**الفصل الثالث عشر**

**بناء أدواتك الخاصّة**

* [**القوّة.. والسهولة.. ومنتهى الروعة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\13-%20بناء%20الأدوات%20الخاصّة\1-%20القوّة.htm)
* [**تطوير الأدوات الموجودة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تطوير%20الأدوات%20الموجودة.htm)
* [**إنشاء أدوات مركّبة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\13-%20بناء%20الأدوات%20الخاصّة\3-%20إنشاء%20أدوات%20مركّبة.htm)
* [**بناء أدوات بأشكال خاصّة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\13-%20بناء%20الأدوات%20الخاصّة\4-%20بناء%20أدوات%20بأشكال%20خاصّة.htm)
* [**الأدوات من النوع :ActiveX**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\13-%20بناء%20الأدوات%20الخاصّة\5-%20الأدوات%20من%20النوع.htm)

**القوّة.. والسهولة.. ومنتهى الروعة:**

لا ريب أنّك ستستمع جدّا بهذا الفصل، ففيه ستتعلّم كيف تنشئ الأدوات Controls التي تروق لك، بحيث توفّر على نفسك وقت إعادة تصميم النماذج التي تستعملها كثيرا.

وكما هي القاعدة في VB.Net، يجب أن تتوقّع أنّك تتعامل مع أسهل لغة برمجةٍ معروفة، بخلاف القوّة الجديدة التي اكتسبتها مؤخّرا.. باختصار: يمكنك أن تُنشئ بهذه اللغة أدوات رائعة عالية الكفاءة.

ويكفي أن أخبرك أنّك تستطيع أن ترث أيّ أداة من أدوات اللغة، لتعدّل فيها أو تضيف بعض الخصائص والوسائل إليها.. إنّ هذا سيضمن لك مبدئيّا أنّك لن تبدأ من الصفر في كلّ مرّة!

أمّا ما سيثلج صدرك تماما، فهو أنّك تستطيع الآن إنشاء أدوات جديدة حتّى في شكلها، دون أن ترتبط بالأشكال التقليديّة للأدوات (خاصة الأشكال المستطيلة المستفزّة)، وذلك عن طريق استخدام دوال الرسم والتلوين لرسم الأداة كما يحلو لك، بل ويصل الأمر إلى قدرتك على تغيير شكل قوائم النموذج Menus!

أعتقد أنّي قد أثرت فضولك بما يكفي، (لدرجة أنّني أعتقد أنّ هذا المكان مناسب لوضع إعلان عن أيّ منتج تافه على غرار ما تفعله المسلسلات!!!)، فتعال نبدأ بدون تضييع لحظة واحدة زائدة.

**تطوير الأدوات الموجودة**

* [**تغيير شكل بعض الأدوات المعروفة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تطوير%20الأدوات%20الموجودة\1-%20تغيير%20شكل%20الأدوات.htm)
* [**تحسين الأدوات الموجودة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تطوير%20الأدوات%20الموجودة\2-%20تحسين%20الأدوات%20الموجودة.htm)

**تغيير شكل الأدوات**

بعض أدوات الويندوز تمنحك الخصائص والوسائل اللازمة لتغيير شكلها.. هذه الأدوات هي القوائم الرئيسيّة Menus والقوائم بأنواعها ListBoxes.. وبهذا تستطيع تغيير خطّ كلّ عنصر في القائمة الرئيسيّة، أو تغيير لون خلفيّة أو لون خطّ كلّ عنصرٍ في القائمة، أو وضع صورة في خلفيّة كلّ عنصر، أو تغيير ارتفاع كلّ عنصر... إلخ.

ولفعل هذا، يجب أن تقوم ببرمجة الحدثين "قياس العنصر" MeasureItem وفيه تستطيع تحديد مقاييس العنصر، و"رسم العنصر" DrawItem.

تعال نرى ذلك:

* [**تغيير شكل القوائم الرئيسيّة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\1-%20تغيير%20شكل%20الأدوات\1-%20تغيير%20شكل%20القوائم%20الرئيسيّة.htm)
* [**تغيير شكل عناصر القائمة ListBox:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\1-%20تغيير%20شكل%20الأدوات\2-%20تغيير%20شكل%20عناصر%20القائمة.htm)

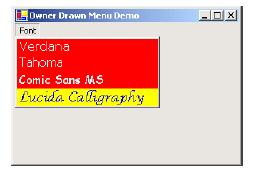
**تغيير شكل القوائم الرئيسيّة:**

يمتلك كائن عنصر القائمة MenuItem خاصيّة OwnerDraw، التي يجب عليك جعل قيمتها True حتّى تتدخّل في رسم العناصر.

في هذه الحالة، فإنّ الحدثين MeasureItem و DrawItem ينطلقان قبل رسم كلّ عنصر.. هذان هما الحدثان اللذان تستطيع فيهما تغيير أبعاد العنصر وشكله.

وفي الحدث MeasureItem يمكنك أن تتعامل مع أبعاد العنصر باستخدام الخاصيتين ItemWidth و ItemHeight.

أمّا في الحدث DrawItem، فيمكنك رسم نصّ العنصر بأيّ خطّ ولون، كما يمكنك استخدام وسائل الرسم لرسم ما تريد من الأشكال، أو لوضع صورة في خلفيّة العنصر.



ولتوضيح الفكرة، سننشئ الآن تطبيقا يحتوى على قائمة بأسماء الخطوط، كلّ اسم منها مكتوب بالخطّ الذي يمثّله، بحيث يمكن للمستخدم أن يعاين شكل الخطّ بنفسه.

لا تنسَ تغيّر خاصيّة OwnerDraw للقائمة إلى True لكلّ عنصر، قبل أن تكتب الكود التالي:

**Private Sub Font\_MeasureItem(ByVal sender As Object, \_**

**ByVal e As MeasureItemEventArgs) \_**

**Handles FontVerdana.MeasureItem, \_**

**FontSimplifiedArabic. MeasureItem ', \_**

**' ……………………………………**

**' اكتب أسماء كل قوائم الخطوط التي تريد أن يستجيب لها هذا الحدث**

**Dim FontName As String = CType(sender, MenuItem).Text**

**Dim fnt As New Font(FontName, 12, FontStyle.Regular)**

**Dim itemSize As SizeF**

**itemSize = e.Graphics.MeasureString(FontName, fnt)**

**e.ItemHeight = itemSize.Height**

**e.ItemWidth = itemSize.Width**

**End Sub**

**Private Sub Font\_DrawItem(ByVal sender As Object, \_**

**ByVal e As DrawItemEventArgs) \_**

**Handles FontVerdana.DrawItem, \_**

**FontSimplifiedArabic.DrawItem ', \_**

**' ……………………………………**

**' اكتب أسماء كل قوائم الخطوط التي تريد أن يستجيب لها هذا الحدث**

**Dim FontName As String = CType(sender, MenuItem).Text**

**Dim fnt As New Font(FontName, 12, FontStyle.Regular)**

**Dim R As New RectangleF(e.Bounds.X, e.Bounds.Y, \_**

**e.Bounds.Width, e.Bounds.Height)**

**e.Graphics.FillRectangle(Brushes.PaleTurquoise, R)**

**e.Graphics.DrawString(FontName, fnt, Brushes.White, R)**

**End Sub**

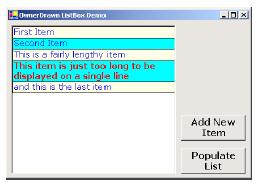
أعرف أعرف.. ستقول لي إنّ كتابة أسماء كلّ قوائم الخطوط في جملة Handles يبدو عقيما.. عامّة هناك وسيلة لعلاج هذا العقم!.. احذف جملة Handels نهائيّا من الحدثين، وأنشئ المستجيبات للأحداث عند تحميل النموذج، باستخدام جملة تكراريّة تمرّ عبر كلّ عناصر القائمة وتستخدم الوسيلة [AddHandler](file:///C:\\Users\\Administrator\\01-%20أساسيّات%20اللغة\\05-%20Forms%20النماذج\\4-%20بناء%20نماذج%20مرنة\\4-%20إنشاء%20مستجيبات%20للأحداث%20في%20وقت%20التنفيذ.htm)، كما تعلّمنا في الفصل الخامس.. بالرفاء والبنين!

**تغيير شكل عناصر القائمة ListBox:**

تمتلك القائمة خاصيّة "طور الرسم" DrawMode، التي تأخذ القيم التالية:

|  |  |
| --- | --- |
| Normal | يتمّ رسم الأداة بالشكل التقليديّ. |
| OwnerDrawnFixed | يمكنك أن تتدخّل في رسم الأداة، ولكن ارتفاع مساحة الرسم سيظلّ ثابتا. |
| OwnerDrawnVariable | يمكنك أن تتدخّل في رسم الأداة، مع قدرتك على تغيير ارتفاع كلّ عنصر. |

بعد تحديد قيمة هذه الخاصيّة، يمكنك استخدام الحدثين MeasureItem و DrawItem لتغيير أبعاد وشكل كلّ عنصر يتمّ إضافته للقائمة، بطريقة مشابهة للتي استخدمناها مع القوائم الرئيسيّة.. انظر للمثال التالي:



**Private Sub ListBox1\_MeasureItem(ByVal sender As Object, \_**

**ByVal e As System.Windows.Forms.MeasureItemEventArgs) \_**

**Handles ListBox1.MeasureItem**

**Dim itmSize As SizeF**

**Dim S As New SizeF(ListBox1.Width, 200)**

**itmSize = e.Graphics.MeasureString(items(e.Index).ToString, fnt, S)**

**e.ItemHeight = itmSize.Height**

**e.ItemWidth = itmSize.Width**

**End Sub**

**Private Sub ListBox1\_DrawItem(ByVal sender As Object, \_**

**ByVal e As System.Windows.Forms.DrawItemEventArgs) \_**

**Handles ListBox1.DrawItem**

**If e.Index = -1 Then Exit Sub**

**Dim txtBrush, bgBrush As SolidBrush**

**Dim TxtFnt As Font**

**If e.Index mod 2 = 0 Then**

**' تلوين العناصر الزوجيّة**

**txtBrush = New SolidBrush(Color.Blue)**

**bgBrush = New SolidBrush(Color.LightYellow)**

**Else**

**' تلوين العناصر الفرديّة**

**txtBrush = New SolidBrush(Color.Blue)**

**bgBrush = New SolidBrush(Color.Cyan)**

**End If**

**If e.State And DrawItemState.Selected Then**

**' استخدم اللون الأحمر والخطّ السميك مع العنصر المحدّد**

**txtBrush = New SolidBrush(Color.Red)**

**TxtFnt = New Font(fnt.Name, fnt.Size, FontStyle.Bold)**

**Else**

**TxtFnt = fnt**

**End If**

**e.Graphics.FillRectangle(bgBrush, e.Bounds)**

**e.Graphics.DrawRectangle(Pens.Black, e.Bounds)**

**Dim R As New RectangleF(e.Bounds.X, e.Bounds.Y, \_**

**e.Bounds.Width, e.Bounds.Height)**

**e.Graphics.DrawString(items(e.Index).ToString, TxtFnt, \_**

**txtBrush, R)**

**e.DrawFocusRectangle()**

**End Sub**

**تحسين الأدوات الموجودة**

أبسط طريقة لبناء الأدوات، هي وراثة أداة موجودة وتطويرها.

افترض أنّك تريد أن تصمم نموذجا يستقبل البيانات من المستخدم عن طريق مجموعة من مربّعات النصّ.. وافترض كذلك أنّك تريد أن نمنع المستخدم من كتابة أيّ حروف في مربّعات النصّ، بحيث يقتصر فقط على كتابة الأرقام.. في هذه الحالة ستضطرّ لتكرار كتابة الكود الذي يمنع المستخدم من كتابة الحروف في كلّ مربّع نصّ لديك.. الحلّ الأفضل هو أن تنشئ أداة جديدة ترث مربّع النصّ، وتحتوي على خاصيّة تحدّد نوعيّة البيانات التي يقبلها مربّع النصّ: نصوص، أرقام، أعداد عشريّة، لا يقبل علامات ترقيم... إلخ.. إنّ هذه الأداة ستكون مفيدة لك كثيرا في تطبيقات إدخال البيانات.

تعال نرى:

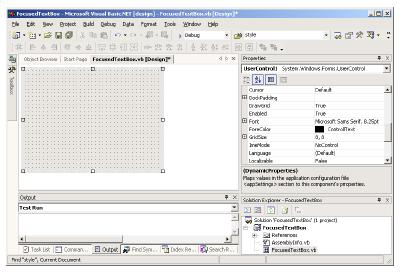
* [**تصميم الأدوات:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تحسين%20الأدوات%20الموجودة\0-%20تصميم%20الأدوات.htm)
* [**إنشاء مربّع النصّ الفعّال FocusedTextBox:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تحسين%20الأدوات%20الموجودة\1-%20إنشاء%20مربّع%20النصّ%20الفعّال.htm)
* [**كتابة بعض الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تحسين%20الأدوات%20الموجودة\2-%20كتابة%20بعض%20الخصائص.htm)
* [**تبويب الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\2-%20تحسين%20الأدوات%20الموجودة\3-%20تبويب%20الخصائص.htm)

**تصميم الأدوات:**

إن تصميم الأداة Control لن يكون جديدًا عليك كلّيّةً، فهي تتكوّن من شقّين:

1. واجهة الأداة:

ويمكنك تصميم هذه الواجهة كما تصمّم أيّ نموذج عاديّ، حيث تمنحك اللغة كائنا شبيها تماما بالنموذج، هو "أداة المستخدم" UserControl، التي يمكنك أن تضع عليها الأدوات أو الرسوم، بنفس الطريقة التي تفعل بها هذا مع النموذج.



1. كود الأداة:

لا بدّ أن تقوم أداتك بوظيفة أو أكثر، كما لا بدّ لها أن تمنح من سيستخدمها العديد من الخصائص والوسائل والأحداث التي سيبرمجها بها، تماما كما تفعل الأدوات التقليديّة.. لا مشكلة: ستطبّق نفس القواعد التي تعلّمتها في إنشاء الخلايا Classes، لإنشاء الخصائص والوسائل والأحداث اللازمة للأداة.. وحتمٌ عليّ أن أخبرك أنّ العديد من هذه الخصائص والوسائل والأحداث، تمنحها لك اللغة تلقائيّا دونَ أن تكتبَ أنت أيّ كود.

إذن يمكنك أن تقول باختصار: إنّ الأداة هي خليّة Class لها واجهة استخدام (نموذج).

ولكنّ هناك بعض الفروق الهامّة بينَ الأداة والنموذج، منها أنّ الأداة لا يمكن أن تستخدم بمفردها، بل يجب أن توضع على نموذج أولا ليتمّ عرضها من خلاله.. ومنها أنّ برمجة الأداة يجب أن تراعي حالتي التشغيل التاليتين للأداة:

1. حالة استخدام المستخدم للأداة في وقت التشغيل.
2. حالة استخدام المبرمج للأداة في وقت التصميم، بوضعها على نموذج وتعديل شكلها وخصائصها، فأنت المسئول أيضا عن سلوك الأداة في هذه الحالة، فرغم أنّه وقت تصميم للمبرمج، إلا إنّه وقت تشغيل للأداة!

من أجل هذا فإنّ أداة المستخدم UserControl تمنحك الخاصيّة "طور التصميم" DesignMode.. هذه الخاصيّة ترجع القيمة True إذا كانت الأداة في وقت التصميم، وترجع القيمة False إذا كانت في وقت التشغيل.. وبهذا تستطيع أن تحدّد ردّ الفعل المناسب الذي تستجيب به لأحداث المبرمج أو المستخدم.. لن نتطرّق لهذه النقطة مرّة أخرى في هذا الفصل، ولكن عليك أن تضعها في ذهنك، فستحتاجها حتما.

**إنشاء مربّع النصّ الفعّال FocusedTextBox:**

في هذا المثال سنبدأ بتطوير مربّع النصّ، بحيث يتغيّر لونه عندما يستقبل مؤشّر الكتابة، ويستعيد لونه الأصليّ عندما يفقد مؤشّر الكتابة.

ابدأ مشروعا جديدا، وفي مربّع حوار "مشروع جديد" اختر النوع "مكتبة أدوات ويندوز" Windows Control Library.. سمِّ المشروعَ FocusedTextBox واضغط موافق.. ستجد أنّ المشروع سيحتوي على أداة شبيهة بالنموذج، ولكن ليس لها أيّ حدود أو شريط عنوان.. هذه الأداة هي أداة المستخدم UserControl1.. هذه هي الأرضيّة التي ستصمّم عليها واجهة الأداة.

افتح نافذة الخصائص، وغيّر اسم أداة المستخدم إلى FocusedTextBox.. الآن علينا أن نجعل هذه الأداة ترث مربّع النصّ.. بسيطة.. اضغط بزر الفأرة الأيمن على أداة المستخدم ومن القائمة الموضعيّة اختَرْ "عرض الكود" View Code.. سيظهر لك الخليّة التي تمثّل الأداة، وستجد فيها الكود التالي:

**Public Class FocusedTextBox**

**Inherits System.Windows.Forms.UserControl**

**End Class**

تلاحظ أنّ سطر الوراثة ينصّ على أنّ هذه الأداة ترث خليّة أداة المستخدم UserControl.. لا نريد ذلك.. قم بتغيير هذا السطر لترث مربّع النصّ كالتالي:

**Inherits TextBox**

الآن احفظ المشروع، وعد إلى تصميم الأداة.. ستكتشف أنّ أداة المستخدم قد اختفت، وظهرت مكانها مساحة فارغة.

**ملحوظة:**

إذا كانت أداة المستخدم ما زالت معروضة، فأغلق واجهة التصميم وأعد فتحها مرّة أخرى، وستجد التغيير المشار إليه قد حدث.

ولاختبار هذه الأداة يجب عليك إضافتها إلى نموذج.. لهذا سنضيف مشروعا جديدا إلى التطبيق سنستخدمه لاختبار الأداة أثناء تصميمها.. اضغط القائمة "ملفّ" / "إضافة مشروع" / "مشروع جديد" File\Add Project\New Project.. وفي مربّع حوار "مشروع جديد" اختر النوع "تطبيق ويندوز" Windows Application وسمّ المشروع TestProject.

الآن يمكنك اختبار الأداة بوضع نسخة منها على النموذج Form1 في مشروع الاختبار.. ولفعل ذلك اتبع هذه الخطوات اليسيرة:

- حدد العنصر FocusedTextBox في متصفّح المشاريع Solution Explorer، ومن القائمة "بناء" Build اضغط الأمر Build FocusedTextBox، وذلك لبناء ملف Dll للأداة يمكن استخدامه.

- حدد العنصر TestProject في متصفّح المشاريع، واجعل هذا المشروع هو مشروع بدء التشغيل، بضغط الأمر Set As Startup Project في قائمة Project.

- افتح النموذج Form1.

- في صندوق الأدولت ToolBox ستجد اسم الأداة FocusedTextBox موجودا (آخر أيقونة).. أضف نسخة من هذه الأداة للنموذج.

- الآن حاول استخدام هذه الأداة.. ستكتشف أنّك تتعامل مع مربّع النصّ التقليديّ بدون أيّ اختلاف.

الآن تعال نضيف بعض الوظائف الجديدة لمربّع النصّ.. عُد إلى الأداة، وفي محرّر الكود Code Editor اضغط القائمة المنسدلة اليسرى، واختر منها العنصر "أحداث الخليّة الأساسيّة" Base Class Events.. في القائمة المنسدلة اليمنى اختر الحدث "دخول الأداة" Enter.. هذا هو الحدث الذي سنغيّر فيه لون مربّع النصّ، فهو ينطلق كلّما دخل مؤشّر الكتابة إلى مربّع النصّ.. اكتب في هذا الحدث الكود التالي:

**Private Sub FocusedTextBox\_Enter(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles MyBase.Enter**

**Me.BackColor = Color.Cyan**

**End Sub**

لاحظ أنّ كلمة Me هنا تشير إلى أداة المستخدم، التي هي في حالتنا هذه هي مربّع النصّ.

كذلك يجب برمجة حدث مغادرة مؤشّر الكتابة للأداة Leave، بحيث نعيد فيه لون مربّع النصّ إلى طبيعته.

**Private Sub FocusedTextBox\_Leave(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles MyBase.Leave**

**Me.BackColor = Color.White**

**End Sub**

**كتابة بعض الخصائص:**

ما رأيك الآن أن نضيف بعض الخصائص الجديدة لمربّع النصّ؟

سننشئ خاصيّة تسمّى "إلزامي" Mandatory، عند جعل قيمتها True سنغيّر لون مربع النصّ إذا كان فارغا، حتّى نشعر المستخدم بحتميّة إدخال قيمة فيه.. ولكي نسمح للمبرمج باختيار لون هذه الخاصيّة، فسننشئ خاصيّة أخرى تسمّى MandatoryColor.. وبالمرّة سننشئ خاصيّة تسمح بتحديد لون مربّع النصّ عند دخول مؤشّر الكتابة إليه، وسنسمّيها EnterFocusColor، وخاصيّة أخرى تسمح بتحديد لون مربّع النصّ عند خروج مؤشّر الكتابة منه، وسنسمّيها LeaveFocusColor.

تعال نكتب هذه الخصائص:

ابدأ بتعريف المتغيّرات الخاصّة التي ستحتفظ فيها بقيمة كلّ خاصيّة.. لقد جرى العرف البرمجيّ على أن تكون لهذه المتغيّرات نفس أسماء الخصائص ولكن مسبوقة بالعلامة "\_":

**Dim \_mandatory As Boolean**

**Dim \_enterFocusColor, \_leaveFocusColor As Color**

**Dim \_mandatoryColor As Color**

أمّا بالنسبة لكود الخصائص، فهو مباشر جدّا ومتماثل، لذلك فسنكتفي بذكر كود خاصيّتين فقط:

**Property Mandatory() As Boolean**

**Get**

**Mandatory = \_mandatory**

**End Get**

**Set(ByVal Value As Boolean)**

**\_mandatory = Value**

**End Set**

**End Property**

**Property EnterFocusColor() As Color**

**Get**

**EnterFocusColor = \_enterFocusColor**

**End Get**

**Set(ByVal Value As Color)**

**\_enterFocusColor = EnterFocusColor**

**End Set**

**End Property**

وأخيرا يجب تعديل الحدثين Enter و Leave ليتعاملا مع هذه الخصائص:

**Private Sub FocusedTextBox\_Enter(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles MyBase.Enter**

**Me.BackColor = \_enterFocusColor**

**End Sub**

**Private Sub FocusedTextBox\_Leave(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles MyBase.Leave**

**' إذا كانت خاصية الإلزام صحيحة، وكان مربّع النصّ فارغا أو به مسافات**

**' فغيّر لون مربّع النصّ إلى لون الإلزام**

**If \_mandatory AndAlso Trim(Me.Text) = "" Then**

**Me.BackColor = \_mandatoryColor**

**Else ' غير لون مربّع النصّ إلى لون المغادرة**

**Me.BackColor = \_leaveFocusColor**

**End If**

**End Sub**

والآن لتجربة هذه الإضافات، أعد بناء الأداة من جديد، وانتقل إلى نموذج الاختبار.. ضع عدّة نسخ من أداتنا عليه، وغيّر الخصائص الجديدة التي أنشأناها لكلّ منها باستخدام نافذة الخصائص.. ثمّ شغّل البرنامج واختبر هذه الخصائص.

رائع وسهل وممتع هو تطوير الأدوات.. أليس كذلك؟

**تبويب الخصائص:**

إذا كنت تعرض الخصائص في نافذة الخصائص مرتّبة على حسب النوع، فستجد أنّ الخصائص التي أنشأناها قد ظهرت كلّها تحت النوع Misc.. ولتغيير هذا الوضع الافتراضيّ، يمكن كتابة بعض السمات attributes أمام تعريف الخاصيّة.. فمثلا لجعل الخاصيّة تظهر تحت مقطع "المظهر" Appearance، استخدم السمة التالية:

**<Category("Appearance")>**

ويمكنك أن تضيف خصائصك لنوع جديد غير موجود سابقا في نافذة الخصائص، حيث سيتمّ إنشاؤه من أجلك، وذلك بمجرّد كتابة اسم النوع الجديد كالتالي:

**<Category("Mine")>**

ولاستخدام السمات، لا بدّ أن تستورد الخليّة System.ComponentModel، فكلّ السمات هي جزء منها:

**Imports System.ComponentModel**

وهناك سمة أخرى تهمّك، هي وصف الخاصيّة، ذلك الذي يظهر في الجزء السفليّ من نافذة الخصائص، ليشرح وظيفة الخاصيّة.. انظر الآن كيف سنعدّل تعريف خاصيّة Mandatory، لنضيف هاتين السمتين:

**<Description("توضّح إذا كان من الممكن ترك الأداة فارغة أم لا"), \_**

**Category("Appearance")> Property Mandatory() As Boolean**

أمّا أهمّ سمة، فهي سمة القيمة الافتراضيّة للخاصيّة DefaultValue، حيث يمكنك كتابة القيمة الافتراضيّة بين القوسين:

**<Description("توضّح إذا كان من الممكن ترك الأداة فارغة أم لا "), \_**

**Category("Appearance"), DefaultValue(False)> \_**

**Property Mandatory() As Boolean**

وهناك سمات لخليّة الأداة نفسها.. فمثلا، يمكنك أن تجعل الخاصيّة Mandatory هي الخاصيّة الافتراضيّة، عدّل تعريف الخليّة كالتالي:

**<DefaultProperty("Mandatory")> Public Class FocusedTextBox**

ويمكنك اختبار كلّ ذلك في تطبيق FocusedTextBox في مجلّد برامج هذا الفصل.

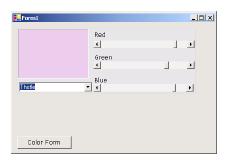
**إنشاء أدوات مركّبة**

في هذا المقطع لن نرث أداةً ونطوّرها، ولكنّنا سنستخدم مجموعة من الأدوات تقوم بوظيفة معيّنة، لإنشاء أداة جديدة مركّبة.

تعال نرى كيف ننشئ أداة تسمح للمستخدم باختيار اللون.. هذه الأداة ستتكوّن من الأجزاء التالية:

* ثلاثة منزلقات أفقيّة، تمثّل نسب الألوان الأساسيّة: الأحمر والأخضر والأزرق، بحيث يستطيع المستخدم تكوين اللون من نسبه.
* قائمة مركّبة منسدلة، يختار المستخدم منها اسم اللون مباشرة.
* لافتة سنجعل لون خلفيّتها هو اللون الذي سيركّبه المستخدم أو يختاره، حتّى يستطيع معاينة اللون.

افتح مشروع أداة ويندوز جديدا وسمّه ColorEdit، وصمّم أداة المستخدم لتبدو كما في الصورة.



سمّ المنزلقات الأفقيّة: RedBar، و GreenBar و BlueBar، واجعل قيمة خاصيّة MinimumValue لكلّ منها صفرا، وقيمة خاصيّة MaximumValue لكلّ منها 255.. وسمّ القائمة المركّبة NamedColors.. ولملء هذه القائمة بالألوان، استخدم الإجراء التالي:

**Private Sub AddNamedColors()**

**With ComboBox1.Items**

**.Add("Red")**

**.Add("Yello")**

**.Add("Blue")**

**' أضف ما تريد من أسماء الألوان بنفس الطريقة**

**' واجعل آخر عنصر في القائمة هو "غير معروف"**

**' لأنّ هناك ملايين الألوان التي لا تمتلك أسماء!**

**.Add("غير معروف")**

**End With**

**End Sub**

ويمكنك الاسترشاد بأسماء الألوان التي يحتوى عليها السجلّ Color، وستجدها في ملفّات إرشادات اللغة.

**ملحوظة:**

لو أردت أن تضيف أسماء الألوان التي يحتوى عليها السجلّ Color ـ وهي بالإنجليزيّة ـ للقائمة، فعليك أن تستخدم الكود التالي:

**Dim CNames() As String = [Enum].GetNames(GetType(KnownColor))**

**ComboBox1.Items.AddRange(CNames)**

**ComboBox1.Items.Add("غير معروف")**

مع ملاحظة أنّ المرقّم KnownColor لا يمدّك فقط بأسماء 128 لونا، بل يمدّك كذلك بأسماء ألوان الويندوز التي يسمح للمستخدم باختيارها من خصائص سطح المكتب DeskTop Properties، مثل ActiveBorder و ControlLight... إلخ.. وهي تقع في بداية المرقّم KnownColor (من 0 إلى 26).

وهذا يفيد في ترك الحرّية للمستخدم في تخصيص ألوان البرنامج كجزء من الويندوز ككلّ.

ولكي يتمّ تكوين الألوان في هذا الحالة، يجب استخدام الكود التالي:

**Clr = Color.FromKnownColor(ComboBox1.SelectedIndex)**

مع ملاحظة حتميّة استخدام معالجات الاستثناء اللازمة، لتوقّي أيّ خطأ، مثل عدم قابلية بعض الأدوات للون الشفاف.

وللتعامل مع أداتنا، سنجعل لها خاصيتي "اللون المحدّد" SelectedColor و"اللون المسمّى" NamedColor، ها هو ذا كود الخاصيّة الأولى:

**Property SelectedColor() As Color**

**Get**

**' تعيد هذه الخاصيّة لون اللافتة**

**SelectedColor = Label1.BackColor**

**End Get**

**Set(ByVal Value As Color)**

**' عند تغيير قيمة هذه الخاصيّة يجب أن تظهر مكوّنات اللون الجديد على المنزلقات**

**RedBar.Value = Value.R**

**GreenBar.Value = Value.G**

**BlueBar.Value = Value.B**

**End Set**

**End Property**

أما خاصيّة NamedColor فسنجعلها للقراءة فقط، وكودها كالتالي:

**ReadOnly Property NamedColor() As String**

**Get**

**' هذه الخاصيّة ترجع قيمة العنصر المحدّد حاليّا في القائمة**

**NamedColor = ComboBox1.SelectedItem**

**End Get**

**End Property**

وعندما يختار المستخدم اسم لون من القائمة، فعليك أن تكوّن اللون المقابل للاسم، باستخدام الوسيلة Color.FromName، إلا في حالة اختيار العنصر "غير معروف"!

**Private Sub ComboBox1\_SelectedIndexChanged(ByVal sender \_**

**As System.Object, ByVal e As System.EventArgs) \_**

**Handles ComboBox1.SelectedIndexChanged**

**Dim namedColor As Color**

**Dim colorName As String = ComboBox1.SelectedItem**

**If colorName <> "غير معروف" Then**

**' لن يفيد هذا الكود لو كتبت الأسماء باللغة العربيّة.. أترك لخيالك حلّ هذه المشكلة**

**namedColor = Color.FromName(colorName)**

**RedBar.Value = namedColor.R**

**GreenBar.Value = namedColor.G**

**BlueBar.Value = namedColor.B**

**End If**

**End Sub**

بقيت آخر خطوة، وهي كتابة الكود المستجيب لانزلاق المنزلقات الثلاثة:

**Private Sub ColorScroll(ByVal sender As System.Object, ByVal e \_**

**As System.Windows.Forms.ScrollEventArgs) \_**

**Handles RedBar.Scroll, GreenBar.Scroll, BlueBar.Scroll**

**Label1.BackColor = Color.FromARGB(HScrollBar1.Value, \_**

**HScrollBar2.Value, HScrollBar3.Value)**

**End Sub**

الآن لم يبقَ إلا اختبار أداة الألوان.. ولفعل ذلك، أضف مشروع اختبار للتطبيق (مشروع ويندوز عادي، سنستخدمه للاختبار)، واجعله مشروع بدء التشغيل.. قم ببناء الأداة وأضف نسخة منها لنموذج الاختبار.. شغّل التطبيق وانظر كيف يمكنك اختيار الألوان باستخدام المنزلقات أو القائمة.

أنهِ التطبيق وأضف زرّا للنموذج، واكتب فيه الكود التالي، لتغيير لون خلفيّة النموذج تبعا للون الأداة:

**Private Sub Button1\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Button1.Click**

**Me.BackColor = ColorEdit1.SelectedColor**

**End Sub**

**بناء أدوات بأشكال خاصّة**

في هذه المرّة لن نضع أيّ أداة على أداة المستخدم، بل سنقوم نحن بالرسم عليها، عن طريق إعادة كتابة وسيلة الرسم OnPaint لتستبدل Overrides وسيلة الرسم الخاصّة بالأداة.. تعال نرى ذلك عمليّا:

* [**اللافتة المجسّمة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\4-%20بناء%20أدوات%20بأشكال%20خاصّة\0-%20اللافتة%20المجسّمة.htm)
* [**تصميم أدوات بأشكال غير تقليديّة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\4-%20بناء%20أدوات%20بأشكال%20خاصّة\6-%20تصميم%20أدوات%20بأشكال%20غير%20تقليديّة.htm)

**اللافتة المجسّمة**

سنقوم هنا بإنشاء لافتة تعرض نصّا مجسّما.. وأنصحك هنا بمراجعة فصل التلوين، فقد قمنا فيه بإنشاء مشروع [Text Effects](file:///C:\\Users\\Administrator\\Desktop\\vb.net\\12-%20الرسم%20والتلوين\\3-%20رسم%20النصوص.htm) الذي يرسم نصّا مجسّما على النموذج.. سنقوم هنا بنفس الأمر، لكنّنا سنرسم في هذه المرّة على أداة المستخدم وليس على نموذج.

ابدأ مشروع أداة ويندوز جديدا وسمّه Label3D.. سمِّ أداة المستخدم أيضا Label3D.



هيّا بنا نرى كيف نصمّم هذه الأداة الشيّقة:

* [**إضافة الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\1-%20إضافة%20الخصائص.htm)
* [**رسم النصّ المجسّم:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\2-%20رسم%20النصّ%20المجسّم.htm)
* [**جعل خلفيّة الأداة شفّافة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\3-%20جعل%20خلفيّة%20الأداة%20شفّافة.htm)
* [**وضع القيم المبدئيّة للأداة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\4-%20وضع%20القيم%20المبدئيّة%20للأداة.htm)
* [**أحداث تغيّر الخصائص Changed:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\5-%20أحداث%20تغيّر%20الخصائص.htm)
* [**خصائص معامل حدث التغيّر:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\6-%20خصائص%20معامل%20الحدث.htm)
* [**استخدام اللافتة المجسمة في مشاريع أخرى:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\13-%20بناء%20الأدوات%20الخاصّة\0-%20اللافتة%20المجسّمة\7-%20استخدام%20اللافتة%20المجسمة%20في%20مشاريع%20أخرى.htm)

**إضافة الخصائص:**

إنّ من الخصائص التي سنضيفها لهذه اللافتة، خاصيّة المحاذاة، وهي تأخذ عدّة قيم، سنضعها في المرقّم Align.. ابدأ بتعريف هذا المرقّم في بداية خليّة الأداة:

**Public Enum Align**

**TopLeft**

**TopMiddle**

**TopRight**

**CenterLeft**

**CenterMiddle**

**CenterRight**

**BottomLeft**

**BottomMiddle**

**BottomRight**

**End Enum**

أمّا النصّ الذي ستعرضه اللافتة، فسنسمح للمبرمج بتحديد التأثير الذي سيعرض به، وذلك من خلال خاصيّة Effect3D التي تأخذ إحدى قيم المرقّم التالي:

**Public Enum Effect3D**

**None ' بدون تأثير**

**Raised ' مرتفع**

**RaisedHeavy ' مرتفع سميك**

**Carved ' محفور**

**CarvedHeavy ' محفور سميك**

**End Enum**

بعد ذلك سنعرّف متغيّرين خاصّين، نحفظ فيهما قيمتي الخاصيتين.. لا بدّ بالطبع أنّ يكون كل متغيّر من نوع المرقّم الذي سيحفظ قيمته:

**Private Shared mAlignment As Align**

**Private Shared mEffect As Effect3D**

الآن سنكتب خاصيتي المحاذاة والتأثير المجسّم:

**Public Property Alignment() As Align**

**Get**

**Alignment = mAlignment**

**End Get**

**Set(ByVal Value As Align)**

**mAlignment = Value**

**Invalidate()**

**End Set**

**End Property**

**Public Property Effect() As Effect3D**

**Get**

**Effect = mEffect**

**End Get**

**Set(ByVal Value As Effect3D)**

**mEffect = Value**

**Invalidate()**

**End Set**

**End Property**

لاحظ استخدامنا للوسيلة Invalidate، حتّى تعيد رسم النصّ من جديد على سطح اللافتة، وبهذا يتمّ تطبيق القيمة الجديدة للخاصيّة على النصّ.

أضف مشروع اختبار للتطبيق، وحاول أن تجرّب الأداة.. ضع نسخة منها على النموذج، وانظر لخصائصها في نافذة الخصائص.. ستكتشف أنّ النافذة لا تحتوي فقط على الخاصيتين اللتين عرّفناهما، بل إنّ هناك عددا هائلا من الخصائص الجاهزة التي تتعامل مع شكل وموقع اللافتة وألوانها وخطوطها.. هذه الخصائص لم تكتبها أنت، ولكنّ أداة المستخدم منحتها لك لحسن الحظّ.

ولكن رغم هذا ما زلنا بحاجة لخاصيّة Text3D التي سيكتب فيها المستخدم النصّ الذي يريد عرضه في اللافتة.. تعال نكتب هذه الخاصيّة:

**Private mText As String**

**Public Property Text3D () As String**

**Get**

**Text3D = mText**

**End Get**

**Set(ByVal Value As String)**

**mText = Value**

**Invalidate()**

**End Set**

**End Property**

**رسم النصّ المجسّم:**

الآن حان وقت الجدّ.. نحتاج لكتابة الوسيلة OnPaint.. هذه الوسيلة ستستدعيها الأداة تلقائيّا كلّما احتاجت لأن يعاد رسمها وعند استخدام الوسيلة Invalidate.. ابدأ بوضع الجملة التالية في بداية الملفّ:

**Imports System.Drawing.Drawing2D**

والآن تعال نرى كيف سنرسم النصّ بتأثيراته ومحاذاته:

**Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)**

**Dim lblFont As Font = Me.Font**

**Dim lblBrush As New SolidBrush(Color.Red)**

**Dim X, Y As Integer**

**' يجب معرفة أبعاد النصّ حتّى نستطيع محاذاته**

**Dim TextSize As SizeF**

**SizeF = e.Graphics.MeasureString(mText, lblFont)**

**Select Case Me.mAlignment**

**Case Align.BottomLeft ' رسم النصّ في الركن السفليّ الأيسر**

**X = 2 ' نقطتان لمراعاة حافة الأداة**

**Y = Me.Height - TextSize.Height**

**Case Align.BottomMiddle ' رسم النصّ في أسفل المنتصف**

**X = CInt((Me.Width - TextSize.Width) / 2)**

**Y = Me.Height - TextSize.Height**

**Case Align.BottomRight ' رسم النصّ في الركن السفليّ الأيمن**

**X = Me.Width - TextSize.Width - 2**

**Y = Me.Height - TextSize.Height**

**Case Align.CenterLeft ' رسم النصّ في منتصف اليسار**

**X = 2**

**Y = (Me.Height - TextSize.Height) / 2**

**Case Align.CenterMiddle ' رسم النصّ في مركز اللافتة**

**X = (Me.Width - TextSize.Width) / 2**

**Y = (Me.Height - TextSize.Height) / 2**

**Case Align.CenterRight ' رسم النصّ في منتصف اليمين**

**X = Me.Width - TextSize.Width - 2**

**Y = (Me.Height - TextSize.Height) / 2**

**Case Align.TopLeft ' رسم النصّ في أعلى اليسار**

**X = 2**

**Y = 2**

**Case Align.TopMiddle ' رسم النصّ في أعلى المنتصف**

**X = (Me.Width - TextSize.Width) / 2**

**Y = 2**

**Case Align.TopRight ' رسم النصّ في أعلى اليمين**

**X = Me.Width - TextSize.Width - 2**

**Y = 2**

**End Select**

**' المتغيّرين التاليين سنضع فيهما مقدار إزاحة النصّ رأسيّا أو أفقيّا حتّى يبدو مجسّما**

**Dim dispX, dispY As Integer**

**Select Case mEffect**

**Case Effect3D.None**

**dispX = 0 : dispY = 0**

**Case Effect3D.Raised**

**dispX = 1 : dispY = 1**

**Case Effect3D.RaisedHeavy**

**dispX = 2 : dispY = 2**

**Case Effect3D.Carved**

**dispX = -1 : dispY = -1**

**Case Effect3D.CarvedHeavy**

**dispX = -2 : dispY = -2**

**End Select**

**' رسم النصّ بلون أبيض**

**lblBrush.Color = Color.White**

**e.Graphics.DrawString(mText, lblFont, lblBrush, X, Y)**

**' رسم النصّ مرّة أخرى بلون أسود، مع إزاحته قليلا رأسيا وأفقيّا ليبدو التأثير المجسّم**

**lblBrush.Color = Me.ForeColor**

**e.Graphics.DrawString(mText, lblFont, lblBrush, X + dispX, Y + dispY)**

**End Sub**

الآن حاول تجربة ما فعلناه.. ستكتشف كم هو رائع!

حاول كذلك أن ترى أحداث اللافتة المجسّمة.. ستكتشف أنّ لها عددا هائلا من الأحداث الجاهزة التي لم تكتبها أنت.. منتهى الراحة!

**جعل خلفيّة الأداة شفّافة:**

ما رأيكُ أن نُضيفَ إمكانيّة جديدةً للافتة المجسّمة، بحيثُ يمكنك أن تجعل خلفيّتها شفّافة؟.. إنّ ذلك سيضمنُ مظهرا أفضلَ للنموذج عند وضع صورة في خلفيّته، حيثُ لن تحجب اللافتة ذلك الجزء من الصورة الذي يقع وراءها.

عامّةً الأمرُ في غاية البساطة: اكتب الجملة التالية في الحدث New للأداة:

**Setstyle(ControlStyles.SupportsTransparentBackColor, True)**

إنّ هذه الجملة ستسمح لك بجعل خلفيّة النموذج أو الأداة شفّافة.. ويمكنك أن تجرّب ذلك.. قم ببناء الأداة من جديد، ثمّ انتقل لنموذج الاختبار.. حدّد اللافتة المجسّمة، وفي نافذة الخصائص حدّد خاصيّة "لون الخلفيّة" BackColor، وفي خانة القيمة اضغط زرّ الإسدال.. في مربّع الألوان اضغط شريط Web.. ستجد أنّ أول قيمة في هذا الشريط هي "شفّاف" Transparent.. اختر هذه القيمة.. وحتّى تتأكّد بالفعلِ أنّ خلفيّة اللافتة شفّافة، غيّر لون النموذج، أو أضف لخلفيّته صورة (عن طريق خاصيّة BackgroundImage).

رأيت؟.. غاية البساطة!

**وضع القيم المبدئيّة للأداة:**

لتحديد القيم المبدئيّة للأداة، تلك التي ستأخذها خصائصها عند وضعها على النموذج لأوّل مرّة، استخدم الحدث New.. وستجد اسمه في القائمة المنسدلة اليمنى في محرّر الكود.. اضغط هذا الاسم للوصول إليه، واكتب فيه ما يلي:

**Public Sub New()**

**' \*\*\*\*\*\*\*\*\*\*\*\*\*لا تغيّر المقطع التالي ولا تحذفه \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**MyBase.New()**

**' This call is required by the Windows Form Designer.**

**InitializeComponent()**

**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**mText = "Label3D"**

**mAlignment = Align.CenterMiddle**

**mEffect = Effect3D.Raised**

**' لاحظ أنّ بإمكانك تحديد الخطّ الافتراضي من نافذة خصائص أداة المستخدم**

**Me.Font = New Font(Me.Font.Name, 14, FontStyle.Bold)**

**' الجملة التالية تجبر الأداة على استدعاء حدث الرسم عندما يتغيّر حجمها**

**SetStyle(ControlStyles.ResizeRedraw, "True")**

**End Sub**

ومن الجدير ذكرا، هو أنّ أداة المستخدم User Control تمتلك خاصيّة جاهزة تريحك من استدعاء الوسيلة SetStyle.. هذه الخاصيّة هي ResizeRedraw، والتي عند جعل قيمتها True تجبر الأداة على استدعاء حدث الرسم عندما يتغيّر حجمها.. ويمكنك استبدال الجملة التالية بجملة SetStyle في كود الإجراء New:

**Me.ResizeRedraw = True**

**أحداث تغيّر الخصائص Changed:**

ستجد أنّ الأداة تمنحك العديد من الأحداث الجاهزة، معظمها يحدث عند تغيّر خاصيّة من الخصائص، مثل تغيّر الخط FontChanged.. على نفس هذا المنوال لا بدّ أن تكتب أحداث تغيّر الخصائص الجديدة التي كتبتها للأداة.

ابدأ بتعريف الأحداث كالتالي:

**Public Event AlignmentChanged(ByVal sender As Object, ByVal e \_**

**As EventArgs)**

**Public Event EffectChanged(ByVal sender As Object, ByVal e \_**

**As EventArgs)**

**Public Event TextChanged(ByVal sender As Object, ByVal e \_**

**As EventArgs)**

ثم أطلق كل حدث منها في مقطع تغيّر الخاصيّة المناظرة له.. اكتب السطر المناسب من السطور الثلاثة التالية بعد السطر الذي يستدعي الوسيلة Invalidate في الخاصيّة المناظرة لكلّ حدث:

**RaiseEvent AlignmentChanged(Me, New EventArgs())**

**RaiseEvent EffectChanged(Me, New EventArgs())**

**RaiseEvent Text3DChanged(Me, New EventArgs())**

**خصائص معامل الحدث:**

أعرف أنّك ستتساءل عن جدوى إرسال المعامل الثاني للحدث ما دام لا يحتوي على أي معلومات مفيدة؟

عامّة يمكنك تغيير ذلك.. فلو أردت إرسال أيّ معلومة جديدة في المعامل الثاني، فعليك أن تنشئ خليّة جديدة ترث الخليّة EventArgs، حيث تضيف لها كلّ ما تريد من الخصائص، لتحمل المعلومات التي تريد تمريرها للمبرمج عند انطلاق الحدث.. افترض مثلا أنّك تريد أن تخبر المبرمج بالقيمة القديمة والقيمة الجديدة للخاصيّة، مع منحه القدرة على إلغاء التغيير قبل حدوثه.. في هذه الحالة عرّف الخليّة التالية:

**Public Class PropertyChangedEvent**

**Inherits EventArgs**

**Public Shared ReadOnly OldValue As Object**

**Public Shared ReadOnly NewValue As Object**

**Public Shared Cancel As Boolean**

**End Class**

الآن يجب أن نعدّل تعريف الأحداث ليصبح المعامل الثاني من النوع PropertyChangedEvent.. تعال نأخذ الحدث AlignmentChanged كمثال، وعليك فعل المثل مع الحدثين الآخرين:

**Public Event AlignmentChanged(ByVal sender As Object, ByRef e \_**

**As PropertyChangedEvent)**

ويجب إعادة كتابة خاصيّة المحاذاة لتصير كالتالي:

**Public Property Alignment() As Align**

**Get**

**Alignment = mAlignment**

**End Get**

**Set(ByVal Value As Align)**

**Dim e As New PropertyChangedEvent**

**e.OldValue = mAlignment**

**e.NewValue = Value**

**RaiseEvent AlignmentChanged(Me, e)**

**If e.Cancel = False Then**

**mAlignment = Value**

**Invalidate()**

**End If**

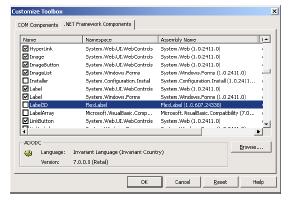
**End Set**

**End Property**

**استخدام اللافتة المجسمة في مشاريع أخرى:**

لكي تستخدم اللافتة المجسّمة في مشروع آخر اتبع هذه الخطوات:

* ابدأ مشروعا جديدا، واعرض نموذجه.
* اضغط صندوق الأدوات بزر الفأرة الأيمن، ومن القائمة الموضعيّة اضغط الأمر "تخصيص صندوق الأدوات" Customize Toolbox.



* في مربّع الحوار الذي سيظهر لك، اضغط شريط .NET Framework Components.
* اضغط زرّ التصفّح Browse، وفي مربّع حوار "فتح ملفّ" افتح الملفّ Label3D.dll (ستجده في المجلّد Bin في مجلّد مشروع اللافتة).
* ستجد أنّ اللافتة المجسّمة قد أضيفت لقائمة .NET Framework components.. تأكّد أن مربّع الاختيار أمام اسمها عليه علامة (✓)، واضغط OK.
* الآن ستظهر أيقونة اللافتة المجسّمة في صندوق الأدوات، حيث يمكنك استخدامها بالطريقة التقليديّة.

**تصميم أدوات بأشكال غير تقليديّة:**

رأيت في الفصل الخامس كيف أنشأنا نموذجا بيضاويّا.. لا يوجد ما يمنع من تنفيذ نفس الأمر مع أداة المستخدم لتبدو بيضاويّة أو تتخذ أيّ شكل تريده.

إنّنا سنقوم هنا بالمثل، ولكن ليس باستخدام خاصيّة TransparencyKey ، ولكن عن طريق استخدام خاصيّة "منطقة الرسم" Region الخاصّة بأداة المستخدم، والتي عن طريقها نستطيع تحديد منطقة الأداة.

**ملحوظة:**

يمتلك النموذج أيضا خاصيّة Region، حيث يمكنك استخدامها لتغيير شكل النموذج، بنفس الطريقة التي سنشرحها مع أداة المستخدم هنا.. ولكنّ هذه الخاصيّة لا توجد في الوضع التلقائيّ في قائمة أعضاء النموذج (التي تظهر أثناء كتابة الكود).. ويمكنك أن تعرض كلّ خصائص ووسائل النموذج في قائمة الأعضاء، باتّباع التالي: افتح القائمة Tools واضغط الأمر Options.. في النافذة التي ستظهر لك، اضغط العنصر Text Editor، ومن عناصره الفرعيّة اختر Basic ومن عناصره الفرعيّة اختر General.. على اليسار ستجد مربّعات اختيار.. أزل العلامة من الاختيار التالي "إخفاء الأعضاء المتقدّمة" Hide Advanced Members.. إنّ هذا سيؤدّي لظهور كلّ أعضاء النموذج والأدوات في قائمة الأعضاء، ومن بينها خاصيّة Region.

انظر للمثال التالي:

**Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)**

**Dim G As Graphics = Me.CreateGraphics**

**Dim roundPath As New GraphicsPath()**

**Dim R As New Rectangle(0, 0, Me.Width, Me.Height)**

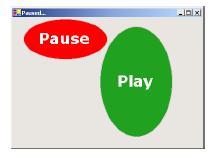
**roundPath.AddEllipse(R)**

**Me.Region = New Region(roundPath)**

**' الخطوة التالية اختياريّة، وهي رسم حافة رمادية حول منطقة الأداة**

**Me.CreateGraphics.DrawEllipse(New Pen(Color.DarkGray, 3), R)**

**End Sub**



وللتدريب، لديك في مجلّد برامج هذا الفصل تطبيق RoundButton.. وقتا ممتعا.

**الأدوات من النوع :ActiveX**

لعلّك لاحظت أنّ الأدوات الجديدة التي تنشئها بـ VB.Net لها الامتداد .dll .. لم يكن هذا هو الحال في النسخ السابقة، حيث كانت التقنية المسمّاة COM هي المستخدمة، وكانت الأدوات تسمّى ActiveX ولها الامتداد .ocx .

ولكن.. هل معنى هذا أن تقنية COM قد أقصيت نهائيّا؟

لا.. إنّ هذا مستحيل، نظرا لأنّ هناك ملايين الأدوات التي تمّ إنتاجها من تلك النوعيّة، ممّا يعني أنّ إهمالها مرّة واحدةً سيكون خسارة فادحةً للمبرمجين.

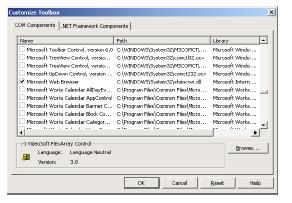
عامّة، كانت (ميكروسوفت) تطلق على تقنية .Net في بدايتها COM2، حيث قامت بإعادة بناء تقنية COM من جديد، لحلّ المشاكل التي كانت تلازمها.

الخلاصة: ما زال باستطاعتك استخدام أدوات COM في VB.Net.

فمثلا يمكنك استخدام أداة تصفّح الإنترنت WebBrowser Control، التي تعتبر بمثابة متصفّح الويندوز Internet Explorer موضوعا في أداة، وبواستطها يمكنك فتح صفحات HTML (ويمكنك استغلال ذلك في عرض ملفّات المساعدة)، كما يمكنك الاتصال بمواقع الإنترنت.



ولإضافة هذه الأداة لتطبيقك، اضغط صندوق الأدوات بزر الفأرة الأيمن، ومن القائمة الموضعيّة اضغط Customize Toolbox.. وفي مربّع الحوار اضغط شريط COM Components.. الآن يمكنك أن تضيف الأداة التي ترغب فيها.. ابحث عن الأداة Microsof Web Browser، فإذا لم تكن موجودة، فاضغط زر Browse، وافتح مجلّد System32 في مجلّد الويندوز، ومنه اختر الملفّ shdocvw.dll.. هذا الملفّ سيكون موجودا بالتأكيد لو كانت لغة VB6 موجودةً على جهازك، فإذا لم تكن موجودة، فلا أضمن إن كنت ستجد هذا الملفّ أم لا.. فإذا وجدته فاختره واضغط موافق.



ستجد اسم الأداة قد أضيف للقائمة وبجوارها علامة (✓).. اضغط زر Ok لإغلاق مربّع الحوار.. الآن ستجد أيقونة الأداة Explorer مضافةً كآخر عنصر في صندوق الأدوات.. اضغطها مرّتين لتضيف منها نسخةً للنموذج.. ستلاحظ أنّ اللغة ستستهلك بعض الوقت قبلَ أن تضع هذه النسخة على النموذج.. هذا الوقت تحتاجه اللغة لتكوين بعض الملفّات الوسيطة التي تستطيع من خلالها أن تتحاور مع الكائنات التي تستخدم تقنية Com.

ولفتح ملفّ HTML باستخدام هذه الأداة، استخدم الوسيلة التالية:

**AxWebBrowser1.Navigate2(FileName)**

حيث FileName هو اسم صفحة HTML التي تريد فتحها.

ولقد رأيت في فصل [الرسم والتلوين](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\6-%20رسم%20الدوالّ.htm)، كيف استخدمنا الأداة Script control في رسم أيّ دالّة يكتبها المستخدم.

إنّ عالم أدوات ActiveX رحيب، ويمتلئ بالإمكانيّات الرائعة.. وإن كان استخدامك لها يستلزم أن تحصل على ملفّات الإرشادات التي تشرح كيفيّة استخدامها، فهذه المعلومات لن تجدها بالطبع في إرشادات VB.

آخر ما أحبّ أن ألفت نظرك إليه، هو أنّ VB.Net يسمح لك بتحويل الخلايا والأدوات التي تنشئها إلى تقنية COM، لتتوافق مع اللغات التي تتعامل مع هذه التقنية.. للأسف.. هذا موضوع معقّد ويحتاج لمرجع أكثر تقدّما.