعادة ترجمة الأداة، ستقوم بيئة التطوير بنسخ ملف مجمع الأداة المحدث إلى مجلد البرنامج الذي يستخدمها بشكل تلقائي.

بينما تعمل على نافذة مصمم النماذج، انقر بزر الفأرة الايمن على صندوق الأدوات ToolBox واختر الامر Customize Toolbox ليظهر لك صندوق حوار بعنوان Customize Toolbox، انتقل إلى خانة التبويب NET Framework Component. واضغط على الزر Brows (في اسفل يمين صندوق الحوار)، ابحث عن ملف الأداة في نفس المسار الذي قمت بترجمة ملف الأداة فيه. وبعد إيجاده والضغط على الزر Open، ستلاحظ انه تم تحديد الأداة في صندوق الحوار Customize ToolBox (**شكل 16-3**).



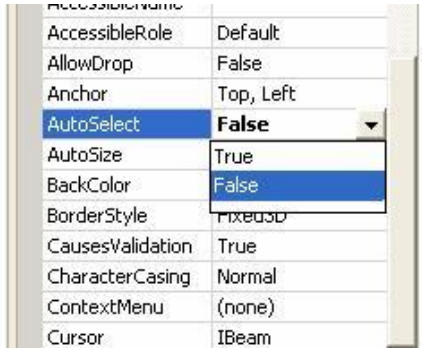
شكل 16-3: إضافة الأداة TextBoxExProject في صندوق الأدوات.

تأكد من تحديدك للأداة واضغط على الزر OK، ستلاحظ ان بيئة التطوير Visual Studio .NET. قد اضافت رمز للأداة في صندوق الأدوات بالاسم TextBoxEx (**شكل 16-4**)، يمكنك الان استخدامها وإضافة نسخة منها في نافذة النموذج.



شكل 16-4: ظهور الأداة TextBoxEx في صندوق الأدوات.

تعامل مع الأداة TextBoxEx كما تتعامل مع الأداة TextBox التقليدية فهي مشتقة منها، الشيء الظريف الذي يثبت لنا تطوير الأداة هو ظهور الخاصية AutoSelect (والتي اضفناها في فئة الأداة الجديدة) بنافذة الخصائص (**شكل 16-5**).



شكل 16-5: ظهور الخاصية AutoSelect في نافذة الخصائص.

جرب اضافة ثلاث أو اربع أدوات TextBoxEx على نافذة النموذج، واسند القيمة True لكافة خصائصها، قم بنقل التركيز وقت التنفيذ بين الأدوات المختلفة، ستلاحظ انه سيتم تحديد كافة النص بمجرد حصول الأداة على تركيزها.

هذه ببساطة شديدة الفكرة من اتباع الاسلوب الاول لتصميم الأدوات الخاصة، وهو وراثة الأدوات لزيادة قوتها باضافة خصائص وطرق اضافية تناسب احتياجاتك. ولا تنسى انه يمكن للأداة المشتقة ان تصل إلى طرق وخصائص الأداة القاعدية Base Control، اضف إلى ذلك قدرتك على اعادة قيادة Overrides واعادة تعريف Overloads اعضاء الفئة القاعدية.

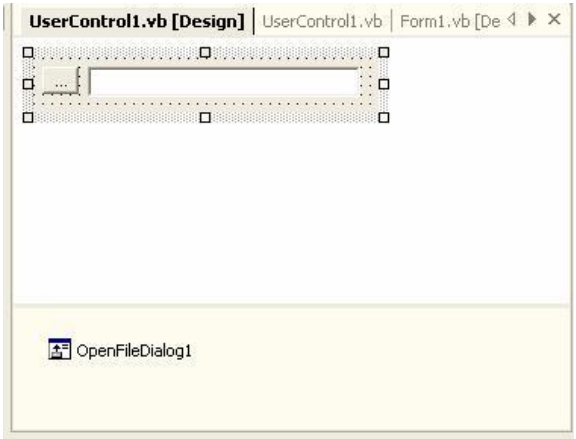
الفقرة التالية تمكنك من تطوير أداة خاصة تحضن مجموعة من الأدوات الجاهزة.

حضن مجموعة من الأدوات

عليك معرفة، ان الأداة الخاصة التي تؤلفها ليست سوى نافذة نموذج Form تقليدية تمكنك من وضع الأدوات عليها وتعديل خصائصها وتصميمها بشكل مرئي Visual كما تفعل مع نوافذ النماذج، والاختلاف بينهما بسيط جدا لا يتعدى الخصائص والطرق، فالأداة الخاصة لا تحتوي على خصائص تتبع منطقيا إلى نوافذ النماذج (كالخصائص WindowState، ShowInTaskBar، MaximizeBox ...الخ).

سأحاول في هذه الفقرة إعطائك مثالا يعرض لك كيفية الاستفادة من اسلوب حضن الأدوات لاختصار العمليات المتكررة، كعملية وضع زر Button مرافق لأداة TextBox يمكن المستخدم من الضغط عليها لكتابة اسم الملف.

أنشئ أداة خاصة Custom Control جديدة، وأضف على جبهتها ثلاث أدوات هي: TextBox، Button، و OpenFileDialog (**شكل 16-6**).



شكل 16-6: أداة خاصة حاضنة لمجموعة أدوات.

عدل الخاصية Anchor للأداة TextBox (بحيث تكون: فوق، يمين، ويسار)، وفي الحدث Click التابع للأداة Button، سطر الشيفرة التالية:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    If OpenFileDialog1.ShowDialog = DialogResult.OK Then
        TextBox1.Text = OpenFileDialog1.FileName
    End If
End Sub
```

والان يمكنك الانتقال من مرحلة التأليف إلى التصميم ووضع الأداة على نافذة النموذج، ودون الحاجة لكتابة شيفرة مصدرية واحدة، تستطيع ان تمكن المستخدم من كتابة اسم الملف في أداة النص بعد اختياره من صندوق حوار فتح.

نقاط ضعها في عين الاعتبار:

عند حضن مجموعة خارجية من الأدوات في أداتك الخاصة، عليك ان ترفق ملفات المجمعات Assemblies مع أداتك الخاصة إلى المبرمج الذي سيتعامل معها، ففي الأداة التي صممناها للتو، تستخدم مجموعة من الأدوات الجاهزة والتابعة لمجمعات إطار عمل NET. Framework، صحيح ان احتمالية وجود هذه ملفات هذه المجمعات كبير جدا في اجهزة المبرمجين، إلا انه يستحسن إبلاغهم بان أدواتك الخاصة تحتاج إلى وجود هذه المجمعات، حتى يحاول المبرمجون توفير نسخ منها ليس في أجهزتهم وحسب بل حتى في اجهزة المستخدمين لبرامجهم.

بالنسبة لمحدد الوصول للأدوات المحضونة في مرحلة التأليف، فلابد ان يكون Public حتى تمكن المبرمج من الوصول إلى أعضاء الأدوات المحضونة، وتذكر ان الأدوات المحضونة تكون محددات وصلها Friend بشكل افتراضية، لذلك قد تحتاج إلى تغييرها من نافذة الشيفرة أو من الخاصية Modifiers في نافذة الخصائص.

صحيح ان اختيار محدد الوصول Public يريحك من عناء التصنيف الفرعي Subclassing لاعضاء الأدوات المحضونة، إلا انه سيمكن المبرمج من الوصول إلى الأداة المحضونة وكأنه مؤلف الأداة الخاصة، وقد لا يقوم المبرمج بالتعامل مع الأدوات المحضونة بالشكل المناسب، الامر الذي قد يؤثر على كامل سلوك تنفيذ الأداة الخاصة وباقي الأدوات المحضونة بها. (تخيل ان قام المبرمج –مثلا- بتغيير موقع الأداة المحضونة TextBox في المثال السابق بطريقة الخطأ).

ملاحظة

اقصد بالتصنيف الفرعي لأعضاء الأدوات المحضونة في السياق السابق، عملية تعريف إجراءات عامة Public لأعضاء الأدوات التي لم تستخدم معها محدد الوصول Public، مثال في الأداة الخاصة السابقة:

```
Public Property FileName() As String
    Get
        Return Me.TextBox1.Text
    End Get
    Set(ByVal Value As String)
        Me.TextBox1.Text = Value
    End Set
End Property
```

إنشاء أداة مستقلة

في هذه الحالة، فانك لن تعتمد على أدوات اخرى لتصميم اداتك الخاصة، فلن ترث أداة اخرى ولن تحضن أداة اخرى، بل ستعتمد على كائنات GDI+ لترسم وتصمم واجهة الأداة.

عيوب هذا اسلوب تغطي على محاسنه، فعليك اعادة كتابة معظم الطرق والخصائص بنفسك، وستكون المسئول الاول والأخير عن كل صغيرة وكبيرة في الشيفرة المصدرية، كما ان الجهد والوقت الذي تبذله في انجاز الأداة يعادل تصميم برنامج يعتمد كامل يعتمد على أدوات جاهزة –اسأل مجرب ولا تسأل طبيب.

لا توجد ميزة استطيع إعطائك إياها عن هذا الاسلوب إلا ميزة السيطرة والتحكم المطلق الذي تجنيه على كل جزء من أجزاء الأداة، فأنت ستكون مؤلف الشيفرة وهي ملكا لك.

مع ذلك، كلي ثقة وأمل بإخواني المبرمجين العرب في تطوير أدوات خاصة قوية وداعمة لبرامجنا العربية، خاصة بعد الدعم العربي القوي الذي وفرته لنا Microsoft في اطار عمل NET Framework..

وكمثال على تطوير هذا النوع من الأدوات، سنقوم بإنشاء أداة بالاسم Rotator تمكن المبرمج من قلب النصوص بمقدار زاوية معينة **(شكل 16-7)**، تحتوي هذه الأداة على خاصيتين Text و Angle، الخاصية الثانية تمثل مقدار زاوية الدوران، وبالنسبة للنص الموجود في الخاصية Text فيمكن رسمه في الحدث Paint والخاص بالأداة:

```
Private Sub Rotator_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) Handles
MyBase.Paint

    Dim gr As Graphics = e.Graphics
    Dim sf As New StringFormat()

    gr.TranslateTransform(Me.ClientRectangle.Width / 2, _
        Me.ClientRectangle.Height / 2)

    gr.RotateTransform(Me.Angle)

    gr.DrawString(Me.Text, Me.Font, Brushes.Black, New RectangleF(0, _
    0, Me.ClientRectangle.Width, Me.ClientRectangle.Height))

    gr.ResetTransform()
    gr.DrawRectangle(Pens.Black, Me.ClientRectangle)

End Sub
```

شكل 16-7: الأداة المستقلة Rotator.

ينقص الأداة Rotator شئ واحد تقريبا وهو إضافة شيفرات لتوسيط النص وسط الأداة بالضبط حتى يتم عرضها بالشكل الصحيح (ستحتاج إلى استخدام الدوال المثلثية Math.Sin() و Math.Cos() لعمل ذلك).

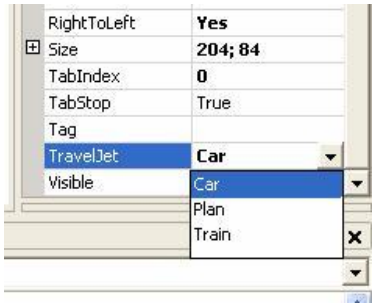
لمسات فنية إضافية

دعني أرشدك في هذه الفقرة إلى بعض اللمسات الفنية التي تضفيها على اداتك الخاصة لتجعلها كباقي أدوات Windows Forms حقيقية وكأنها مصممة لأغراض تجارية.

في البداية حاول استخدام التركيبات من النوع Enum دائما للخاصية ان كانت القيم التي تسندها اليها معينة:

```
Enum Flight
    Car
    Plan
    Train
End Enum

Property TravelJet() As Flight
    …
    …
End Property
```

الثمرة الابتدائية التي تجنيها من استخدام التركيبات من النوع Enum، هو التسهيل على المبرمج لاسناد قيمها سواء برمجيا من نافذة محرر الشيفرة، أو وقت التصميم عن طريق نافذة الخصائص (**شكل 16-8**).

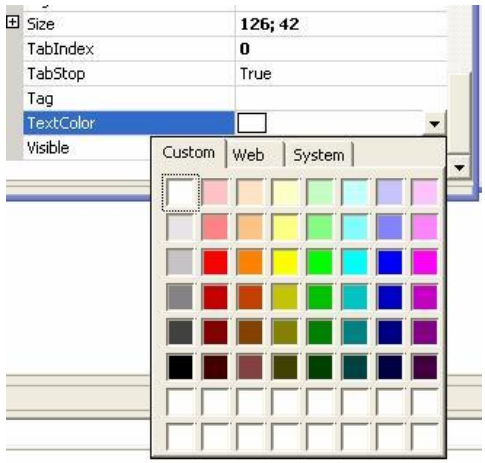


شكل 16-8: ظهور قائمة تمثل القيم المختلفة للخاصية في نافذة الخصائص.

المزيد ايضا، حاول استخدام الأنواع المعرفة من البيانات ان كانت قريبة من الانواع البيانات الشهيرة والاكثر استخداما في مشاريع Windows Forms (كـ Font، Color، Size، Rectangle ...الخ)، فذلك يسهل على المبرمج ايضا اسناد قيمها بطرق مختلفة (**شكل 16-9 بالصفحة التالية**):

```
Property TextColor() As Color
    …
    …
End Property
```

PDF



شكل 16-9: أضيف لوح الوان للخاصية TextColor في نافذة الخصائص.

ملاحظة

ان كنت تنوي عرض حقول وخصائص فئاتك الخاصة في نافذة الخصائص بشكل شجري، عليك استخدام كائن من النوع TypeConverter –راجع مكتبة MSDN للحصول على مثال لاستخدامه.

بعض الأدوات تحتوي على مجموعة من الخصائص تسمى **خصائص وقت التصميم Design time properties**، وهي خصائص لا يمكن تعديل قيمها إلا وقت التصميم، تستطيع عمل ذلك باختبار قيمة الخاصية DesignMode والتي تعود بالقيمة True ان كانت الوقت هو وقت التصميم و False وقت التنفيذ:

```
Public Property ModeType() As Boolean
    Get
        If Not Me.DesignMode Then
            Throw New Exception("هذه الخاصية وقت التصميم فقط")
        End If
    End Get
    Set(ByVal Value As Boolean)
        If Not Me.DesignMode Then
            Throw New Exception("هذه الخاصية وقت التصميم فقط")
        End If
    End Set
End Property
```

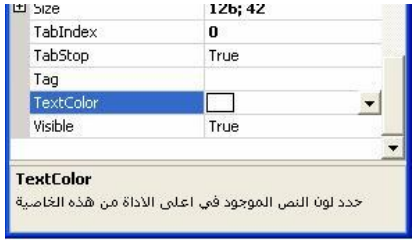
ملاحظة

رغم ان اختبار قيمة الخاصية DesignMode تابعة لمرحلة التأليف Authoring time للأداة، إلا ان القيمة التي تعود بها تابعة لمرحلة التصميم Design time من قبل المبرمج وليس مؤلف الأداة.

مواصفات إضافية:

يوفر لك مجال الاسماء System.ComponentModel مجموعة كبيرة من المواصفات Attributes التي تعطيك تحكما اكثر من اعضاء أدواتك الخاصة، لديك مثلا المواصفة Description والتي تحدد فيها وصف نصي للخاصية يفيد المبرمج (**شكل 16-10**):

```
Imports System.ComponentModel
…
…
<Description("حدد لون النص الموجود في اعلى الأداة من هذه الخاصية")> _
Property TextColor() As Color
    …
    …
End Property
```



شكل 16-10: وصف الخاصية ظهر في اسفل نافذة الخصائص.

ان عرفت خصائص للقراءة فقط ReadOnly في اداتك الخاصة، فلا يوجد داعي من اظهارها على نافذة الخصائص، حيث ان المبرمج لن يتمكن من تعديل قيمها، اسند القيمة False إلى مشيد المواصفة Browsable حتى يتم إخفاء الخاصية من نافذة الخصائص:

```
<Browsable(False)> _
ReadOnly Property IsSomething() As Boolean
    Get
        …
        …
        …
    End Get
End Property
```

اذا كانت الخاصية حرفية أو أي نوع اخر تعتقد انه يتأثر بالاعدادات الإقليمية، فمن المفضل ارسال القيمة True لمشيد المواصفة Localizable، حتى يتم حفظ نسخة من قيمة الخاصية لكل دولة النماذج المحلية Localized Forms في ملفات المصادر:

```
<Localizable(True)> _
Property TitleName() As String
    …
    …
End Property
```

المواصفتين DefaultProperty و DefaultEvent تحددان فيهما اسم الخاصية الافتراضية (التي يتم تحديدها في نافذة الخصائص بمجرد إنشاء نسخة من الأداة)، والحدث الافتراضي (الذي يتم اختياره في محرر الشيفرة لحظة النقر المزدوج وقت التصميم على الأداة:

```
<DefaultProperty("Text"), DefaultEvent("Click")> _
Public Class MyUserControl
    Inherits System.Windows.Forms.UserControl
    …
    …
End Class
```

اخيرا، استخدم المواصفة ToolboxBitmap لتحدد فيها الرمز أو الايقونة التي تود ان تظهر بها الأداة في صندوق الأدوات ToolBox (انسب حجم للصورة 16 x 16 بكسل):

```
<ToolboxBitmap ("C:\MyIcon.ico")> _
Public Class MyUserControl
    Inherits System.Windows.Forms.UserControl
    …
    …
End Class
```

تطوير خدمات Windows

في هذا القسم اعرض لك طريقة تطوير مشاريع تعرف بخدمات Windows Services.

ملاحظة

الفئات المستخدمة في هذا القسم مشمولة في مجال الأسماء التالي:

```
Imports System.ServiceProcess
```

مع ذلك، معظم انواع المشاريع (كـ Windows Application) لا تدعم مجال الاسماء السابق، لذلك عليك استخدام صندوق حوار المراجع Add References واضافة المجمع System.ServiceProcess.dll. اما المشاريع من النوع Windows Services فهي تشمله بشكل تلقائي.

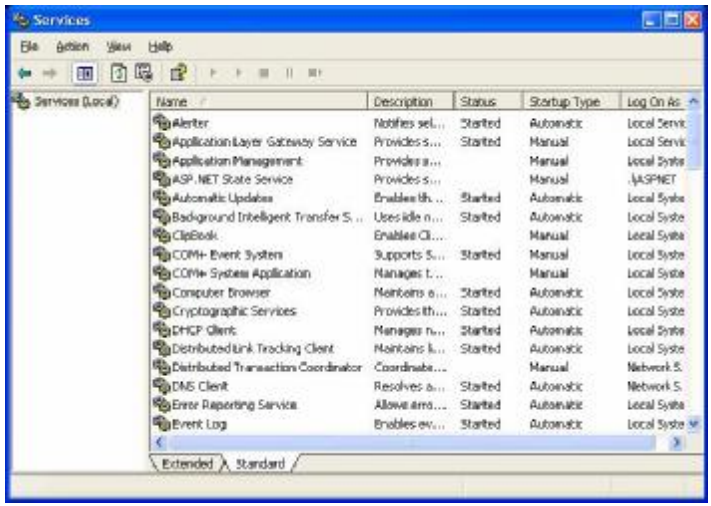
مقدمة إلى خدمات Windows

خدمة Windows Service ما هي إلا برنامج تنفيذي تقليدي EXE يتم تشغيله بمجرد عملية اقلاع النظام System Booting –أي بمجرد بدء تشغيل نظام التشغيل. الفكرة من خدمات Windows موجه لتلك النوعية من البرامج التي تعمل فترة طويلة ولا تحتاج إلى ردة فعل من المستخدم Interaction، وفي الحقيقة معظم خدمات Windows لا تحتوي على واجهة استخدام User Interface، فالتوجه الذي تتبعه هو ان تعمل كخادم Server يقوم بمهمات معينة.

توجد مجموعة كبيرة من خدمات Windows كخدمة Internet Information Server (المألوفة بالاختصار خادم IIS)، كخدمة Microsoft SQL Server، Microsoft Proxy Server وغيرها، الغرض منها تنفيذ مهام إدارية Administrating، توجيه Directing، مراقبة Watching، ...الخ دون الحاجة لوجود مستخدم على الجهاز.

يمكنك معرفة جميع خدمات Windows الموجودة في جهازك الشخصي بالنقر المزدوج على الرمز Services من المجموعة Administrative Tools في لوحة التحكم Control Panel (**شكل 16-11**)

PDF Factory Pro trial version www.fineprint.com

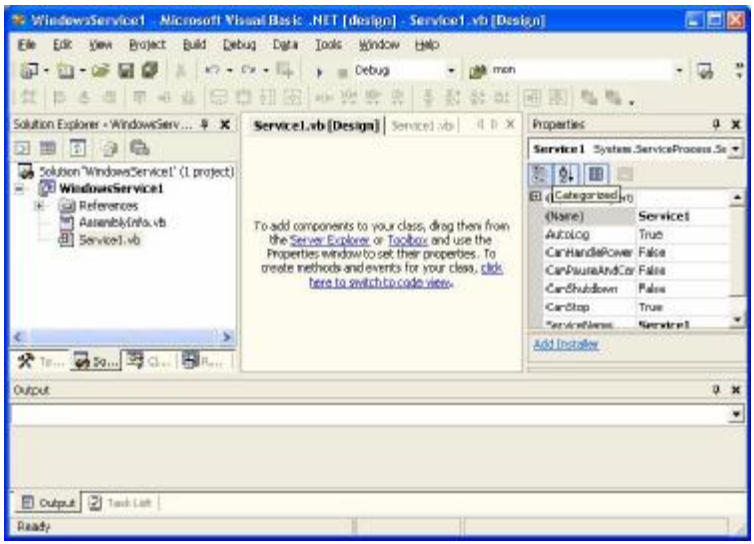


شكل 16-11: خدمات Windows الموجودة في الجهاز.

يمكنك الضغط بزر الفأرة الايمن على أي خدمة من الخدمات الموجودة في الجهاز واختيار امر من ثلاثة اوامر تميز خدمات Windows هي: **Start** لتنفيذ الخدمة، **Stop** لايقاف عمل الخدمة، و **Pause** للايقاف المؤقت للخدمة (ستحتاج إلى الضغط على **Resume** لتنفيذ الخدمة بعد الوقف المؤقت).

إنشاء مشاريع من النوع Windows Service

كل ما هو مطلوب منك اختيار الامر New->Project من قائمة File وتحديد الرمز Windows Service لإنشاء مشروع يعمل كخدمة Windows، ستنشى لك بيئة التطوير Visual Studio .NET. مشروعا جديدا يحتوي على الملف Service1.vb (**شكل 16-12**).



شكل 16-12: مشروع من النوع Windows Service.

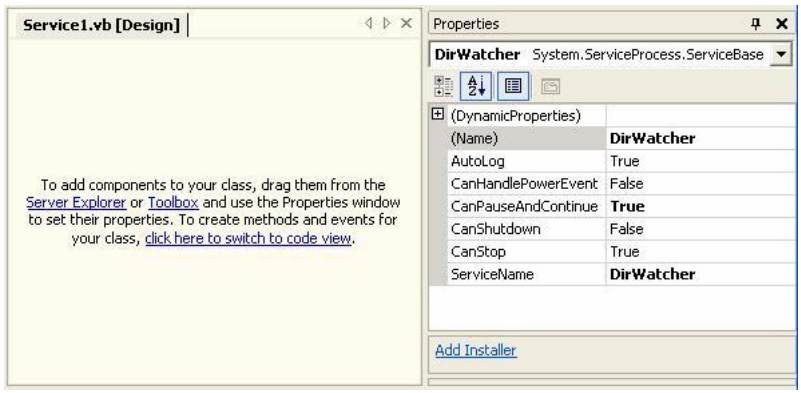
خدمة Windows التي سنقوم ببنائها عبارة عن مراقبة مجلد النظام Windows ومعرفة الملفات التي تم حذفها منذ بداية تشغيل الخدمة (مع بداية تشغيل الجهاز)، حتى إنهائها، غير اسم المشروع من WindowsService1 إلى FolderWatcher.

في الفقرة السابقة عرفتك على معنى خدمة Windows بمنظور المستخدم، لذلك دعني اعيد صياغة التعريف بمنظور مبرمج NET. والذي يعرف خدمة Windows على انها فئة Class مشتقة من الفئة القاعدية System.ServiceProcess.ServiceBase.

من التعريف السابق نستنتج ان المشروع الذي أنجزناه للتو يحتوي على خدمة Windows باسم Service1، انتقل إلى نافذة الخصائص وعدل الخاصيتين (Name) و ServiceName من Service1 إلى DirWatcher.

اسنادك للقيمة True للخاصية CanStop تحدد فيها ما اذا كانت الخدمة قابلة للتوقف Stop (من لوحة التحكم Control Panel أو أي برنامج اخر)، اما الخاصية CanPauseAndContinue فتحدد فيها قابلية الايقاف المؤقت Pause للخدمة (**شكل 16-13**).

PDF



شكل 16-13: تعديل خصائص الخدمة.

تصحيح الشيفرة

ان قمت بفتح نافذة محرر الشيفرة للخدمة DirWatcher السابقة، سترى هذه الشيفرة المولدة:

```
Imports System.ServiceProcess

Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase

#Region " Component Designer generated code "

    Public Sub New()
        MyBase.New()

        ' This call is required by the Component Designer.
        InitializeComponent()

        ' Add any initialization after the InitializeComponent() call

    End Sub

    'UserService overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' The main entry point for the process
```

```
    <MTAThread()> _
    Shared Sub Main()
        Dim ServicesToRun() As System.ServiceProcess.ServiceBase

        ' More than one NT Service may run within the same process. To
add
        ' another service to this process, change the following line
to
        ' create a second service object. For example,
        '
        '   ServicesToRun = New System.ServiceProcess.ServiceBase ()
{New Service1, New MySecondUserService}
        '

        ServicesToRun = New System.ServiceProcess.ServiceBase() {New
Service1()}

        System.ServiceProcess.ServiceBase.Run(ServicesToRun)
    End Sub

    'Required by the Component Designer
    Private components As System.ComponentModel.IContainer

    ' NOTE: The following procedure is required by the Component
Designer
    ' It can be modified using the Component Designer.
    ' Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        '
        'DirWatcher
        '
        Me.CanPauseAndContinue = True
        Me.ServiceName = "DirWatcher"

    End Sub

#End Region

    Protected Overrides Sub OnStart(ByVal args() As String)
        ' Add code here to start your service. This method should set
things
        ' in motion so your service can do its work.
    End Sub

    Protected Overrides Sub OnStop()
        ' Add code here to perform any tear-down necessary to stop
your service.
    End Sub

End Class
```

الجزء المكتوب بالخط السميك Bold في الشيفرة السابقة، يتطلب منك تنقيحه يدويا (حيث لم يظهر فيه تأثير تغيير الخاصية (Name))، عدل الكلمة من Service1 إلى DirWathcer:

```
ServicesToRun = New System.ServiceProcess.ServiceBase() {New
DirWathcer()}
```

بهذا نكون قد صححنا الخطأ وأصبحنا جاهزين لكتابة شيفرات الخدمة، ولكن قبل ذلك دعني أعرفك على الفئة FileSystemWatcher.

الفئة System.IO. FileSystemWatcher

لست هنا بصدد شرح كافة تفاصيل الفئة FileSystemWatcher فهي لا تتعلق بموضوع خدمات Windows بشكل مباشر، ولكن بودي عرض طريقة استخدامها بشكل سريع حيث انني سأستخدمها في مشروع الخدمة FolderWatcher الذي صممناه.

الغرض الأساسي من الفئة FileSystemWatcher يتضح من اسمها في مراقبة الملفات، حيث تحدد في خاصيتها Path مسار المجلد الذي تود مراقبة ملفاته:

```
Dim FW As New System.IO.FileSystemWatcher()

FW.Path = "C:\Windows"
```

أي تعديل في ملفات هذا المجلد (حذف، اضافة، أو تعديل خصائص ملفاته) سيتم إبلاغك به فورا عن طريق مجموعة من الاحداث توفرها لك الفئة كـ Created، Deleted، Changed، و Renamed (عليك قنص الاحداث اما بـ WithEvents أو AddHandler بنفسك). ترسل هذه الاحداث مع وسيطتها كائن من نوع FileSystemEventArgs يحتوي على مجموعة من الخصائص منها FullPath التي تعود بالمسار الكامل للملف الذي طرأ عليه التعديل:

```
Dim WithEvents FW As New System.IO.FileSystemWatcher()

Public Sub FW_Deleted(ByVal sender As Object, _
    ByVal e As System.IO.FileSystemEventArgs) Handles FW.Deleted

    MsgBox (e.FullPath)
End Sub
```

PDF

حتى يتم تنفيذ الاحداث السابقة في الوقت المناسب، عليك اسناد القيمة True لخاصيته EnableRaisingEvents لبدء عملية مراقبة المجلد، وقد تسند القيمة True ايضا للخاصية IncludeSubdirectories لتشمل المراقبة المجلدات الفرعية:

```
FW.IncludeSubdirectories = True
FW.EnableRaisingEvents = True
```

هذا كل ما تحتاج معرفته حول الفئة System.IO.FileSystemWatcher لتطبقه في خدمة Windows التي نصممها، ان اردت مزيد من التفاصيل حول هذه الفئة يمكنك مراجعة مكتبة MSDN.

كتابة الشيفرات

تحتوي الفئة القاعدية System.ServiceProcess.ServiceBase على طريقتين هما OnStart() و OnStop()، يتم تنفيذ الاولى لحظة تشغيل الخدمة والثانية لحظة ايقافها، عليك اعادة قيادة Overrides هاتين الطريقتين حتى تضع الشيفرات المتطلب تنفيذها لحظة بداية تنفيذ وايقاف الخدمة:

```
Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase
    …
    …
    Dim WithEvents FW As New System.IO.FileSystemWatcher()

    Protected Overrides Sub OnStart(ByVal args() As String)
        FW.Path = "C:\Windows"
        FW.IncludeSubdirectories = True
        FW.EnableRaisingEvents = True
    End Sub

    Protected Overrides Sub OnStop()
        FW.EnableRaisingEvents = False
    End Sub
End Class
```

بما اننا أسندنا القيمة True للخاصية CanPauseAndContinue (**شكل 16-13**)، فعلينا اعادة قيادة الطريقتين OnPause() و OnContinue() حتى نضف الشيفرات المطلوب تنفيذه لحظة الايقاف المؤقت Pause وإكمال التنفيذ بعد الإيقاف المؤقت Resume:

```
Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase
    …
    …
    Protected Overrides Sub OnPause()
        FW.EnableRaisingEvents = False
    End Sub

    Protected Overrides Sub OnContinue()
        FW.EnableRaisingEvents = True
    End Sub
End Class
```

الخطوة الاخيرة هي قنص احداث الكائن FileSystemWatcher لتحديد ماذا نريده مـــن الخدمة ان تفعل عندما يتم حذف ملف من ملفات المجلد Windows. الشيفرة التالية ستقوم بكتابة اسم الملف الذي تم حذفه في ملف نصي Test.TXT (لا تنسى ان خدمات Windows لا تظهر أي واجهة استخدام، حيث لا يفترض وجود شخص على جهاز الخادم Server):

```
Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase
    …
    …
    Dim WithEvents FW As New System.IO.FileSystemWatcher()

    Private Sub FW_Deleted(ByVal sender As Object, _
        ByVal e As System.IO.FileSystemEventArgs) Handles FW.Deleted

        Dim textFile As New System.IO.StreamWriter("C:\Test.TXT", _
            True)

        textFile.WriteLine("تم حذف الملف" & e.FullPath)

        textFile.Close()
    End Sub
End Class
```

انظر ايضا

لمزيد من التفاصيل حول التعامل مع وحدات التخزين Streams والملفات النصية، قد تحتاج إلى قراءة الفصل الثامن **الملفات والمجلدات**.

تسجيل الخدمة

ان استعجلت وحاولت تنفيذ الخدمة (بالضغط على المفتاح [F5])، ستظهر لك بيئة التطوير Visual Studio .NET رسالة خطأ (**شكل 16-14**) مفادها ان الخدمة لا يمكن ان يتم تنفيذها كما تفعل مع تطبيقات EXE الاخرى، اذ عليك تركيبها وتثبيتها في الجهاز ليتم تسجيلها في النظام ومن ثم يمكنك تنفيذها.

شكل 16-14: رسالة خطأ مفادها ان الخدمة لا يمكن تنفيذها دون تسجيلها.

حتى تمكن الخدمة من التركيب، عليك كتابة مجموعة من الشيفرات الإضافية لتعريف كائنات من فئات الخاصة بتثبيت وتسجيل الخدمة، مع ذلك لست بحاجة لفعل ذلك يدويا، حيث يمكن لبيئة التطوير Visual Studio .NET من توليد الشيفرة تلقائيا نيابة عنك. اضغط على الرابط المعنون **Add Installer** في اسفل نافذة الخصائص والخاصة بفئة الخدمة (**شكل 16-13 صفحة 590**)، ستلاحظ أن بيئة التطوير أنشأت لك ملف ProjectInstaller.vb يحتوي على أداتين هما ServiceProcessInstaller1 و ServiceInstaller1 (**شكل 16-15**).

شكل 16-15: ملف تثبيت الخدمة.

حدد الأداة الاولى ServiceInstaller وانتقل إلى نافذة الخصائص، ستلاحظ وجود مجموعة من الخصائص والخاصة بها، يمكنك استكشاف معانيها من مستندات .NET Documentation.

حيث لا يهمني في الوقت الحالي إلا الخصائص ServiceName، DisplayName، و StartType، الاولى لابد ان يكون اسم الخدمة مطابق للاسم الخدمة المراد تثبيتها (في حالتنا ستكون DirWatcher)، الثانية هو الاسم الظاهري للخدمة (اكتب -مثلا- "خدمة DirWatcher")، وبالنسبة للخاصية الثالثة فيمكنك اسناد القيمة Automatic ان اردت تنفيذ الخدمة بمجرد اقلاع النظام System Booting (يعني بمجرد تشغيل نظام التشغيل).

بالنسبة للأداة الاخرى ServiceProcessInstaller1 فعليك تحديد خاصيتها Account لتحديد نوع حساب المستخدم User Account الذي تعمل الخدمة فيه، اسند القيمة LocalSystem لها والتي تتجاهل حساب المستخدم وتعمل لاي مستخدم سواء في النظام الحالي أو على مستوى الشبكة المحلية. (راجع مكتبة MSDN لمزيد من التفاصيل حول القيم الاخرى).

قم الان بعملية ترجمة ملفات مشروع الخدمة ليتم توليد ملف ثنائي بالامتداد EXE، ستحتاج اليه عند استخدام الأداة InstallUtil.EXE والتي تقوم بالتثبيت الفعلي للخدمة بمسجل نظام التشغيل System Registry.

الأداة InstallUtil.EXE

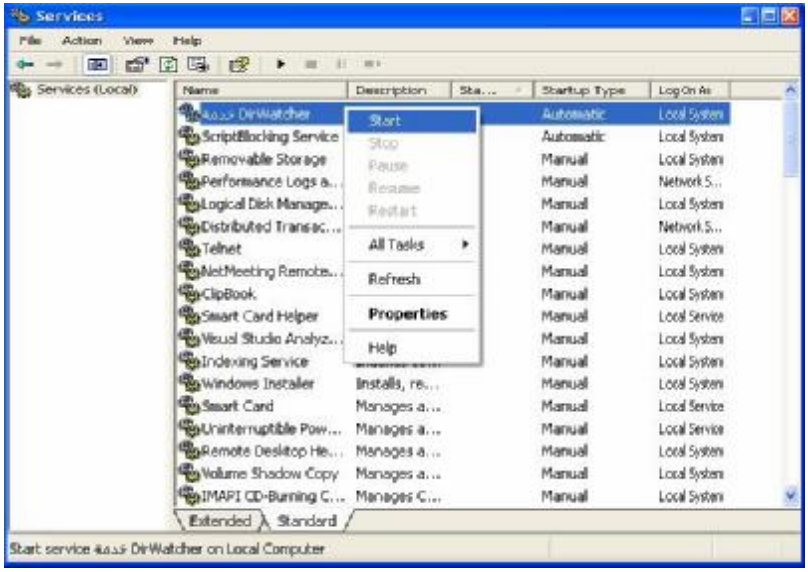
ان قمت بتشغيل موجه الاوامر Command Prompt فلا تنسى تنفيذ الملف corvars.bat (والذي تجده في المجلد X:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin) حيث يقوم بتحميل مسارات Paths الأداة InstallUtil.EXE ويسهل عليك الوصول لها، مع ذلك لست بحاجة إلى تنفيذ هذا الملف ان كنت قد شغلت نافذة موجه الاوامر من خلال الرمز Visual Studio .NET Command Prompt الموجود في المجموعة البرمجية Microsoft Visual Studio .NET بقائمة Start (تماما مثل ما فعلنا في الفصل الحادي عشر **المجمعات Assemblies** عندما استخدمنا أدوات الترجمة، الربط، والتسجيل) (**شكل 11 - 6**).

لتسجيل خدمتنا DirWatcher، اكتب اسم ملف الخدمة التنفيذي EXE مع الأداة InstallUtil:

```
C:\>InstallUtil FolderWatcher.EXE
```

ان تم كل شيء على ما يرام، سيتم تشغيل الخدمة بمجرد بدء اقلاع النظام، مع ذلك لست بحاجة إلى اعادة تشغيل الجهاز Restart لعمل ذلك، فتستطيع الانتقال إلى قائمة الخدمات Service في

المجموعة Administrative Tools في لوحة التحكم Control Panel، والضغط بزر الفأرة الايمن على خدمتنا ثم اختيار الامر Start من القائمة المنبثقة (**شكل 16-16**).



شكل 16-16: تشغيل خدمة DirWatcher من قائمة الخدمات في لوحة التحكم.

اخيرا، ان اردت الغاء تثبيت الخدمة وتسجيلها من النظام، استخدم نفس الأداة InstallUtil.EXE بنفس الصيغة السابقة، ولكن مع ارسال المدخل u/:

```
C:\>InstallUtil FolderWatcher.EXE /u
```

ملاحظة

عليك ايقاف عمل الخدمة Stop قبل الغاء تثبيتها. بالنسبة لخدمتنا DirWatcher فيمكنك ايقافها من قائمة الخدمات (**شكل** 16-16) وذلك باختيار الامر Stop من القائمة المنبثقة التي ستظهر بنقر زر الفأرة الايمن عليها.

فئات أخرى

ونحن على مشارف الانتهاء من عالم تطوير تطبيقات Windows، بودي عرض مجموعة من الفئات التي قد تحتاجها عند تطوير برامجك العاملة تحت نظم Windows، سواء كانت تطبيقات قياسية Windows Application، خدمات Windows Services، أو حتى أدوات خاصة Custom Controls.

الفئة Application

تمثل الفئة Application المجمع الحالي والذي يتم تنفيذه الان، ولا يمكنك إنشاء نسخة كائن منها باستخدام New، فكل مجمع يحتوي على كائن Application واحد. معظم خصائص وطرق الفئة Application مشتركة Shared Members، كما انها للقراءة فقط ReadOnly، كالخاصية ExecutablePath التي تعود بالمسار الكامل لملف المجمع الرئيسي، والخاصية StartupPath التي تعود بالمسار دون الملف:

```
MsgBox (Application.ExecutablePath) ' C:\Folder\MyAss.EXE
MsgBox (Application.StartupPath) ' C:\Folder
```

من الخصائص التي تعود بمعلومات حول المجمع: الخاصية CompanyName التي تعود باسم الشركة المضمونة في المجمع، الخاصية CurrentCulture التي تعود بالاعدادات الاقليمية للمجمع، الخاصية ProductVersion التي تعود بالاصدار، والخاصية ProductName التي تعود باسم المجمع.

اما الطرق، فتوجد الطريقة DoEvents التي توزع وقت المعالجة في نفس مسار التنفيذ احداث الأدوات المختلفة، الطريقة Exit() التي تنهي عمل المجمع بينما الطريقة ExitThread() تغلق جميع النوافذ التي تعمل في مسار التنفيذ الحالي.

راجع مكتبة MSDN لمزيد من التفاصيل حول اعضاء الفئة Application، وذلك لانني سأختم هذه الفقرة بذكر ثلاث احداث منها هي: ApplicationExit، ThreadExit، وIdle يتم تنفيذها بمجرد انهاء البرنامج، انهاء مسار تنفيذ، أو ان البرنامج في حالة الاستقرار Idle (أي لا توجد رسائل نظام أو أي مهام في طابور الرسائل يتطلب تنفذيها).

الفئة Cursor

النماذج والأدوات تحتوي على الخاصية Cursor والتي لم اتطرق لها في الفصول السابقة لحاجة ما في نفس يعقوب وربطها بالفئة Cursor في هذه الفقرة.

قبل ان ابدأ بالتحدث عن الفئة Cursor دعني اتحدث عن الخاصية Cursor والتابعة للأدوات الفئات، يمكنك اسناد قيمة إلى هذه الخاصية تمثل شكل مؤشر الفأرة من 28 شكل توفره لك الخاصية. يمكنك تخصيص شكل المؤشر عند مروره فوق كل أداة من الأدوات عن طريق خاصية الأداة Cursor:

```
Button1.Cursor = Cursors.No
```

اما ان اردت تخصيص أشكال مشيرة من عندك، فلن تجد اسهل من استخدام الفئة Cursor والتي يمكنك تحميل ملف المؤشر بإرساله إلى مشيدها:

```
Dim myCur As New Cursor ("C:\myCur.cur")
Button1.Cursor = myCur
```

ولا تنسى قتل كائن المؤشر عند عدم الحاجة اليه باستدعاء الطريقة Dispose():

```
myCur.Dispose()
```

المزيد ايضا، بدلا من تعيين شكل المؤشر لكل أداة على حده، يمكن اسناد قيمة إلى الخاصية المشتركة Current والتابعة للفئة Cursor حتى يتغير شكل المؤشر في كافة أدوات ونوافذ البرنامج:

```
Cursor.Current = Cursors.Hand
```

مع ذلك، ان كانت قيمة الخاصية Cursor والتابعة للأداة أو نافذة النموذج لا تساوي Default، فسيتم تغيير المؤشر عن المرور على الأداة أو النافذة بنفس القيمة الموجودة في خاصية الأداة أو النافذة Cursor.

لا ينحصر استخدام الفئة Cursor لتغيير شكل المؤشر، بل يشمل ايضا حكر المؤشر على نافذة أو أداة معينة، يمكنك عمل ذلك باسناد المنطقة إلى الخاصية المشتركة Clip (تتطلب كائن من النوع Rectangle)، الشيفرة التالية تحكر الفأرة في المنطقة الداخلية لنافذة النموذج:

```
Cursor.Clip = Me.RectangleToScreen(Me.ClientRectangle)
```

ولا تنسى تحرير الفأرة بإسناد القيمة Nothing إلى الخاصية Clip السابقة.

ملاحظة

الغرض من الطريقة RectangleToScreen() -والتي استخدمتها في الشيفرة الأخيرة- هو الحصول على المنطقة بالنسبة للشاشة وليس النموذج.

اخيرا، لديك الخاصية المشتركة Position والتي تمكنك من تحريك مؤشر الفأرة إلى نقطة من الشاشة:

```
Cursor.Position = New Point(0, 0)
```

الفئة SendKeys

تمكنك الفئة SendKeys من محاكاة لوحة المفاتيح Keyboard والضغط على المفاتيح، تستطيع تحديد المفاتيح التي ترغب بتفعيلها باستخدام الطريقة المشتركة Send(). الشيفرة التالية تقوم بارسال حروف اسمي ومن ثم الضغط على المفتاح [Enter]:

```
SendKeys.Send("تركي~")
```

الرمز السابق ~ يمثل مفتاح [Enter]، الرمز + مفتاح [Shift]، الرمز ^ مفتاح [Ctrl]، والرمز **%** مفتاح [Alt]، وان اردت استمرار الضغط على المفتاح [Shift] -مثلا- وإرفاقه بمفتاح اخر، استخدم الاقواس، فالشيفرة التالية ستكتب الحروف كبيرة Capital:

```
SendKeys.Send("+(turki)")
```

معظم المفاتيح الغير مطبوعة الاخرى تستخدم الاقواس المعكوفة { و } مثل {TAB}، {ESC}، {LEFT}، {RIGHT}، {F1}، {F2}، {F3}، ....الخ، راجع مستندات .NET Documentation لمزيد من التفاصيل.

PDF Factory Pro trial version www.fastio.com

ملاحظة

لا يمكن استخدام الطريقة ()Send لتنشيط برنامج آخر، والحل يتم اما يدويا بتنشيط ذلك التطبيق بالفأرة أو استخدامات إجراءات API كـ FindWindow و SetForegroundWindows.

الفئتان Registry و RegistryKey

تمكنك الفئتان Registry و RegistryKey من الوصول إلى مسجل النظام Windows Registry وقراءة محتوياته. وحتى يسهل عليك التعامل مع هاتين الفئتين، يفضل فهم واستيعاب تركيب مسجل النظام (**شكل 16-17**).

شكل 16-17: تركيب مسجل النظام Windows Registry.

يحتوي مسجل النظام على مجموعة كبير جدا جدا من **المفاتيح Keys** أشبه ما توصف بـه بمجلدات، كل مفتاح من هذه المفاتيح يحتوي على مجموعة من القيم –اشبه بالخصائص، كل قيمة من هذه القيم تسمى **بيان Data**، وكل بيان يحتوي على قيمة تسمى **Value**.

ان اردت التعامل مع الفئتين Registry و RegistryKey فالخطوة الاولى هي بالحصول على مرجع للمفتاح الجذري للمسجل، يمكنك الحصول على سبع مفاتيح جذرية عن طريق سبع خصائص مشتركة للفئة Registry والتي تعود بكائن من النوع RegistryKey:

PDF Factory Pro trial version www.fineprint.com

```
Dim root1 As RegistryKey = Registry.ClassesRoot
Dim root2 As RegistryKey = Registry.CurrentConfig
Dim root3 As RegistryKey = Registry.CurrentUser
Dim root4 As RegistryKey = Registry.DynData
Dim root5 As RegistryKey = Registry.LocalMachine
Dim root6 As RegistryKey = Registry.PerformanceData
Dim root7 As RegistryKey = Registry.Users
```

ملاحظة

كلا الفئتين Registry و RegistryKey مشمولتان في مجال الاسماء Microsoft.Win32، لذلك قد تقوم باستيراده لاختصار كتابة اسماء انواع البيانات في شيفراتك المصدرية:

Imports Microsoft.Win32

الكائنات المنشة من الفئة RegistryKey ستحتوي على ثلاث خصائص هي: Name اسم المفتاح، SubKeyCount عدد المفاتيح الفرعية للمفتاح الحالي، عدد البيانات التي يحتويها المفتاح الحالي.

يمكنك استكشاف المفاتيح الفرعية باستدعاء الطريقة GetSubKeyNames() والتي تعود بمصفوفة حرفية تشمل جميع المفاتيح الفرعية:

```
Dim name As String
For Each Name In root1.GetSubKeyNames
    …
    …
Next
```

ان كنت على دراية باسم المفتاح الفرعي، فيمكنك الوصول اليه مباشرة باستدعاء الطريقة OpenSubKey() لفتح ذلك المفتاح والعودة بكائن من النوع RegistryKey ايضا:

```
Dim IE As RegistryKey

IE = root1.OpenSubKey("Internet Explorer.Main")
```

ان فتحت كائن بالطريقة السابقة، عليك اغلاقه دائما:

```
IE.Close()
```

اما ان اردت تعديل قيمة بيانات المفتاح، فاستخدم الطريقة السابقة OpenSubKey() مع ارسال القيمة True لفتح المفتاح للكتابة التي تتم باستدعاء الطريقة SetValue() والتي تحدد فيها اسم البيان المراد تعديل قيمته، والقيمة المراد إسنادها له:

```
Dim IE As RegistryKey

IE = root1.OpenSubKey("Internet Explorer.Main")
Dim IE As RegistryKey

IE = root2.OpenSubKey("Internet Explorer.Main", True)
IE.SetValue("Search Page", "http://www.dev4arabs.com")
```

كما يمكنك استدعاء الطريقة GetValue() لقراءة قيمة البيان.

الفئة Help

يمكنك إنشاء ملفات التعليمات Help Files باستخدام مجموعة من البرامج الجاهزة –لعل أبرزها Microsoft HTML Help Compiler - والتي تنتج ملفات من النوع CHM، تستطيع الوصول إلى احد صفحات ملف التعليمات عن طريق الفئة Help (لم اتحدث عن بناء ملفات التعليمات في هذا الكتاب).

تحتوي هذه الفئة على طريقتين مشتركة Shared Methods، الطريقة الاولى ShowHelpIndex() تعرض صفحة الفهرس Index (تتطلب وسيطة تمثل مرجع النافذة التي تتبع لها):

```
Help.ShowHelpIndex(Me, "C:\helpfile.chm")
```

اما الطريقة الاخرى ShowHelp() فتم اعادة تعريفها Overloads باربع صيغة، الصيغة الاولى تفتح لك القسم الخاص بعرض المحتويات Contents:

```
Help.ShowHelp(Me, "C:\helpfile.chm")
```

والصيغة الثانية ترسل وسيطة اضافية تحدد القسم الذي توده، كخانة تبويب البحث Search:

```
Help.ShowHelp(Me, "C:\helpfile.chm", HelpNavigator.Find)
```

PDF Factory Pro trial version www.fastio.com

ان ارسلت القيمة HelpNavigator.Topic إلى الوسيطة الثالثة في الصيغة السابقة، عليك ارفاقها برقم الموضوع في ملف التعليمات لتستخدم الصيغة الثالثة للطريقة ()ShowHelp:

```
Help.ShowHelp(Me, "C:\helpfile.chm",HelpNavigator.Topic ,3)
```

اما الصيغة الاخيرة فتمكنك من ارسال كلمة محجوزة keyword عرفت في ملف التعليمات لحظة إنشائه:

```
Help.ShowHelp(Me, "C:\helpfile.chm", "برمجة")
```

يدعمك إطار عمل NET. Framework بمئات الفئات التي تسهل حياتك البرمجية لتطوير تطبيقات Windows، في أربعة فصول حاولت تلخيص جولتي حول ما قرأته من مستندات NET. Documentation.. مع ذلك، ينقصك جزء هام وكبير في حياتك البرمجية مع إطار عمل NET Framework. وهو موضوع الجزء الرابع **برمجة قواعد البيانات**.

PDF