**الفصل السادس**

**أدوات الويندوز الأساسيّة**

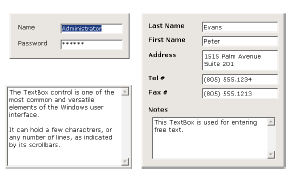
**Basic Windows Controls**

في هذا الفصل، سنناقش أهمّ الأدواتِ التي تحتاجها لبناء تطبيقك، وسنركّز في كل أداة، على شرح أهم الخصائص والوسائل والأحداث، الخاصّة بها دون غيرها من الأدوات، نظرًا لأنّ باقي هذه الأشياء إمّا نادرة الاستخدام، وإمّا مشتركة لجميع الأدوات، وقد شرحنا هذه الأخيرة بالفعل في الفصل الأوّل.

* [**مربّع النصّ TextBox:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ.htm)
* [**التعامل مع الخطوط:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20التعامل%20مع%20الخطوط.htm)
* [**القوائم Lists:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\2-%20القوائم.htm)
* [**الفاصل Splitter:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\3-%20الفاصل%20Splitter.htm)
* [**أداة تلميحات الشاشة ToolTip:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20أداة%20تلميحات%20الشاشة.htm)
* [**المنزلق ScrollBar**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\5-%20المنزلق.htm)
* [**شريط القيمTrackBar :**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\6-%20شريط%20القيم.htm)
* [**السحبُ والإسقاط Drag And Drop:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط.htm)

**مربّع النصّ TextBox**

وهو غنيّ عن التعريف، فلقد رأينا مرارا، كيف يمكن استخدامه لكتابة وتحرير النصوص.



* [**أهم الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\2-%20أهم%20الخصائص.htm)
* [**تحديد النصوص في مربّع النصّ:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\1-%20تحديد%20النصوص.htm)
* [**التراجع عن عمليات التحرير:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\11-%20التراجع%20عن%20عمليات%20التحرير.htm)
* [**مشروع محرر النصوص TextPad Project:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\3-%20مشروع%20محرر%20النصوص.htm)
* [**التربّص بضربات الأزرار Capturing Keystrokes:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\4-%20التربّص%20بضربات%20الأزرار.htm)
* [**التربّص بحروف الوظائف Capturing Function Keys:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\5-%20التربّص%20بحروف%20الوظائف.htm)
* [**تغيير لغة لوحة المفاتيح برمجيّا:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\1-%20مربّع%20النصّ\6-%20تغيير%20لغة%20لوحة%20المفاتيح.htm)

**أهم الخصائص Basic Properties:**

**محاذاة النصّ TextAlign:**

يمكن أن تكون واحدة من ثلاث قيم: اليسار Left، واليمين Right، والوسط Center.

**متعدّد الأسطر MultiLine:**

بجعل هذه الخاصيّة صوابا، يمكن للمستخدم كتابة أسطر عديدة في مربع النصّ، وذلك بضغط زر الإدخال Enter للانتقال لسطر جديد.

وفي الوضع الافتراضيّ، تكون قيمة هذه الخاصية خطأ، ممّا يعني أنّ مربع النص قابل لكتابة سطر واحد فقط.

**المنزلقان ScrollBars:**

تسمح لك هذه الخاصيّة باختيار المنزلق الذي تودّ ظهوره (الأفقي أو الرأسي أو كليهما أو ولا واحد منهما)، في حالة تجاوز النص لحدود مربع النص.

**ملاحظتان:**

* لا يظهر أي من المنزلقين إلا إذا كان مربع النص متعدد الأسطر.
* في حالة وجود المنزلق الأفقي، لا يلتفّ السطر الطويل إلى السطر التالي.

**التفاف الأسطر WordWrap:**

لو كانت هذه الخاصية "صوابا" (وهي القيمة الافتراضيّة بالفعل)، فإنّ السطر الذي يتجاوز طوله عرض مربع النصّ، سيلتفّ إلى السطر التالي، بدلا من الاحتياج إلى منزلق أفقيّ لقراءته.

**ملاحظة:**

ليس لهذه الخاصيّة أي تأثير على النص الأصليّ، فالسطر الذي تراه ملتفا ليتم إكماله في سطر تالٍ ما زال سطرا واحدا.. إنّها مجرّد طريقة للعرض.

**يستقبل زر الإدخال AcceptsReturn ويستقبل زر الانتقال AcceptsTab:**

تعرف أنّ بإمكانك جعل أحد الأزرار هو زر الموافقة للنموذج AcceptButton.. في هذه الحالة يتم ضغط الزر عند ضغط زر الإدخال Enter من لوحة المفاتيح.

ولكن ماذا سيحدث لو كان مؤشّر الكتابة موجودا الآن في أحد مربعات النصوص؟

هذا هو ما توضّحه خاصيّة AcceptsReturn، فلو كانت قيمتها صوابا (وهي القيمة الافتراضيّة)، يستقبل مربع النص ضغطة زر الإدخال، ولو كانت خطأ، يستقبل الزر الافتراضي هذه الضغطة.

ولكن كيف يمكن الكتابة في سطر جديد في الحالة الأخيرة؟

يمكن ذلك عن طريق ضغط Ctrl+Enter.

غنيّ عن الذكر، أنّ هذه الخاصيّة تصبح عديمةَ الجدوى، إذا لم يحتوِ النموذج على زر موافقة افتراضيّ!

موقفٌ مشابهٌ ذلك الذي يتعلّق بزر الانتقال Tab، فهو ذو وظيفتين: الانتقال من أداة لأخرى على النموذج، أو كتابة حرف جدولة يعادل 4 مسافات في مربع النص.

لهذا يمكنك وضع خاصيّة AcceptsTab بصواب للسماح بكتابة حرف الجدولة في مربع النص، فإذا أردت الانتقال إلى أداة أخرى في هذه الحالة، يمكن استخدام Ctrl+Tab (مع مربع النص الحاليّ فقط وليس باقي الأدوات).

كما يمكن جعل هذه الخاصيّة خطأ (وهي القيمة الافتراضيّة)، لجعل المستخدم ينتقل للأداة التالية عند ضغط زر الانتقال Tab.

**أقصى طول MaxLength:**

القيمة الافتراضيّة لهذه الخاصيّة هي 32,767، وهي أقصى طول كان مربع النص يحتويه في VB6.. ولكن لو جعلت قيمة هذه الخاصيّة صفرا، يمكنك أن تكتب حوالي 2 مليار حرف ( 2 جيجا) في مربع النصّ!

ويمكن وضع هذه القيمة بأيّ رقم تريده (5 مثلا)، لمنع المستخدم من كتابة أي نص أطول من هذا.

**تمّ تعديله Modified:**

عندما يجري المستخدم أيّ تغيير على محتويات مربّع النصّ، بالكتابة أو الحذف، فإنّ قيمة هذه الخاصيّة تتغيّر تلقائيّا لتصبح True.. أمّا لو كان التغيير في محتويات مربّع النصّ قد قمت به أنت من خلال الكود، فإنّ قيمة هذه الخاصيّة تظلّ كما هي.

ويمكنك استخدام هذه الخاصيّة للتأكّد من حدوث تغيّرا على النصّ، حتّى تسأل المستخدم إذا كان يريد حفظ هذه التغييرات أم لا.. وهنا يجب أن تعيد قيمة هذه الخاصيّة إلى False بنفسك، كلّما قام المستخدم بحفظ التغييرات.

**الأسطر Lines:**

يمكنك بهذه الخاصيّة قراءة أي سطر في مربع النصّ، بمجرّد تحديد رقمه.. فمثلا: يمكن قراءة السطر الأوّل باستخدام تعبير Lines(0)، والسطر الثاني باستخدام تعبير Lines(1).. وهكذا.

ولو فتحت هذه الخاصيّة في نافذة الخصائص، فستجد مكتوبا في خانة القيمة التعبير "مصفوفة نصوص" String[] Array، وبجواره زر انتقال.. اضغط هذا الزر، ليظهر لك مربع حوار "محرر مجموعة النصوص"، حيث يمكنك كتابة السطور التي تريد أن تظهر في مربع النص، مع ملاحظة أنّها لن تظهر فيه بطريقة صحيحة، إلا إذا كانت قيمة خاصية "متعدد الأسطر" صوابا.

ونظرا لأن Lines مصفوفة نصوص، يمكن استخدام وسيلة "اقرإ الحدّ الأعلى" GetUpperBound معها تماما كأيّ مصفوفة، لمعرفة رقم آخر خانة في المصفوفة، كما يمكن استخدام الوسيلة "طول" Length لمعرفة عدد هذه الخانات.

فمثلا، يمكنك أن تتعامل مع سطور مربع النص واحدا بعد الآخر، بجملة كالتالية:

**Dim L As Integer**

**For L = 0 To TextBox1.Lines.GetUpperBound(0)– 1**

**' أي كود للتعامل مع السطر الحالي**

**MsgBox (TextBox1.Lines(L))**

**Next**

ونظرا لأنّ كل خانة في مصفوفة السطور تحتوي على نصّ، لذلك يمكن استخدام وسائل النصوص لكل سطر.. فمثلا، لمعرفة عدد حروف السطر الثاني، استخدم كودا كالتالي:

**Dim LineLength As String**

**LineLength = TextBox1.Lines(1).Lenght**

لا يمكنك تغيير أي سطر في مربع النص، بتغيير قيمة أي خانة في مصفوفة السطور.. فالجملة التالية لن تؤثّر في مربّع النصّ على الإطلاق:

**TextBox1.Lines(1) = "سطر آخر"**

إذن ما الحل؟

بسيطة: عرّف مصفوفة نصوص، وانقل إليها سطور النصّ، وغيّر فيها كما تريد، ثمّ أعد وضعها في خاصيّة الأسطر كالتالي:

**Dim L() As String**

**L = TextBox1.Lines**

**L(1) = "سطر آخر"**

**TextBox1.Lines = L**

**نص Text:**

تحمل هذه الخاصيّة النصوص التي تكتب في مربع النص.

وهذه الخاصيّة من نفس نوع المتغيرات النصّيّة String، لهذا يمكن معاملتها كما تعامل المتغيرات من هذا النوع: ترسلها كمعامل للدوال التي تستقبل نصوصا، تضيف محتوياتها إلى محتويات نص آخر، أو تستخدم معها وسائل خليّة النصوص String Class.

مثلا: هذا التعبير يحسب طول النص الموجود في مربع النص:

**Dim StrLen As Integer**

**StrLen = TextBox1.Text.Length**

والذي ما زال يمكن حسابه بالطريقة القديمة:

**StrLen = Len(TextBox1.Text)**

أو حسابه مباشرة بهذه الطريقة الثالثة:

**StrLen = TextBox1.TextLength**

ويمكنك إفراغ مربع النص ومسح كل محتوياته بوضع نص فارغ فيه:

**TextBox1.Text = ""**

أو باستخدام الوسيلة "مسح" Clear:

**TextBox1.Clear**

**إضافة نص AppendText:**

تمكّنك هذه الوسيلة من إضافة أي نص على نهاية ذلك الموجود في مربع النص، وهي أسرع بمراحل من استخدام الطريقة التقليديّة:

**TextBox1.Text = TextBox1.Text & "نص جديد"**

الآن يمكنك استخدام الطريقة التالية:

**TextBox1.AppendText("نص جديد")**

حيث ستتم إضافة جملة "نص جديد" على النص الحاليّ، فإذا ما أردت أن تتمّ كتابتها في سطر مستقلّ، فاستخدم التعبير:

**TextBox1.AppendText(vbCrLf & " نص جديد")**

حيث vbCrLf هو ثابت من ثوابت VB، حيث Cr هي اختصار كلمة "منعطف السطر" Carriage Return، وهو رمز يدلّ على نهاية السطر، يناظر الرقم 13 في الأرقام الدالة على الحروف ASCII، و Lf هي اختصار تعبير "مغذي السطر" Line Feed، وهو رمز يدل على بداية السطر، يناظر الرقم 10.. الخلاصة أنّ هذا الثابت يؤدّى لإضافة سطر جديد لمربع النص، بطريقة بديلة لاستخدام الجملة المتاحة التالية:

**TextBox1.AppendText(Chr(13) & Chr(10) & "نص جديد")**

**للقراءة فقط ReadOnly:**

بجعل هذه الخاصيّة "صوابا"، لن يتمكّن المستخدم من تغيير النص المعروض في مربع النص.

وكما ذكرنا سابقا، فإنّ هذه الخاصيّة كانت تحمل في الماضي اسم "مغلق" Locked، ولكنّ هذا الاسم الآن صار لخاصيّة أخرى، تقوم بتثبيت الأداة على النموذج، حتّى لا يتم تحريكها في وقت التصميم.

**حرف كلمة المرور PasswordChar:**

عندما تستخدم مربع نصّ لاستقبال كلمة المرور من المستخدم، يمكنك أن تضع أي حرف في هذه الخاصيّة، ليتم تحويل كل ما يكتبه المستخدم إلى هذا الحرف، بحيث لا يرى المحيطون به الحروف الحقيقيّة لكلمة المرور.. وفي هذه الحالة، لن يتمكّن أي مستخدم من نسخ أو قصّ أي جزء من كلمة المرور.

عامّةً، لا تؤثّر هذه الخاصيّة على قيمة النص الموجود في مربع النصّ، فالحرف الذي تراه مكتوبا هو مجرد طريقة للعرض، ولكن لو استخدمت خاصية "نص" Text، فيمكنك التعامل مع النص الأصليّ الذي أدخله المستخدم.

**تحديد النصوص Text-Selection:**

من أهمّ الأمور عند التعامل مع مربّع النصّ، قدرتك البرمجيّة على تحديد النصوص التي يبحث عنها المستخدم، أو التي تريد أن ترشده لوجود خطإ بها.

**خصائص تحديد النص:**

لو حدّد المستخدم أيّ جزء من النصّ في مربع النصّ، فكيف تعرف هذا الجزء وتتعامل معه؟.. لديك ثلاث خصائص في هذا الصدد:

|  |  |
| --- | --- |
| "النصّ المحدّد" SelectedText | لمعرفة أو تغيير الجزء المحدّد حاليا من النصّ.. فمثلا، يمكنك تحويل الحروف المحّددة إلى حروف كبيرة Capital، بالجملة التالية:  TextBox1.SelectedText = \_  TextBox1.SelectedText.ToUpper()  ولمسح النصّ المحدّد حاليا، استخدم جملة كالتالية:  TextBox1.SelectedText = ""  كما يمكن استخدام هذه الخاصيّة لإدراج نصّ ما في الموضع الحاليّ لمؤشّر الكتابة كالتالي:  TextBox1.SelectedText = "أي كلام"  حيث تقوم الجملة السابقة بكتابة النص "أي كلام" بدلا من النصّ المحدّد حاليا في مربع النصّ، ولو لم يكن هناك أي نص محدّد، يتم إضافة النّصّ "أي كلام" في الموضع الحالي لمؤشّر الكتابة. |
| "بداية التحديد" SelectionStart | لمعرفة أو تغيير رقم أول حرف في النص المحدّد.. أو لمعرفة أو تغيير الموضع الذي يوجد به مؤشّر الكتابة حاليّا، مع ملاحظة أنّ الموضع 0، يعني أنّ المؤشّر موجود قبل أوّل حرفٍ في مربع النصّ. |
| "طول التحديد" SelectionLength | لمعرفة أو تغيير طول الجزء المحدّد من النصّ، مع ملاحظة أنه في حالة وجود نصّ محدّد (طول التحديد أكبر من صفر)، فإنّ تغيير بداية التحديد SelectionStart، يعمل تلقائيّا على تغيير طول التحديد السابق زيادةً أو نقصًا. |

والكود التالي يبحث عن كلمة "مصر" في مربع النصّ، ويحدّدها بحيث تظهر جليّةً للمستخدم:

**Dim SeekString As String**

**Dim TextStart As Integer**

**SeekString = "مصر"**

**TextStart = TextBox1.Text.IndexOf(SeekString)**

**If TextStart > 0 Then**

**TextBox1.SelectionStart = TextStart – 1**

**TextBox1.SelectionLength = SeekString.Length**

**End If**

**' الجملة التالية تعمل على إجبار المنزلقين الأفقي والرأسي**

**' على تعديل وضعيهما، بحيث يظهر للمستخدم ذلك الجزء المحدّد من النصّ**

**TextBox1.ScrollToCaret()**

**إخفاء التحديد HideSelection:**

إذا انتقل المستخدم من مربّع النصّ إلى أيّ أداة أخرى أو نموذج آخر، فإنّ النصّ المحدّد في مربّع النصّ لن يظلّ مضاءً Highlighted، حتّى إنّك ستظنّ أنّ التحديد قد أزيل.. وعند العودة إلى مربع النصّ، سيُضاء التحديد وستراه من جديد.

إنّ ذلك يحدث، لأنّ لخاصيّة "إخفاء التحديد" HideSelection القيمة الافتراضيّة "صواب"، ولو شئت لبدّلت هذه القيمة إلى "خطأ"، ممّا سيعمل على بقاء التحديد ظاهرا في كلّ وقت، حتّى لو انتقلتَ إلى أداة أخرى أو نموذج آخر.

**وسائل تحديد النصّ:**

بالإضافة لاستخدام خواص تحديد النصّ، يمكنك استخدام الوسيلة "حدّد" Select، لتحديد نص معيّن.. ولهذه الوسيلة الصيغة التالية:

**TextBox1.Select(البداية , الطول)**

فمثلا، يمكنك تحديد النصّ من الحرف رقم 100 إلى الحرف رقم 105 كالتالي:

**TextBox1.Select(99, 6)**

وكما ذكرنا من قبل، فإنّ كتابة نصّ على سطرين، يعني وجود حرفين زائدين: "منعطف السطر" Carriage Return و"مغذّي السطر" Line Feed، لهذا يجب أن تضيف 2 على طول التحديد، إذا كان التحديد مستمرا عبر سطرين.

مثال: افرض أنّ مربع النصّ يحتوي على النصّ "متاهات"، فإنّ الجملة التالية تحدّد المقطع "تاه"

**TextBox1.Select(1, 3)**

ولكن لو كسرت النص إلى السطرين التاليين:

متا

هات

فإنّ تحديد المقطع "تاه" يتم كالآتي:

**TextBox1.Select(1, 5)**

بسبب وجود الحرفين المعبّرين عن كسر النصّ إلى سطرين.

ويمكن أن تحدّد كل النصّ الموجود في مربّع النصّ باستخدام الجملة التالية:

**TextBox1.Select(0, TextBox1.TextLength)**

أو يمكنك تحديده مباشرةً باستخدام الوسيلة "تحديد الكلّ" SelectAll:

**TextBox1.SelectAll()**

وهناك وسيلة هامّة ترتبط بتحديد النصّ سبق أن أشرنا إليها، وهي الوسيلة "انزلاق إلى العلامة الضوئيّة" ScrollToCaret:

**TextBox1.ScrollToCaret()**

وهي تقوم بتحريك المنزلق ليعرض الموضع الذي يوجد فيه مؤشر الكتابة (أو التحديد) حاليّا، بحيث يصبح ظاهرا للمستخدم.

**التراجع عن عمليات التحرير Undoing Edits:**

يمكن التراجع Undo عن آخر عملية تحرير قمت بها في مربّع النّصّ (مثل كتابة أو حذف بعض الحروف)، بمجرّد ضغط Ctrl+Z من لوحة المفاتيح.

ويمكن أن تفعل ذلك من الكود أيضا.. تحقّق أولا من قيمة خاصيّة "يمكن الإعادة" CanUndo، فلو كانت "صوابا" True، يمكنك استدعاء الوسيلة "تراجع" Undo.

ما يجب أن تلاحظه هنا، أنّ كتابة الحروف في النصّ دون حذف أيّ منها، يعتبر عمليّة تحرير واحدة، يمكن التراجع عنها دفعة واحدة.. معنى هذا أنّ المستخدم الذي استمرّ ساعةً في كتابة النصوص في مربّع النصّ بدون تعديل، يمكن أن يصاب بسكتة قلبيّة عندما يفاجأ بأنّك قد تراجعت عن كل ما كتبه بعمليّة واحدة!

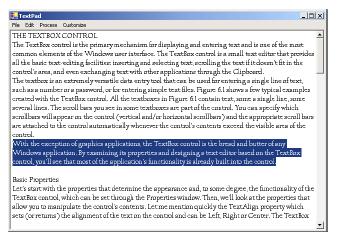
لحسنّ الحظّ أنّ تكرار عمليّة التراجع مرّة أخرى، سيعيد ما تمّ حذفه من جديد، وكأنّها عمليّة إعادة ReDo.

ويمكنك إلغاء عمليّة الإعادة عن طريق استدعاء الوسيلة "مسح معلومات التراجع" ClearUndo.

**مشروع محرر النصوص TextPad:**

الآنَ لنا أن ننشئ تطبيقا قويًّا متكاملا، نطبّقُ فيه معظم ما تعلمناه عن مربّع النصّ، وما تعلّمناه في الفصول السابقة عن القوائم والأحداث وغيرهما.

هذا التطبيق هو محرّر النصوص، وسيكون شبيها ببرنامج NotePad، ذلك الذي يأتي مع الويندوز (يمكنك أن تجده في مجلّد الملحقات Accessories في قائمة البرامج Programs Menu.



ابدأ مشروعًا جديدا، وسمِّ النافذة الرئيسيّة TxtPadForm، وضع عليه مربع نصّ سمِّهِ Editor، وثبت هوامشه الأربعة باستخدام خاصيّة Anchor.. ثمّ أضف نموذجا آخر للبرنامج، باستخدام القائمة "مشروع" / "إضافة نموذج ويندوز" Project.Add Windows Form، وسمِّ هذا النموذج FindForm، حيث سنستخدمه كمربع حوار البحث والاستبدال.

بعد ذلك أضف قائمة رئيسيّة للنموذج الأوّل، وأضف إليها القوائم التالية:

|  |  |  |
| --- | --- | --- |
| القائمة الرئيسيّة | الأمر | وظيفته |
| ملفّ File | جديد New | يعرض مربع النصّ فارغا، لاستقبال ملف جديد. |
| فتح Open | يفتح ملفا موجودا على الجهاز، ليكتب محتوياته في مربّع النصّ. |
| حفظ Save | يحفظ النصّ الموجود في مربع النصّ للملفّ المفتوح حاليا، فإذا لم يكن هناك ملف مفتوح، يتم إنشاء ملف جديد. |
| حفظ باسم Save As | يحفظ النصّ الموجود حاليا في مربع النصّ، في ملف جديد. |
| خروج Exit | يغلق البرنامج. |
| تحرير Edit | تراجع / إعادة Undo/Redo | للتراجع عن آخر عملية تحرير حدثت للنصّ، أو إعادتها مرّة أخرى بعد التراجع عنها. |
| نسخ Copy | ينسخ النصّ المحدّد حاليا في مربّع النصّ إلى لوحة القصاصات Clipboard. |
| قصّ Cut | يحذف النصّ المحدّد حاليا، ولكن بعد أن ينسخه إلى لوحة القصاصات. |
| لصق Paste | يلصق النصّ الموجود في لوحة القصاصات إلى موضع مؤشّر الكتابة في مربّع النصّ، ولو كان هناك نصّ محدّد في مربّع النص، يتمّ وضع نص لوحة القصاصات بدلا منه. |
| تحديد الكلّ Select All | يحدّد كل النصّ الموجود في مربّع النصّ. |
| بحث Find | يعرض مربّع حوار البحث والاستبدال. |
| التفاف الأسطر Word Wrap | يمكّن المستخدم من اختيار إذا ما كانت الأسطر الطويلة ستلتفّ إلى سطور تالية، أم لا، وذلك على حسب وجود أو عدم وجود علامة الاختيار بجوار هذا الأمر. |
| عمليات Process | الحالة العليا Upper Case | يحوّل كل حروف النصّ المحدّد إلى حروف كبيرة Capital. |
| الحالة الدنيا Lower Case | يحوّل كل حروف النصّ المحدّد إلى حروف صغيرة Small. |
| ترقيم الأسطرNumber Lines | يضع ترقيما لكلّ سطرٍ في مربّع النصّ. |
| تخصيص Customize | الخط Font | يسمح للمستخدم باختيار خطّ النصّ وحجمه وخصائصه. |
| لون الصفحة Page Color | يسمح للمستخدم باختيار لون خلفيّة النصّ. |
| لون النص Text Color | يسمح للمستخدم باختيار لون النصّ. |

**قائمة ملف:**

سنستخدم في هذه القائمة الأدوات القياسيّة لفتح الملفّات وحفظها، وسنتعرّف عليها بالتفصيل في الفصل القادم، إن شاء الله.

**قائمة تحرير:**

**أمر التراجع أو الإعادة:**

**Private Sub EditUndo\_Click(ByVal sender As Object, \_**

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

**If EditUndo.Text = "Undo" Then**

**Editor.Undo()**

**' تغيير عنوان الأمر إلى إعادة**

**EditUndo.Text = "Redo"**

**Else**

**Editor.Undo()**

**' تغيير عنوان الأمر إلى تراجع**

**EditUndo.Text = "Undo"**

**End If**

**End Sub**

ويجب أن نأخذ في الاعتبار، أنّ المستخدم لو تراجع عن عمليّة ما، ثم قام بكتابة أو حذف أي حرف من مربع النصّ، فإنّه لن يتمكّن من الإعادة.. يمكنه فقط التراجع عن عمليّة الكتابة أو الحذف التي قام بها.. معنى هذا أنّه كلّما تغيّرت محتويات مربّع النصّ، يجب أن نجعل عنوان القائمة "تراجع" Undo.. ولكن كيف نعرف أنّ النصّ قد تغيّر؟

سنستخدم في هذا الحدث "تغيّر النصّ" TextChanged، الذي يتمّ إطلاقه كلّما تغيّر ما يحتويه مربع النصّ من نصوص:

**Private Sub Editor\_TextChanged(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles Editor.TextChanged**

**EditUndo.Text = "Undo"**

**End Sub**

**أوامر التحرير:**

سنستخدم هذه الأوامر في نقل النصوص بين مربع النصوص ولوحة القصاصات Clipboard.. وسنحتاج هنا لوسيلتين هامّتين من وسائل لوحة القصاصات:

- ضع كائن البيانات SetDataObject: لوضع أي نوع من البيانات (نص، صورة، ... إلخ) في لوحة القصاصات.

- اقرأ كائن البيانات GetDataObject: لاسترجاع المعلومات من لوحة القصاصات.

ها هو ذا كود الأمر نسخ:

**Protected Sub EditCopy\_Click(ByVal Sender As Object, \_**

**ByVal e As System.EventArgs)**

**Clipboard.SetDataObject(Editor.SelectedText)**

**End Sub**

وها هو ذا كود الأمر قصّ:

**Protected Sub EditCut\_Click(ByVal Sender As Object, \_**

**ByVal e As System.EventArgs)**

**Clipboard.SetDataObject(Editor.SelectedText)**

**Editor.SelectedText = ""**

**End Sub**

وأخيرا، ها هو ذا كود الأمر لصق:

**Protected Sub EditPaste\_Click(ByVal Sender As Object, \_**

**ByVal e As System.EventArgs)**

**With Clipboard.GetDataObject**

**If .GetDataPresent(DataFormats.Text) Then**

**Editor.SelectedText = .GetData(DataFormats.Text)**

**End If**

**End With**

**End Sub**

طبعًا لا بدّ أن تراعي أنّ لوحة القصاصات عامّة للويندوز كلّه، ومن الممكن أن تحتوي على أنواع عديدة من البيانات كالصور مثلا، لهذا يجب أن تتأكّد أولا من نوع هذه البيانات قبلَ لصقها في برنامجك.. هذا هو سبب استخدام الجملة الشرطيّة في أمر اللصق، حيث لن نقوم بهذه العمليّة إلا في حالة وجود نصوص في لوحة القصاصات، حتّى لا يحدث خطأ.

إن خاصيّة "اقرأ البيانات الموجودة" GetDataPresent، تتأكّد ممّا إذا كانت البيانات الموجودة في لوحة القصاصات من نوع المعامل المرسل إليها أم لا، وعلى أساس هذا تكون قيمتها صوابا أو خطأ.

ويمكنك أن تعرف باقي أنواع البيانات المتاحة للوحة القصاصات، باختبار المُرقَّم "أنساق البيانات" DataFormats، وذلك بمجرّد كتابة نقطة بعد اسم هذا المرقّم، لتظهر لك قائمة تعبيراته الرقميّة.

**قائمة تخصيص:**

وتقوم بفتح مربعي حوار الخط واللون القياسيّين، الذين سنتعرّف عليهما إن شاء الله في الفصل القادم.

**قائمة عمليات:**

طبعا ستجده تافها، ذلك الكود الذي يحوّل النصّ إلى الحروف الكبيرة أو الصغيرة:

**Private Sub ProcessUpper\_Click(ByVal sender As Object, \_**

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

**Editor.SelectedText = Editor.SelectedText.ToUpper**

**End Sub**

**Private Sub ProcessLower\_Click(ByVal sender As Object, \_**

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

**Editor.SelectedText = Editor.SelectedText.ToLower**

**End Sub**

أمّا ما يستحقّ العناية نوعا، فهو أمر ترقيم الأسطر، حيث سنرى فيه كيف يمكن معالجة كل سطر من سطور النصّ على حِدَة:

**Private Sub ProcessNumber\_Click(ByVal sender As Object, \_**

**e As System.EventArgs) Handles ProcessNumber.Click**

**Dim I As Integer**

**Dim NewText As New Text.StringBuilder()**

**For I = 0 To Editor.Lines.Length - 1**

**NewText.Append((I + 1).ToString & vbTab & \_**

**Editor.Lines(I) & vbCrLf)**

**Next**

**Editor.Text = NewText.ToString**

**End Sub**

لقد تعاملنا من قبل مع تعبيرات كالتالي:

**I.ToString**

ولكنّ هذه هي المرّة الأولى التي نتعامل فيها مع تعبير طريف كالتالي:

**(I + 1).ToString**

لا تتعجّب.. هذه اللغة يا صديقي هي المرونة نفسها، لدرجة أنّ بإمكانك أن تكتب تعبيرا كالتالي:

3.ToString

دعنا من هذا ولنتفرّغ لما هو أهمّ.. إنّ هذا الإجراء يستخدم متغيرا كائنا من النوع "باني النصوص" StringBuilder، وهو مماثل للنصّ العاديّ String، ولكنّه أسرع بكثير في التعامل مع النصوص.. عامّة سنناقشه بالتفصيل في الفصل 12 بإذن الله.

جرّب الآن ترقيم الأسطر ثمّ حاول أن تتراجع عنه.. ستكتشف أنّ هذه العمليّة لا يمكن التراجع عنها، لأنّ التغيير الذي حدث في مربع النصّ حدث من الكود!

إذن ما الحلّ؟

يمكن أن نقوم بخدعة صغيرة، بوضع النصّ المرقّم في لوحة القصاصات، ثم لصقه في مربع النصّ، وبذلك يمكن التراجع عن عمليّة الترقيم.

ولفعل ذلك، احذف السطر الأخير في إجراء الترقيم، واكتب بدلا منه التالي:

**Clipboard.SetDataObject(NewText.ToString())**

**Editor.SelectAll()**

**Editor.Paste()**

**عمليتي البحث والاستبدال:**

الأمر الأخير في قائمة تحرير، يعرض لك مربع حوار البحث والاستبدال، الموجود في غالبية محررات النصوص.. وهو موجود بالمناسبة في بيئة تطوير VS.NET، ليمكّنك من البحث عن أي متغيّر أو رقم أو كلمة في الكود.. جرّب إمكانيات البحث والاستبدال في VS في قائمة تحرير Edit، تحت القائمة الفرعيّة "البحث والاستبدال" Find And Replace، حيث ستجد خمسة أوامر هامّة، ستفيدك كثيرا أثناء تعديلك برامجك.

نعود الآن لتطبيقنا.. وأوّل ما سنفعله، هو أن نجعل مربع النصّ متاحا للاستخدام من نموذج البحث والاستبدال، إمّا بتغيير خاصيّة "المجال" Modifier إلى عامّ Public، أو بتعريف متغير عام مشترك في النموذج الرئيسيّ كالتالي:

**Public Shared TxtBox As TextBox**

ثمّ نربط هذا المتغيّر بمربّع النصّ، في حدث تحميل النموذج الرئيسيّ Load كالتالي:

**TxtBox = Editor**

الآن يصبح بمقدورك التعامل مع مربع النصّ من أيّ موضع في نموذج البحث والاستبدال بتعبير كالتالي:

**TxtPadForm.TxtBox.Text = ""**

**نموذج البحث والاستبدال:**

ويحتوي على الأدوات التالية:



**مربع نص البحث:**

وفيه يكتب المستخدم النصّ الذي يريد البحث عنه.. واسمه البرمجيّ SearchWord.

**مربّع نص الاستبدال:**

وفيه يكتب المستخدم النصّ الذي يريد استبداله بنصّ البحث.. واسمه البرمجيّ ReplaceWord.

**زر البحث Find:**

عند ضغط هذا الزر، يبحث البرنامج عن أوّل موضع يوجد فيه كلمة البحث، بدءا من أوّل حرف في مربع النصّ، فإذا تم العثور على الكلمة، يتمّ تحديدها، ويتم كذلك تفعيل أزرار "بحث عن التالي" Find Next ، و"استبدال" Replace، و"استبدال الكلّ" Replace All.

**زر البحث عن التالي:**

يبحث بعد الموضع الحالي، عن الموضع التالي الذي توجد فيه كلمة البحث، مع ملاحظة أنّ المستخدم لو نقل مؤشّر الكتابة إلى موضع آخر من النص، فسيتم البحث بدءا من هذا الموضع الجديد.. وهذا الزرّ غير فعّالٍ Disabled مبدئيّا، ولكن يتمّ تفعيله بعد العثور على أول بحث.

**زر الاستبدال:**

يستبدل النصّ البديل، بآخر نصّ تم العثور عليه، ثمّ يبحث عن موضع كلمة البحث التالي.. ويكون هذا الزرّ غير فعّالٍ Disabled مبدئيّا، ولكن يتمّ تفعيله بعد العثور على أول بحث.

**زر استبدال الكلّ:Replace All**

يستبدل النصّ البديل، بكلّ نسخ نصّ البحث الموجودة في مربع النصّ، وهو أيضا غير فعّالٍ Disabled مبدئيّا، ولكن يتمّ تفعيله بعد العثور على أول بحث.

**مربع الاختيار "مراعاة حالة الأحرف" Case Sensitive CheckBox:**

عند اختيار هذا المربّع، يتمّ مراعاة حالة الأحرف (كبيرة أم صغيرة) ـ طبعا في اللغة الإنجليزيّة ـ ولو أزيلت علامة الاختيار، يتمّ البحث دون الاهتمام بحالة الأحرف.

وسنستخدم في عمليّة البحث دالة "في النصّ" InStr، لأنّ لها معاملا رابعا اختياريا، يحدّد إذا ما كان البحث يراعي حالة الأحرف أم لا، بينما الوسيلة "رقم الحرف" IndexOf لا تبحث إلا مع مراعاة حالة الأحرف Case-sensitive.

**أكواد الأزرار:**

ها هو ذا كود زرّ البحث:

**Private Sub bttnFind\_Click(ByVal sender As Object, \_**

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

**Dim SelStart As Integer**

**Dim SrchMode As CompareMethod**

**' تحديد قيمة متغير نوع البحث، بناء على قيمة مربع الاختيار**

**If ChkCase.Checked = True Then**

**' البحث ثنائي، ممّا يعني أنّه يطابق حالة الحرف كما هي (مراعاة حالة الحرف)**

**SrchMode = CompareMethod.Binary**

**Else**

**' البحث نصّيّ، ممّا يعني أنّه لا يراعي حالة الحرف**

**SrchMode = CompareMethod.Text**

**End If**

**With TxtPadForm.TxtBox**

**SelStart = InStr(1, .Text, SearchWord.Text, SrchMode)**

**If SelStart = 0 Then**

**MsgBox("لم يتم العثور على الكلمة")**

**Exit Sub**

**End If**

**' لاحظ أن هذا الجزء يعتبر بمثابة جملة Else، لأنّ جملة Exit Sub لن تسمح بتنفيذ**

**' هذا الجزء في حالة تحقّق الشرط (عدم وجود عنصر البحث).**

**.Select(SelStart - 1, SearchWord.Text.Length)**

**.TxtBox.ScrollToCaret() ' تأكد من ظهور النص أمام المستخدم**

**bttnFindNext.Enabled = True**

**bttnReplace.Enabled = True**

**bttnReplaceAll.Enabled = True**

**End With**

**End Sub**

**ملاحظة:**

هناك نقطة قد تربكك، وهي من مواضع الأخطاء المحتملة في كتابة الكود.. تلك النقطة هي التعارض في موضع أول حرف في النصّ، بينَ دالة "في النص" InStr التي تعتبر هذا الموضع 1، وبين دالة التحديد Select، التي تعتبر هذا الموضع 0.. هذا هو السبب في أنّ موضع البداية في دالة التحديد، يقل بواحد عن نتيجة البحث (SelStart - 1)!!

ويجب أن تنتبه إلى أن غالبية وسائل النصوص، تتعامل مع النصّ باعتبار أن موضع أول حرف فيه هو 0، بينما دوال VB القديمة تعتبر هذا الموضع 1.. انتبه لهذا جيدا وأنت تكتب الكود!

وطبعا سيكون كود زر البحث عن التالي مماثلا، إلا إنّ بداية البحث ستكون هي موضع أوّل حرفٍ يلي التحديد الحالي، لهذا فإنّ الجملة التالية فقط هي التي ستختلف:

**SelStart = InStr(.SelStart + 2, .Text, SearchWord.Text, SrchMode)**

لاحظ أنّنا أضفنا 2 على موضع المؤشّر الحالي: 1 لنبدأ من الموضع التالي، و 1 لأنّ موضع بداية النصّ في خاصيّة SelStart هو 0، بينما في InStr هو 1.

وبدلا من التكرار، يمكن كتابة إجراء واحد، يتم استدعاؤه من الحدثين بمعامل مختلف يدل على موضع البداية.

أمّا كود الاستبدال فهو كالتالي:

**Private Sub bttnReplace\_Click(ByVal sender As Object, \_**

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

**If TxtPadForm.TxtBox.SelectedText = SearchWord.Text Then**

**TxtPadForm.TxtBox.SelectedText = ReplaceWord.Text**

**End If**

**bttnFindNext\_Click(sender, e) ' استدعاء إجراء البحث عن التالي**

**End Sub**

**تدريب 1:**

تبعًا للمرجع الأصليّ، كان السطر الأوّل في الإجراء السابق مكتوبا كالتالي:

**If TxtPadForm.TxtBox.SelectedText <> "" Then**

وهو ما زال مكتوبا هكذا في البرنامج الموجود في مجلّد أمثلة هذا الفصل.. هل تستطيع أن تعرفَ الخطأَ الموجود في هذه الجملة، والذي دفعني لتغييرها كما في الإجراء السابق؟

جرّب البرنامج في مجلّد هذا الفصل وحاول اكتشاف هذا الخطإ.. حاول كذلك ألا تقنط قبلَ الوصول إليه، واضعًا في اعتبارك أنّ برنامجك قد يصل لمستخدم "لمض" يحاول أن يجرّبه بطريقةٍ غير متوقّعة مستمتعا برغبةٍ شريرةٍ في إظهارك بمظهر المغفّل!

إنّ جزءا كبيرا من قوّتك كمبرمج، يقع في قدرتك على تخيّل جميع الاحتمالات المختلفة التي يواجهها برنامجك، ومن ثمّ تستطيع تلافي أيّ خطإٍ حتّى ولو كان نادر الحدوث.

وكود زر استبدال الكلّ كالتالي:

**Private Sub bttnReplaceAll\_Click(ByVal sender As Object, \_**

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

**Dim CurPos, CurSel As Integer**

**Dim SrchMode As CompareMethod**

**' تحديد قيمة متغير نوع البحث، بناء على قيمة مربع الاختيار**

**If ChkCase.Checked = True Then**

**' البحث ثنائي، ممّا يعني أنّه يطابق حالة الحرف كما هي (مراعاة حالة الحرف)**

**SrchMode = CompareMethod.Binary**

**Else**

**' البحث نصّيّ، ممّا يعني أنّه لا يراعي حالة الحرف**

**SrchMode = CompareMethod.Text**

**End If**

**With TxtPadForm.TxtBox**

**CurPos =.SelectionStart**

**CurSel =.SelectionLength**

**.Text = Replace (.Text, SearchWord.Text, \_**

**ReplaceWord.Text, , , SrchMode)**

**.SelectionStart = CurPos**

**.SelectionLength = CurSel**

**End With**

**End Sub**

تلاحظ أنّنا استخدمنا في هذا الإجراء الدالة "استبدال" Replace، وهي دالة قديمة من دوال VB الجاهزة، تأخذ المعاملات التالية بالترتيب:

* النصّ الأصليّ: الذي يتمّ البحث فيه.
* نصّ البحث: المراد العثور عليه.
* نصّ الاستبدال: الذي تريد استبداله بنصّ البحث.
* موضع البداية: الذي يبدأ البحث منه.. وهو اختياري، ولو لم يُرسل يعتبر 1، أي بداية النصّ.
* عدد مرات الاستبدال: وهو اختياري، ولو لم يرسل يُعتبر -1 بمعنى أن العدد غير محدّد (استبدال نصّ البحث حيثما عُثر عليه).
* نوع البحث: هل تُراعي حالة الأحرف أم لا، وهو أيضا معامل اختياري، ولو لم يتمّ إرساله يتم مراعاة حالة الأحرف.

**إجابة** [**تدريب1**](#ت1)**:**

لو بحث المستخدم عن كلمة ما ووجدها، فسيتم تحديدها في مربّع النصّ، ممّا يعني أنّها ستُستبدل عند ضغط زر الاستبدال.. ولكن لو قام المستخدم بتحديد أي كلمة غيرها في مربّع النصّ، فإنّ ضغط زر الاستبدال في هذه الحالة سيعمل على استبدال الكلمة المحدّدة رغم أنّها ليست هي المطلوبة!!.. من ثَمَّ يجبُ أن تتأكّد أولا قبل أن تستبدل أيّ كلمة، من أنّ الكلمة المحدّدة هي كلمة البحث وليست أيّ شيءٍ آخر!.. أرجو أن تكونَ قد اكتشفتَ هذا بنفسك.. ولو لم تفعل فلا تبتئس، فحتّى مؤلف هذا المرجع قد غفل عنه!!

**التربّص بضربات الأزرار Capturing Keystrokes:**

رغم أنّ الأحداث الناشئة عن ضغط أزرار لوحة المفاتيح ليست موقوفةً على مربّع النصّ، إلا إنّ هذه الأداة هي أهمّ موضع تحتاج فيه لمعرفة الأزرار المضغوطة والاستجابة لها.

ومن الأحداث التي يمكن استخدامها في هذا الصدد، حدث ضغط الزرّ KeyPress، الذي يحدث كلما ضغط المستخدم زرا من لوحة المفاتيح، حيث يمنحك المعامل e القدرة على معرفة الزر الذي تمّ ضغطه.

تعالَ نرى كيف نستخدم هذا الحدث:

افترض أنّك تريد إرسال النصّ الذي أدخله المستخدم برقيا عبر الهاتف.. في هذه الحالة يجب أن تكون جميع الحروف كبيرة Capital، ويجب ألا يحتوى النصّ على أيّ رمز من الرموز الخاصّة، إذ يجب أن تستبدل النصّ المنطوق برمزه، فتستبدل DLR بالرمز $، و AT بالرمز @، وهكذا.

هذا هو ما سنفعله في تطبيق TELEXPad، الذي يماثل تطبيق TextPad، إلا إنّه مخصّص لإرسال البرقيات، لهذا كتبنا الإجراء المستجيب لحدث ضغط الأزرار كالتالي:

**Public Sub Editor\_KeyPress(ByVal sender As Object, ByVal e \_**

**As KeyPressEventArgs) Handles Editor.KeyPress**

**Dim Ch As Char**

**If Char.IsControl(e.KeyChar) Then Exit Sub**

**Ch = Ch.ToUpper(e.KeyChar) ' تحويل الحرف إلى حرف كبير**

**Select Case Ch.ToString**

**Case "@"**

**Editor.SelectedText = "AT"**

**Case "#"**

**Editor.SelectedText = "BPT"**

**Case "$"**

**Editor.SelectedText = "DLR"**

**Case "%"**

**Editor.SelectedText = "O/O"**

**Case "&"**

**Editor.SelectedText = "AND"**

**Case Else**

**' الحرف المضغوط ليس حرفا خاصا، لهذا سنكتبه بدون تعديل**

**Editor.SelectedText = Ch**

**End Select**

**e.Handled = True ' إلغاء الحرف المضغوط، حتّى لا تتمّ كتابته**

**End Sub**

بادئ ذي بدء، يتمّ اختبار الحرف المضغوط، للتأكّد من أنّه ليس أحد الحروف الخاصّة (Del، Backspace، Escape، Ctrl+V، ... إلخ)، وذلك باستخدام الوسيلة "إنّه حرف تحكّم" IsControl الخاصّة بخليّة الحرف Char، فلو كانت صوابا، نغادر الإجراء في الحال، فلسنا في حاجة لأداء أي فعل زائد.

ولكن كيف نعرف الحرف المضغوط؟

يمكن ذلك باستخدام خاصيّة "الحرف" KeyChar، الخاصّة بالمعامل e، وقد حولناه إلى حرف كبير Capital قبل تخزينه في المتغيّر Ch.

وتلاحظ أنّنا استخدمنا الدالة "إلى نصّ"ToString لتحويل الحرف إلى نصّ، حتّى يمكن مقارنته بالنصوص "@"، "#".. إلخ.. وهناك طريقة أخرى لإجراء هذه المقارنة، وذلك كالتالي:

**Select Case Ch**

**Case "@"c**

**Editor.SelectedText = "AT"**

**Case "#"c**

**Editor.SelectedText = "BPT"**

**Case "$"c**

**Editor.SelectedText = "DLR"**

**Case "%"c**

**Editor.SelectedText = "O/O"**

**Case "&"c**

**Editor.SelectedText = "AND"**

**Case Else**

**Editor.SelectedText = Ch**

**End Select**

حيث إنّ وضع حرف c بجوار النصّ، يخبر VB أنّه من النوع "حرف" Char.. إنّ هذا الحرص على تناسق الأنواع يمكن تجاهله، حيث سيقوم VB بالتحويل آليا ما بين النصوص والحروف، إلا إذا كان اختيار "التحويل الدقيق" فعالا Option Strict On.

**ملحوظة1 (لمبرمجي VB6):**

لم يعد بإمكانك تغيير الحرف المضغوط من أيّ حدث من أحداث الأزرار، فخاصيّة KeyChar للقراءة فقط، ولا يمكن تغيير قيمتها.. كل ما يمكنك فعله هو إلغاء الزرّ المضغوط، باستخدام الخاصيّة "تمت الاستجابة له" Handled كالتالي:

**e.Handled = True**

فإذا كان من المحتّم تغيير الحرف المضغوط، فاتبع الوسيلة التي استخدمناها في الإجراء السابق، حيث كتبنا ما نريد في مربع النصّ، وألغينا الزر المضغوط.

**ملحوظة2:**

إذا أردت أن تتأكّد من أنّ زرّ الإدخال قد تمّ ضغطه، فاستخدم الجملة التالية:

**If e.KeyChar = vbCr Then**

حيث إنّ اسم الثابت vbCr هو اختصار لجملة Visual Basic Carriage Return، وقد تناولنا موضوع منعطف السطر Carriage Return ومغذّي السطر Line Feed بالتفصيل سابقا.. كما يمكن استخدام الجملة التالية:

**If e.KeyChar = Chr(13) Then**

وإذا كنتَ تخشى من نسيان الرقم 13، أو كنتَ ممّن يستاءونَ من استخدام الأرقام في الكود، فيمكنك استخدام الصيغة التالية:

**If e.KeyChar = Chr(Keys.Enter) Then**

وما هي إلا ثوانيَ وتفهم هذه الصيغة.. فصبرٌ جميل.

وختاما، أهديك هذه الفكرة الجميلة: كيف نمنع المستخدم من كتابة الحروف الإنجليزيّة في مربّع النصّ، بحيث يكتب فقط الحروف العربيّة وعلامات الترقيم والأرقام؟

استخدم هذا الكود في حدث ضغط الزرّ:

**If (e.KeyChar >= "a"c AndAlso e.KeyChar <= "z"c) OrElse \_**

**(e.KeyChar >= "A"c AndAlso e.KeyChar <= "Z"c) Then**

**e.Handled = True**

**End If**

أعتقد أنّه بسيط وواضح.. حيث نتأكّد ممّا لو كان الحرف المضغوط أحد الحروف من A إلى Z أو a إلى z (الحروف الكبيرة تختلف عن الحروف الصغيرة من حيث الكود الرقميّ ASCII Code الخاص بكلّ منها.. وبالتالي فالحرف الكبير لا يساوي الحرف الصغير).. فإذا كان الحرف كذلك، نقوم بإلغائه قبل أن تتمّ كتابته في مربّع النصّ.. وبهذا لن يستطيع المستخدم كتابة أيّ حرف إنجليزيّ في مربّع النصّ.

**التربّص بحروف الوظائف Capturing Function Keys:**

في برنامج Notepad الخاصّ بالويندوز، يمكنك استخدام F5 لكتابة التاريخ في الموضع الحالي من النصّ.. ماذا لو أردت أن تفعل المثل في تطبيق TextPad؟

في هذه الحالة لن يمكنك استخدام الحدث "ضغط الزر" KeyPress، لأنّ هذا الحدث لا ينطلق عند ضغط أزرار الوظائف!

لكن لحسن الحظّ، لدينا الحدثان "انخفاض الزر"KeyDown و"ارتفاع الزر"KeyUp"، اللذان يستجيبان لحروف الوظائف.

وبخلاف حدث "ضغط الزرّ" فإنّ هذين الحدثين يستقبلان الرقم الدال على الحرف المضغوط وليس الحرفَ نفسه، وذلك من خلال التعبير e.KeyCode.

ويجب أن تلاحظ أنّ رقم الزرّ KeyCode، مختلف عن ذلك المسمّى ASCII، ففي الشفرة الأخيرة، يكون لكل حرف رقم مختلف، فمثلا الحرف الكبير A يختلف رقمه عن الحرف الصغير a.. ولكن في رقم الزر KeyCode، فإنّ لكل زر على لوحة المفاتيح رقما خاصا به، وبهذا تجد أنّ حرفي A و a لهما نفس الرقم لأنّهما على نفس الزرّ.. إذن كيف نفرّق بينهما؟

إنّ هذين الحدثين يمنحانك القدرة على معرفة حالة الأزرار Shift و Ctrl و Alt، عن طريق اختبار إذا ما كانت الخصائص التالية صوابا أم خطأ: e.Shift و e.Control و e.Alt.. وبهذا تستطيع أن تعرف بالضبط أن حرف A الكبير هو المقصود، ما دام زر Shift مضغوطا، والعكس بالعكس.

وإذا أردت أن تعرف إذا ما كان حرفا تحكّم مضغوطين معا، استخدم تعبيرا كالتالي:

**If e.Control And e.Alt Then**

**' أي كود يناسبك**

**End If**

كما أنّ المعامل e يمنحك خاصيّة رابعة لاختبار حالة أزرار التحكّم، هي خاصيّة Modifiers، وهي لا تقدّم لك أيّ جديد، سوى أنّها تمزج حالة الأزرار الثلاثة معا في قيمة واحدة.. ولاستخدام هذه الخاصيّة للتأكّد من ضغط زري Alt و Control استخدم الكود التالي:

**If (e.Modifiers And Keys.Alt ) <> 0 AndAlso \_**

**(e.Modifiers And Keys.Control ) <> 0 Then**

**' أي كود يناسبك**

**End If**

ولا تشغل ذهنك بأرقام الأزرار، فبإمكانك استخدام مرقّم الأزرار Keys، لتتعامل بطريقة أوضح وأسهل مع الأزرار.. انظر لهذا المثال الذي يشرح نفسه ولا يحتاج لأيّ إيضاح:

**Public Sub Editor\_KeyUp(sender As Object, e As KeyEventArgs) \_**

**Handles Editor.KeyUp**

**Select Case e.KeyCode**

**Case Keys.F5 :**

**Editor.SelectedText = Now().ToLongDateString**

**Case Keys.F6 :**

**Editor.SelectedText = Now().ToLongTimeString**

**End Select**

**End Sub**

**تغيير لغة لوحة المفاتيح:**

كثيرا ما تحتاجُ لتغيير لغة لوحة المفاتيح إلى اللغة العربيّة، بحيث لا يُضطرّ المستخدم لتغييرها بنفسه كلّما فتح البرنامج.. اتبع معنا هذه الخطوات:

- استورد فضاء الاسم System.Globalization كالتالي:

**Imports System.Globalization**

- إذا كانت إعدادات اللغة على الجهاز هي "العربيّة ـ مصر" فعرّف متغيّرا من النوع "معلومات اللغة" CultureInfo لهذه اللغة كالتالي:

**Dim CL As New CultureInfo ("ar-EG", True)**

- ثمّ استخدم خليّة "لغة الإدخال" InputLanguage لتغيير لغة الإدخال الحاليّة CurrentInputLanguage، وذلك كالتالي:

**InputLanguage.CurrentInputLanguage = \_**

**InputLanguage.FromCulture(CL)**

ولكن.. ماذا لو كانت اللغة "العربيّة ـ مصر" ليست معدّةً على الجهاز؟.. إنّك ستوزّعُ برنامجك على أجهزةِ أناسٍ في بلدانٍ مختلفة، ولا تدري كيف أعدّ كلّ منهم لغة جهازه!

لا تخشَ شيئا.. إنّ خليّة "معلومات اللغة" CultureInfo تمنحك الوسيلة "اقرإ اللغات" GetCultures التي تمكّنك من معرفة اللغات المعدّة على الجهاز.. كما تمنحك الوسيلة "اسم اللغة بالإنجليزيّة" EnglishName، لتخبركَ باسم اللغة.. إنّ إعدادات اللغة العربيّة تبدأ بكلمة Arabic وتليها شرطة ثمّ اسم الدولة، مثل Arabic – Egypt.. والآن كلّ ما عليك هو أن تبحثَ في أسماء اللغاتِ المعدّة على الجهاز عن تلك التي تبدأ بكلمة Arabic.. ها هو ذا الكود الذي يفعل هذا:

**Dim Cs() As CultureInfo**

**Dim I As Integer**

**Cs = CultureInfo.GetCultures(CultureTypes.InstalledWin32Cultures)**

**For I = 0 To Cs.GetUpperBound(0)**

**If Cs(I).EnglishName Like "Arabic\*" Then**

**CL = Cs(I)**

**Exit For**

**End If**

**Next**

الآن يمكنك أن تغيّر لغة الإدخال إلى العربيّة وأنت مطمئنّ:

**InputLanguage.CurrentInputLanguage = \_**

**InputLanguage.FromCulture(CL)**

وحتّى لا ترهق نفسك أكثرَ من اللازم في محاولة فهم هذا الكود ـ وإن كنت لا أرى فيه أيّة صعوبة ـ فسأكتب لك الإجراء الذي يغيّر لغة لوحة المفاتيح، وكلّ ما عليك بعدها هو أن تضعه في Module وتضيفه لتطبيقك.

من قائمة Project أضف قالب كود Module لتطبيقك، وسمّه Utilities.. هكذا يجب أن يكون قالب الكود:

**Imports System.Globalization**

**Module Utilities**

**Public Sub Arabic()**

**Dim CL, Cs() As CultureInfo**

**Dim I As Integer**

**Cs = CultureInfo.GetCultures \_**

**(CultureTypes.InstalledWin32Cultures)**

**For I = 0 To Cs.GetUpperBound(0)**

**If Cs(I).EnglishName Like "Arabic\*" Then**

**CL = Cs(I)**

**Exit For**

**End If**

**Next**

**InputLanguage.CurrentInputLanguage = \_**

**InputLanguage.FromCulture(CL)**

**End Sub**

**End Module**

الآن يمكنك تغيير لغة لوحة المفاتيح في حدث تحميل النموذج بالجملة التالية:

**Arabic()**

وحيثما احتجت لهذه الإمكانيّة، أضف قالب الكود Utilities إلى تطبيقك، ليصبح هذا الإجراء متاحا للاستخدام.

**التعامل مع الخطوط**

* [**كائن الخطّ Font Object:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20التعامل%20مع%20الخطوط\1-%20كائن%20الخطّ.htm)
* [**مربّع حوار اختيار الخطوط FontDialog:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20التعامل%20مع%20الخطوط\2-%20مربّع%20حوار%20اختيار%20الخطوط.htm)

**كائن الخطّ Font Object:**

يمثّل هذا الكائن خطّا من الخطوط.. ويمكنك إنشاؤه بجملة كالتالية:

**Dim drawFont As New Font("Comic Sans MS", FontStyle.Bold)**

وهناك 13 صيغة أخرى للتعريف، كلّها تتطلّب اسم الخطّ، ثمّ تسمح لك بتحديد واحدة أو أكثر من خصائص الخط مثل حجمه وطرازه ووحدة قياس أبعاده.. إلخ.

ومن خصائص هذا الكائن:

**حجم الخط Size:**

وهي تقبل أيّ عدد مفرد Single.

**طراز الخطّ Style:**

تقبل أيّ قيمة من قيم المرقّم FontStyle، وهي:

سميك **Bold** ـ مائل *Italic* ـ عادي Regular ـ يعبره خطّ ~~Strikeout~~ ـ تحته خطّ Underline.

ويمكنك دمج أكثر من طراز باستخدام المعامل Or كالتالي:

**FontStyle.Bold Or FontStyle.Italic**

**سميك Bold:**

ترجع True إذا كان الخطّ سميكا.

**مائل Italic:**

ترجع True إذا كان الخطّ مائلا.

وللأسف هذه الخصائص للقراءة فقط، فليس بإمكانك تغيير خصائص الخطّ بعد إنشائه.. وللتغلّب على هذه المشكلة، أنشئ كائن خطّ جديد يعتمد على خصائص القديم ويعدّل ما تشاء منها.. انظر للمثال التالي:

**Dim textFont As Font**

**textFont = New Font(Me.Font.FontFamily, 2 \* Me.Font.Size, \_**

**Me.Font.Style Or FontStyle.Bold)**

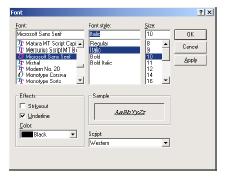
وبهذا سينتج لك كائن خطّ جديد له نفس اسم خطّ النموذج، وضعف حجمه، وله نفس الطراز مع تحويله إلى خطّ سميك ـ إن لم يكن.

**ملحوظة:**

لمزيد من التفاصيل حول كيفيّة التحكّم في خصائص كائن الخطّ، راجع موضوع [تكوين الأقنعة](file:///C:\Users\Administrator\Desktop\05-%20الملاحق\02-%20نظام%20العدّ%20الثنائي%20Binary\6-%20تكوين%20الأقنعة.htm) في ملحق رقم 2.

**مربّع حوار اختيار الخطوط FontDialog:**

إنّ مربّع الحوار هذا يسمح للمستخدم باختيار اسم الخطّ وحجمه وشكله (سميك، مائل، عادي، يعبره خطّ، تحته خطّ).



**وهذه أهمّ خصائصه:**

**السماح بتغيير اللغة AllowScriptChange:**

إذا جعلت هذه الخاصيّة True، فسيظهر على مربّع الحوار قائمة مركّبة ComboBox، للسماح للمستخدم باختيار الطائفة (اللغة) التي تنتمي إليها مجموعة حروف الخط.

**السماح بالمحاكاة AllowSimulations:**

اجعل هذه الخاصيّة True، لعرض نموذج معاينة، ليرى المستخدم تأثير اختياراته.

**السماح بخطوط متجهة AllowVectorFonts:**

**السماح بخطوط رأسيّة AllowVerticalFonts:**

**عرض الألوان ShowColor:**

اجعل هذه الخاصيّة True، للسماح للمستخدم باختيار لون الخطّ، من مربّع حوار الخطّ.

**اللون Color:**

تعيد لك اللون الذي اختاره المستخدم ليكون لون الخط.. وسيكون عليك أن تضع هذا اللون في خاصيّة ForeColor الخاصّة بالأداة التي تختار لها الخطّ.

**ثابتة الميل فقط FixedPitchOnly:**

إذا جعلت هذه الخاصيّة True، فسيتمّ عرض فقط أسماء الخطوط ذات الأحرف المتماثلة في درجة ميلانها.

**الخطّ Font:**

تعيد لك كائن خطّ، يحتوي على كلّ خصائص الخطّ الذي اختاره المستخدم، ما عدا لون الخطّ بالطبع، فهو ليس ضمن كائن الخطّ.. كما يمكنك استخدامها لعرض خطّ ابتدائيٍّ معيّن عند فتح مربّع الحوار.. انظر المثال التالي:

**' عند فتح مربّع الحوار، سيرى المستخدم مواصفات خطّ مربّع النصّ معروضة**

**FontDialog1.Font = TextBox1.Font**

**' لا بد كذلك من جعل لون الخطّ في مربّع الحوار كمثيله في مربّع النصّ**

**FontDialog1.Color = TextBox1.ForeColor**

**' عرض مربّع الحوار**

**If FontDialog1.ShowDialog = DialogResult.OK Then**

**' إذا أغلق المستخدم مربّع الحوار بضغط موافق، نغيّر خطّ مربّع النصّ**

**TextBox1.Font = FontDialog1.Font**

**' لا بدّ من تغيير لون النصّ في خطوة مستقلّة**

**TextBox1.ForeColor = FontDialog1.Color**

**End If**

**لا بدّ أن يوجد الخطّ على الجهاز FontMustExist:**

يسمح مربّع الحوار للمستخدم بكتابة اسم الخطّ.. وقد يكتب المستخدم اسم خطّ غير موجود على الجهاز.. اجعل هذه الخاصيّة True حتّى يتأكّد مربّع الحوار من صلاحية اسم الخطّ، بدلا من أن تحدث أخطاء في برنامجك.

**أصغر حجم MinSize وأكبر حجم MaxSize:**

استخدم هاتين الخاصيتين للتحكّم في مجال الأرقام التي تظهر في القائمة المركّبة التي يختار منها المستخدم حجم الخطّ.

**اللغات فقط ScriptsOnly:**

اجعل هذه الخاصيّة True، حتّى تمنع المستخدم من اختيار الخطوط التي تتكوّن حروفها من رموز غير كتابيّة.

**عرض زرّ تطبيق ShowApply:**

اجعل هذه الخاصيّة True، لكي يعرض مربّع حوار الخطّ زر "تطبيق" Apply، وهو زر لو ضغطه المستخدم يتمّ إطلاق حدث التطبيق Apply.. في هذا الحدث يجب أن تنفّذ اختيارات المستخدم على الأداة التي تغيّر خطّها، تماما كما لو أنّه ضغط زر موافق.. الفارق الوحيد بين الزرّين، هو أنّ زرّ تطبيق لا يغلق مربّع حوار الخطّ، بينما موافق يفعل.. انظر للمثال التالي:

**Private Sub FontDialog1\_Apply(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles FontDialog1.Apply**

**TextBox1.Font = FontDialog1.Font**

**TextBox1.ForeColor = FontDialog1.Color**

**End Sub**

**عرض التأثيرات ShowEffects:**

اجعل هذه الخاصيّة True، لعرض مربعي اختيار "تحته خطّ" Underline و"يعبره خطّ" ~~Strikethrough~~.. لاحظ أنّ هذه التأثيرات تظهر في كائن الخطّ، فهي من خصائصه.

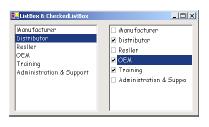
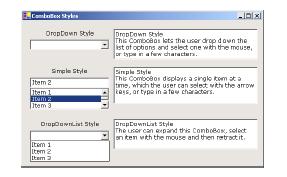
وللتدريب على استخدام هذه الأداة، افحص الأمر Font في القائمة Customize في مشروع TxtPad في مجلّد برامج الفصل السادس.

**القوائم Lists**

* [**أنواع مختلفة من القوائم:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\2-%20القوائم\1-%20أنواع%20مختلفة%20من%20القوائم.htm)
* [**الخصائص الأساسيّة المشتركة بين أنواع القوائم:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\2-%20القوائم\2-%20الخصائص%20الأساسيّة%20المشتركة%20بين%20أنواع%20القوائم.htm)
* [**مشروعListDemo :**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\2-%20القوائم\3-%20مشروع%20القائمة.htm)
* [**مشروعListBoxFind :**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\2-%20القوائم\4-%20مشروع%20البحث%20في%20القائمة.htm)
* [**بعض الخصائص الخاصة بالقائمة المركّبة ComboBox:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\2-%20القوائم\5-%20بعض%20الخصائص%20الخاصة%20بالقائمة%20المركّبة.htm)

**أنواع مختلفة من القوائم:**

لدينا ثلاث أدوات، تُستخدم كل منها لعرض مجموعة من العناصر كقائمة، بحيث يمكن للمستخدم أن يختار منها عنصرا أو أكثر: "القائمة" ListBox، و"القائمة الاختياريّة" CheckedListBox، و"القائمة المركّبة" ComboBox.

والقائمة الاختياريّة، تعرض كلّ عنصرٍ من عناصرها وبجواره مربع اختيار، بحيث يمكن للمستخدم أن يضع علامة الاختيار بجوار أيّ عدد من عناصرها.

أمّا القائمة المركّبة فهي مماثلةٌ للقائمة التقليديّة، إلا إنّها تظهر في حجم أقلّ، حيث يمكن ضغط زر إسدالها لاختيار أيّ من عناصرها، قبل أن تعود للانكماش، بالإضافة إلى أنّها تعمل كذلك كمربّع نصّ، حيث يمكن للمستخدم أن يكتب فيها ما يريد، دونَ أن يكون مقيّدا بالاختيار من عناصرها فقط.

ويمكن ملء هذه القوائم باستخدام نافذة الخصائص، باستخدام خاصيّة "العناصر" Items، والتي يمكن كتابتها بضغط زر الانتقال، حيث ستظهر لك نافذة تحرير النصوص.. اكتب فيها ما تريد من العناصر، كل عنصر في سطر مستقلّ.. وبهذا ستظهر لك هذه العناصر في القائمة عند تشغيل البرنامج.

ولكي تتعامل مع عناصر القائمة من الكود، يمكنك استخدام مجموعة العناصر Items Collection، وطبعا صار مألوفا لديك شكل المجموعات Collections، فهي تشابه المصفوفات، فمثلا أول عنصر في القائمة هو Items(0)، ويليه Items(1)، وهكذا.

وككلّ المجموعات، تحتوي مجموعة عناصر القائمة على الخواص التقليديّة: "العدد" Count، و"إضافة" Add، و"إزالة" Remove، و"إفراغ" Clear، وهي مماثلة لنظيراتها التي تعاملنا معها في القوائم وأدوات النماذج.

ويمكنك التعامل مع العناصر المحدّدة في القائمة بهذه الوسائل:

- رقم العنصر المحدّد SelectedIndex:

- مجموعة أرقام العناصر المحدّدةSelectedIndices Collection :

- العنصر المحدّد SelectedItem:

- مجموعة العناصر المحدّدة SelectedItems Collection:

فمثلا، يمكنك استخدام الكود التالي لطباعة كلّ العناصر المحدّدة في القائمة:

**Dim Itm As Object**

**For Each Itm In ListBox1.SelectedItems**

**Console.WriteLine(Itm)**

**Next**

كما يمكنك البحث عن أي عنصر في القائمة بإحدى هاتين الطريقتين، اللتين تعملان سواء كانت القائمة مرتّبة أم لا، وكلتاهما لا تراعي حالة الأحرف:

- "ابحث عن النص"FindString : للبحث التقريبي عن أوّل عنصر في القائمة يحتوي على نصّ البحث.. فمثلا لو بحثت بهذه الوسيلة عن النصّ "فصل" فمن الممكن أن تعثر على النصّ "فصلكم".

- "ابحث عن النصّ المطابق" FindStringExact: للبحث عن أول عنصر في القائمة يطابق نصّ البحث.

وتقبل هاتان الوسيلتان معاملين: عنصر البحث، ورقم الخانة التي يبدأ منها البحث في القائمة، وهو معامل اختياريّ يمكن ألا ترسله، وفي هذه الحالة سيتم البحث من أوّل عنصر في القائمة.

كما يمكن البحث عن أحد العناصر مع مراعاة حالة الأحرف، باستخدام الوسيلة IndexOf الخاصّة بمجموعة العناصر Items Collection، كالتالي:

**Dim I As Integer = ListBox1.Items.IndexOf(""عنصر البحث)**

وكما هو واضح، لا تقبل هذه الطريقة تحديد الخانة التي يبدأ منها البحث.

ولتسريع طريقة عرض العناصر في القائمة عند إضافة عناصر جديدة، استخدم الوسيلتين التاليتين:

- "بداية التحديث" BeginUpdate: عند إضافة كل عنصر جديد للقائمة، يقوم VB بإعادة رسم القائمة لإنعاشها، بحيث يظهر بها هذا العنصر، وهي عمليّة تستهلك وقتا طويلا.. لهذا استدع هذه الوسيلة قبل البدء في إضافة عناصر القائمة، وذلك حتّى يتوقفَ VB عن تكرار عملية إعادة رسم القائمة.

- "إنهاء التحديث" EndUpdate: بعد أن تنتهي من إضافة كل العناصر، استدع هذه الوسيلة، لتظهر كل العناصر التي أضيفت في القائمة.

ولنسخ عناصر قائمة لمصفوفة، استخدم الوسيلة "نسخ إلى" CopyTo:

**ListBox1.CopyTo( اسم المصفوفة,[في المصفوفة ليتم النسخ بدءا منه موضع البداية])**

**الخصائص الأساسيّة المشتركة بين أنواع القوائم:**

**ارتفاع متكامل IntegralHeight:**

إذا وضعت في هذه الخاصيّة القيمة "صواب"، فإنّ ارتفاع القائمة سيتمّ ضبطه تلقائيّا، حتّى يظهر العنصر الأخير كاملا، دون أن يختفي أي جزء منه.. بطريقة أخرى: سيتمّ عرض عدد صحيح من الصفوف.

**متعدّدة الأعمدة MultiColumn:**

واضح من الاسم، أنّ جعل هذه الخاصيّة صوابا، يمكّن القائمة من عرض عناصرها في أكثر من عمود.. وفي هذه الحالة سيظهر منزلق أفقيّ ليمكّنك من رؤية الأعمدة المختفية.

**طراز التحديد SelectionMode:**

لا يمكن تغيير قيمة هذه الخاصيّة إلا في وقت التصميم، وهي تتحكّم في كيفية تحديد المستخدم لعناصر القائمة، ولها القيم التالية:

|  |  |
| --- | --- |
| ولا واحد None | غير مسموح للمستخدم أن يحدّد أيا من عناصر القائمة. |
| عنصر واحد One | وهي القيمة الافتراضيّة، حيث يمكن تحديد عنصر واحد في المرّة. |
| متعدّد بسيط MultiSimple | يمكن تحديد عناصر عديدة في نفس الوقت، بمجرّد ضغط أي عنصر بالفأرة ضغطة واحدة، أو بضغط مسطرة المسافات.. ويمكن تحرير العنصر المحدّد وإزالة تحديده، بإعادة ضغطه بالفأرة أو مسطرة المسافات. |
| متعدّد موسّع MultiExtended | يمكن تحديد عناصر عديدة في نفس الوقت، باستخدام إحدى هذه الوسائل:   * بضغط زر Shift وضغط العنصر بالفأرة، وذلك لتحديد كل العناصر بدءا من آخر عنصر كان محددا إلى العنصر الذي تمّ ضغطه. * بضغط Shift مع أزرار الأسهم، لتوسيع التحديد أو تقليصه لأعلى أو أسفل. * بضغط زر Ctrl وضغط العناصر بالفأرة، وبذلك تتمكّن من تحديد عناصر متفرّقة لو أردت. |

**مرتّبة Sorted:**

تظهر العناصر في القائمة بنفس ترتيب كتابتك لها، ولكن لو أردت ترتيبها هجائيّا، غيّر قيمة هذه الخاصيّة إلى "صواب" True.

**ملاحظة:**

صمّمت القائمة أساسا لتعرض نصوصا، لهذا لو وضعت فيها أرقاما، وأردت من القائمة ترتيبها، فإنّ هذا الترتيب سيكون هجائيّا.. فمثلا لو وضعت في القائمة الأرقام من 1 إلى 10، وجعلت خاصيّة مرتّبة Sorted صوابا، فستظهر لك الأرقام بالترتيب التالي: (1، 10، 2، 3، 4، 5، 6، 7، 8، 9).

ويمكن حلّ هذه المشكلة، بوضع أصفار لا قيمة لها على يسار الأرقام، بحيث يتفق ترتيبها هجائيّا مع ترتيبها رقميّا: (01، 02، 03، 04، 05، 06، 07، 08، 09، 10).

**تحذير:**

لو جرّبت ملء قائمة غير مرتّبة بمئة ألف عنصر، فستستغرق هذه العمليّة حوالي 13 ثانية، ولكن لو كانت القائمة مرتّبة، فستستغرق هذه العمليّة دقائق عديدة، وذلك لأنّ VB سيستهلك وقتا ملموسا عند إضافة كل عنصر، للبحث عن موضعه في الترتيب.. لهذا يجب أن تجعل لخاصيّة Sorted القيمة False قبل أن تضع عددا كبيرا من العناصر في القائمة، ثمّ بعد ذلك تعيدها للقيمة True.. بهذه الخدعة البسيطة، ستتحوّل الدقائق العديدة إلى ثوانٍ مرّةً أخرى.

**مجموعة عناصر القائمة Items Collection:**

من الطبيعيّ أن تتخيّل أنّ القائمة قد صمّمت لتخزين مجموعة من النصوص.. ولكنّ الجديد في الأمر، هو أنّ كل عنصر من عناصر القائمة هو كائن Object.. معنى هذا أنّ بإمكانك أن تخزّن في القائمة أيّ مجموعة من الكائنات.

ستسألني متعجّبا:

- ولكن كيف سيتم عرض الكائن في القائمة؟

- لن يتمّ عرض الكائن، ولكن سيتمّ عرض النصّ الناتج عن خاصيّة ToString الخاصّة بالكائن.

- ولكن افترض أنّني لا أريد عرض هذا النصّ؟

- لا تحمل همّا.. يمكنك استخدام خاصيّة "عضو القيمة" ValueMember، لتغيير هذا الوضع الافتراضيّ، واختيار أيّ خاصيّة أخرى من خصائص الكائن، تريد عرض قيمتها في القائمة.

فمثلا، لو أضفت لونا ومستطيلا للقائمة بالجملتين التاليتين:

**ListBox1.Items.Add(Color.Yellow)**

**ListBox1.Items.Add(New Rectangle(0, 0, 100, 100))**

فسيظهر لك هذان السطران في القائمة:

Color [Yellow]

{X=0, Y=0, Width=100, Height=100}

وبإمكانك التعامل مع الكائنين عبر عنصري القائمة، فمثلا، يمكنك استخدام جملة كالتالية:

**Console.WriteLine(ListBox1.Items.Item(0).G)**

وG هو اختصار Green بمعنى أخضر.. إنّ VB يمثّل الألوان عن طريق مكوناتها الثلاثة: الأحمر والأخضر والأزرق، حيث يأخذ كل منها قيمةً ما بينَ 0 و 255.. وبهذه الطريقة يمكنك تكوين ملايين الألوان.. الخلاصة أنّ الجملة السابقة تطبع قيمة المكوّن الأخضر في اللون الأصفر (وأنت تعرف بالطبع أنّه يتكون من الأحمر والأخضر بدون أي أزرق (255، 255، 0).. لهذا ستظهر القيمة 255 في نافذة المخرجات.

كما يمكنك استخدام الجملة التالية لطباعة عرض المستطيل:

**Console.WriteLine(ListBox1.Items.Item(1).Width)**

حيث ستعرض نافذة المخرجات 100.

ومن جديد لا بدّ أن أذكّرك أن عناصر القائمة هي كائنات عامّة، واستخدام الجملتين السابقتين يُعتبر ربطا متأخّرا Late-bound، لأنّ VB لا يستطيع أن يعرف نوع الكائن على وجه الخصوص قبل تشغيل البرنامج.. لهذا من الأفضل ـ كما قلنا مرارا وتكرارا ـ استخدام دالة تحويل الأنواع CType، لتوفير العناء عن VB:

**Console.WriteLine( CType (ListBox1.Items.Item(0), Color).G )**

**Console.WriteLine( CType (ListBox1.Items.Item(1), Rectangle).Width )**

وإذا أردت أن تتأكّد من نوع الكائن أولا، استخدم جملة كالتالية:

**If ListBox1.Items.Item(0).GetType Is GetType(Color) Then**

**Console.WriteLine( CType (ListBox1.Items.Item(0), Color).G )**

**End If**

**إضافة العناصر لمجموعة العناصر:**

يمكن استخدام أي من الصيغتين التاليتين:

1. **ListBox1.Items.Add(عنصر)**
2. **ListBox1.Items.Insert( رقم العنصر, العنصر)**

انظر لهذا المثال، الذي يضيف عناصر مصفوفة لقائمة:

**Dim words(100) As String**

**' الكود المناسب لملء المصفوفة**

**Dim I As Integer**

**ListBox1.BeginUpdate ' لإيقاف إنعاش القائمة حتّى تكون إضافة العناصر أسرع**

**For I = 0 To 99**

**ListBox1.Items.Add(words(I))**

**Next**

**ListBox1.EndUpdate ' لإعادة إنعاش القائمة**

أمّا هذا المثال، فهو يضيف عنصرا في الخانة الأولى من القائمة:

**ListBox1.Items.Insert(0, "عنصر جديد")**

**ملاحظة:**

لو استخدمت طريقة الإضافة محدّدة الموضع Insert مع قائمة مرتّبة، فستتمّ إضافة العنصر في الموضع الذي تريده، ولكنّ هذا لا يضمن لك أن تظلّ القائمة مرتّبة.. هذا بخلاف طريقة Add، فهي تضيف العنصر الجديد في نهاية القائمة غير المرتّبة، وفي موضعه الصحيح في المصفوفة المرتّبة.

**حذف عنصر من القائمة:**

يمكن حذف أي عنصر من القائمة إذا كنت تعرف رقمه، باستخدام الوسيلة "حذف من الموضع" كالتالي:

**ListBox1.Items.RemoveAt(رقم العنصر)**

كما يمكن حذف العنصر نفسه باستخدام الوسيلة "حذف" كالتالي:

**ListBox1.Items.Remove(عنصر)**

سواء كان هذا العنصر نصًّا أم كائنا.. فإذا كان المراد حذف نصّ، وتصادف تكرار هذا النصّ أكثر من مرّة، يتمّ حذف النصّ الأوّل فقط.

**التأكّد من وجود عنصر معين في القائمة:**

يمكنك استخدام الوسيلة "تحتوي" Contains كالتالي:

**Dim Itm As String = "عنصر 1"**

**If Not ListBox1.Items.Contains(Itm) Then**

**ListBox1.Items.Add(Itm)**

**End If**

هذا بالإضافة إلى الوسيلة "رقم العنصر" IndexOf التي تكلمنا عنا من قبل، حيث تُرجع -1 إذا لم يكن العنصر موجودا في القائمة.

**مشروعListDemo :**

يحتوي نموذج هذا المشروع على قائمتين: واحدة تركناها بالخصائص الافتراضيّة وأسميناها "قائمة المقصد" DestinationList، والثانية جعلناها مرتّبة، وجعلنا خاصيّة "متعددة التحديد" MultiSelect "متعددة موسّعة" MultiExtended، وأسميناها "قائمة المصدر SourceList.



ويحتوي النموذج على زرين لإضافة العناصر لكل قائمة، حيث يعرض كل منهما مربع إدخال InputBox للمستخدم، ليكتب اسم العنصر الذي يريد إضافته.. وكود الزرين متماثل، لهذا فسنكتفي بإيضاح أولهما:

**Private Sub bttnAdd1\_Click(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles bttnAdd1.Click**

**Dim ListItem As String**

**ListItem = InputBox("أدخل اسم العنصر الجديد")**

**If ListItem.Trim <> "" Then SourceList.Items.Add(ListItem)**

**End Sub**

كما يحتوي النموذج على زرين لإزالة العناصر المحدّدة في كل قائمة، ونتيجةً لاختلاف خاصيّة متعددة التحديد في القائمتين، فإن كود كل من الزرين مختلف.

هذا هو كود حذف العناصر المحدّدة من قائمة المصدر:

**Private Sub bttnRemoveSelSrc\_Click(sender As Object, \_**

**e As EventArgs) Handles bttnRemoveSelSrc.Click**

**Dim I As Integer**

**For I = 0 To SourceList.SelectedIndices.Count - 1**

**SourceList.Items.RemoveAt(SourceList.SelectedIndices(0))**

**Next**

**End Sub**

**تدريب 2:**

لاحظت طبعا أننا نحذف العنصر المحدّد الأوّل فقط في كل لفّة من لفات جملة التكرار.. هل تستطيع أن تعرف لماذا؟.. وهل تقترح صيغة أخرى بديلة؟

وهذا هو كود حذف العنصر المحدّد من قائمة المقصد:

**Private Sub bttnRemoveSelDest\_Click( sender As Object, e As \_**

**EventArgs) Handles bttnRemoveSelDest.Click**

**DestinationList.Items.Remove(DestinationList.SelectedItem)**

**End Sub**

**تدريب3:**

الإجراء الأخير يحتوي على خطإ محتمل.. هل تستطيع اكتشافه؟

لدينا بعد ذلك أربعة أزرار، اثنان يحمل كل منهما سهما منفردا، لنقل العناصر المحدّدة من قائمة للأخرى، واثنان يحمل كل منهما سهما مزدوجا، لنقل كل العناصر من إحداهما للأخرى دفعة واحدة.

هذا هو كود الزر الذي ينقل العناصر المحددة من قائمة المصدر إلى قائمة المقصد:

**Private Sub bttnMoveDest\_Click(ByVal sender As Object, \_**

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

**While SourceList.SelectedIndices.Count > 0**

**DestinationList.Items.Add(SourceList.SelectedItems(0))**

**SourceList.Items.Remove(SourceList.SelectedItems(0))**

**End While**

**End Sub**

وهو مماثل لكود الزر الذي ينقل العنصر المحدد من قائمة المقصد إلى قائمة المصدر، لهذا فلا داعي لأن نذكره.

**ملاحظة:**

حتّى لو كانت القائمة لا تسمح إلا بتحديد عنصر واحد فقط، فما زال باستطاعتك أن تستخدم مجموعة العناصر المحدّدة SelectedItems Collection، ومجموعة أرقام العناصر المحدّدة SelectedIndices Collection، بالإضافة لخاصيتي العنصر المحدّد SelectedItem، ورقم العنصر المحدّد SelectedIndex.

ولنقل العناصر بالكامل من قائمة لأخرى، يمكنك استخدام طريقتين:

1- الطريقة العقيمة: بأن تنسخ العناصر واحدا بعد الآخر في جملة تكراريّة، وبعد نسخ كل عنصر تقوم بحذفه من قائمته الأصليّة.

2- الطريقة الذكيّة: بأن تستخدم الوسيلة "إضافة مجموعة عناصر" AddRang، لإضافة مجموعة من العناصر من قائمة إلى أخرى، وبعد ذلك تستخدم الوسيلة "إفراغ" Clear لإفراغ القائمة الأصليّة.

وللوسيلة AddRang الصيغتان التاليتان:

**1- List1.Items.AddRang (مجموعة عناصر من قائمة)**

**2- List1.Items.AddRang (مصفوفة)**

وبهذا يمكن كتابة كود زر نقل مصفوفة المصدر إلى مصفوفة المقصد كالتالي:

**Private Sub bttnMoveAllSrc\_Click(ByVal sender As Object, ByVal \_**

**e As EventArgs) Handles bttnMoveAllSrc.Click**

**SourceList.Items.AddRange(DestinationList.Items)**

**DestinationList.Items.Clear()**

**End Sub**

**إجابة** [**تدريب2**](#ت2)**:**

بعد حذف أول عنصرٍ محدّد في القائمة، يتم تحديث مجموعة العناصر المحدّدة لإزالته منها، فيحتلّ العنصر المحدّد التالي موضع العنصر الذي تمّ حذفه، فيصبح هو العنصر الأوّل.. لهذا فنحن لا نحتاج إلا لحذف العنصر الأوّل في كلّ مرّة.. ولو أردت أن تبدو عمليّة الحذف تقليديّة، بحيث يمكنك أن تحذف العنصر الذي يحمل رقم عدّاد جملة التكرار، فابدإ الحذف من آخر عنصر محدّد كالتالي:

**Private Sub bttnRemoveSelSrc\_Click(sender As Object, \_**

**e As EventArgs) Handles bttnRemoveSelSrc.Click**

**Dim I As Integer**

**For I = SourceList.SelectedIndices.Count - 1 To 0 Step -1**

**SourceList.Items.RemoveAt(SourceList.SelectedIndices(I))**

**Next**

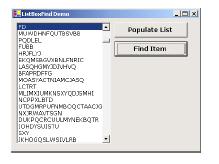
**End Sub**

**إجابة** [**تدريب3**](#ت3)**:**

حذف عنصر من القائمة باسمه هو عملية خطيرة، فلو أدخل المستخدم عنصرين بنفس الاسم، وأراد حذف العنصر الثاني، فإنّ عمليّة الحذف بالاسم Remove ستقوم بحذف العنصر الأوّل!.. لهذا لا تستخدم هذه الطريقة إلا إذا كنت واثقا من عدم وجود قيم مكرّرة، والأفضل دائما أن تحذف العنصر برقمه RemoveAt.

**مشروعListBoxFind :**

في هذا المشروع، سيتم ملء قائمة بعدد كبير من العناصر، وذلك بضغط زر "ملء" Populate، الذي سيملأ القائمة بعشرة ألاف نصّ عشوائيّ، وعند ضغط زر البحث، سيعرض لك مربع إدخال يطالبك بكتابة العنصر الذي تريد البحث في القائمة عنه.



وفي البداية، سيحاول البرنامج البحث عن نص مطابق باستخدام دالة FindStringExact، فإن لم يجد، فسيستخدم دالة FindString، لعل هناك نصّا مقاربا لنصّ البحث.

**Private Sub bttnFind\_Click(ByVal sender As Object, \_**

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

**Dim SrchWord As String**

**Dim ID As Integer**

**With ListBox1**

**SrchWord = InputBox("اكتب النص الذي تريد البحث عنه")**

**ID = .FindStringExact(SrchWord)**

**If ID >= 0 Then**

**MsgBox("رقم العنصر = " & ID.ToString & " =" & \_**

**(.Items(ID)).ToString, , "تطابق تام")**

**.TopIndex = ID**

**.SelectedIndex = ID**

**Else**

**ID = .FindString(SrchWord)**

**If ID >= 0 Then**

**MsgBox("رقم العنصر = " & ID.ToString & \_**

**" =" & (.Items(ID)).ToString, , "عثور تقريبي")**

**.TopIndex = ID**

**.SelectedIndex = ID**

**Else**

**MsgBox("العنصر " & SrchWord & \_**

**"غير موجود بالقائمة")**

**End If**

**End If**

**End With**

**End Sub**

ولكن كيف سنملأ القائمة بعشرة آلاف عنصر عشوائيّ؟

سنستخدم لفعل هذا خليّة الأعداد العشوائيّة Random Class، لتوليد رقم عشوائي بين الصفر والواحد (قد يكون الرقم الناتج صفرا، ولكنّه لا يمكن أن يكون 1)، حيث سنضربه في 20، لينتج رقم ما بين الصفر والعشرين (في الواقع أكبر رقم سينتج سيزيد عن 19 ولكنّه سيقلّ عن 20)، ونظرًا لأنّنا لا نريد الصفر، فسنجمع 1 على الناتج، لنتأكّد أنّ الرقم الناتج سيكون محصورا بين 1 و 20.. هذا الرقم العشوائيّ هو عدد حروف النصّ الذي سنكتبه في العنصر الحالي.. طبعا لا يمكن أن يكون عدد الحروف عددا عشريّا، لذلك سنستخدم الدالة CInt لتحويل هذا الرقم إلى عدد صحيح.

**WordLen = CInt(Rnd.NextDouble \* 20 + 1)**

بعد هذا سنولّد الحروف عشوائيّا بطريقة مشابهة.. إنّ الأرقام الدالة على الحروف الإنجليزيّة الكبيرة محصورة بين 65 و 91.. إذن فسنولّد رقما عشوائيّا محصورا بين الصفر و(91 – 65) أي 26، هذا الرقم سنجمعه على 65، وبهذا نضمن أن الناتج محصور دائما بين 65 و 91.. هذا الرقم سنرسله لدالة "حرف" Chr لتوليد الحرف المناظر له.

**RndChar = Chr( 65 + CInt( Rnd.Next, 26) + 1 ) )**

ولمزيد من التفاصيل حول توليد الأعداد العشوائيّة، عليك بملحق رقم 3.

ها هو الإجراء:

**Protected Sub PopulateButton\_Click(ByVal sender As Object, \_**

**ByVal e As System.EventArgs)**

**Dim WordLen As Integer**

**Dim NWords As Integer = 9999**

**Dim Rnd As System.Random = New System.Random()**

**Dim RndChar As Char**

**Dim ThisWord As String**

**Dim I, J As Integer**

**ListBox1.BeginUpdate**

**For I = 0 To NWords**

**WordLen = CInt(Rnd.NextDouble \* 20 + 1)**

**ThisWord = ""**

**For J = 0 To WordLen**

**RndChar = Chr( 65 + CInt( Rnd.Next, 26) + 1 ) )**

**ThisWord = ThisWord & RndChar**

**Next J**

**ListBox1.Items.Add(ThisWord)**

**Next**

**ListBox1.EndUpdate**

**End Sub**

**بعض الخصائص الخاصة بالقائمة المركّبة ComboBox:**

**الطراز Style:**

لديك ثلاثة طرز من القوائم المركّبة، يمكن اختيارها عن طريق خاصيّة Style، التي تأخذ القيم التالية:

|  |  |
| --- | --- |
| منسدلة DropDown | وهي القيمة الافتراضيّة.. ستظهر القائمة المركّبة كمربّع نصّ يمكن أن يكتب المستخدم به، وقائمة منسدلة يمكن الاختيار منها. |
| منسدلة كالقائمة DropDownList | تماثل القيمة السابقة، إلا إنّ المستخدم لن يكون بمقدوره الكتابة في مربّع النصّ. |
| بسيطة Simple | ستظهر القائمة المركّبة كمربّع نصّ يمكن أن يكتب المستخدم به، وقائمة غير منسدلة يمكن الاختيار منها. |

ويمكنك استكشاف هذه الطرز في مشروع ComboBoxStyles في مجلّد برامج هذا الفصل.

**عرض القائمة المنسدلة DropDownWidth:**

وأعتقد أنّ وظيفتها واضحة من اسمها.



**نصّ Text:**

تمكّنك هذه الخاصيّة من قراءة النصّ المكتوب حاليا في مربع النص الخاص بالقائمة المركّبة.. ويوضّح لك مشروع FlexComboBox كيف يمكن السماح للمستخدم بإدخال عناصر جديدة في القائمة المركّبة، وذلك لأنّ مجرّد الكتابة في مربّع نصّ القائمة، لا يضيف النصّ المكتوب لعناصر القائمة.. لهذا يجب أن تضيف أنت ما كتبه المستخدم لعناصر القائمة، إمّا بوضع زر لهذا الغرض، أو بتحديث عناصر القائمة عندما يتركها المستخدم منتقلا إلى أيّ أداة أخرى (بكتابة الكود في الحدث "فقد مؤشّر الكتابة" LostFocus)، أو عندما يضغط المستخدم زر الإدخال Enter بعد كتابة العنصر الجديد في مربع نصّ القائمة المركّبة.

**الفاصل Splitter:**

لو فتحت متصفّح الويندوز Windows Explorer، فستجد أنّه يتكون من منطقتين رئيسيّتين: المنطقة اليسرى، وهي مخصّصة لعرض أجزاء الجهاز كشجرة TreeView، والمنطقة اليمنى، وهي مخصّصة لعرض قائمة المحتويات ListView.. الطريف في الأمرِ أنّك تستطيع تكبير أيٍّ من المنطقتين على حساب الأخرى، بتحريك الفاصل الذي يحدّهما، وذلك بالتحليق فوقه بالفأرة، حيث سيتحول مؤشرها إلى سهم مزدوج الاتجاه.. هنا يمكنك ضغط زر الفأرة الأيسر وتحريك الفاصل يمينا أو يسارا.

الآن صار بإمكانك أن تفعل ذلك في مشاريعك، فهناك أداة جديدة Splitter Control ، يمكنك أن تضعها بين أي أداتين، لتمنح المستخدم القدرة على تغيير مساحتيهما.

ضع مربع نص على نموذج، وغير خاصية "متعدد الأسطر" Multiline إلى صواب، واجعل حافته اليسرى تنطبق على حافة النموذج، باستخدام خاصية Dock.. سيملأ مربع النص الجانب الأيمن من النموذج، من أعلى لأسفل.

والآن أضف فاصلا إلى النموذج، بضغط الأيقونة التي تحمل الاسم Splitter في مربع الأدوات.. سيتم إدراج الفاصل بجوار مربع النص.

أدرج مربع نص آخر على النموذج، على يمين الفاصل، وغير خاصية "متعدد الأسطر" Multiline إلى صواب، واجعله يملأ باقي مساحة النموذج، بتغيير خاصية "انطباق الحافّة" إلى "ملء" Fill.



شغّل البرنامج، وجرّب وظيفة الفاصل.. يمكنك أيضا تجربة التطبيق الموجود في مجلد أمثلة هذا الفصل، باسم Splitter1.

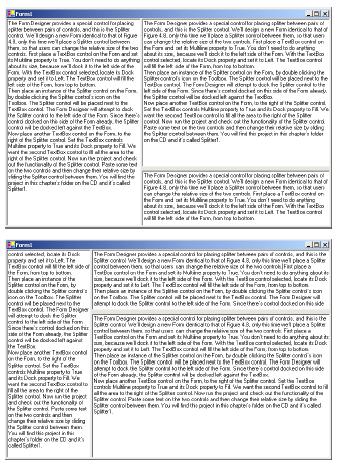
لديك أيضا التطبيق Splitter2، وفيه ستجد فاصلين، لمنحك المزيد من التحكّم في شكل الأدوات على النموذج.

إنّ الفاصل لا يمكن أن يربط بينَ أكثر من أداتين، ولكن في هذا التطبيق ستجد أنّك تستخدم الفاصل الرأسيّ ليفصل بين مربع نصّ على اليسار، ومربعي نص على اليمين، مع وضع فاصل أفقيّ ليفصل بين هاتين الأداتين وبين مربّع نصّ ثالث.. تُرى كيف ذلك؟

هنا خدعة بسيطة: لقد وضعنا مربعي النص في أداة احتواء Container (وهي هنا "اللوحة" Panel)، وجعلنا الفاصل يربط بين مربع النص الثالث واللوحة!

بعد ذلك وضعنا فاصلا أفقيّا ليفصل بينَ مربعي النص الموجودين على اللوحة.

وليست اللوحةَ أداة الاحتواء الوحيدة.. فيمكن كذلك استخدام "مربّع تجميع الأدوات" GroupBox (وكان قديما يٌسمّى "الإطار" Frame).



**ملاحظة:**

الفاصل الأفقيّ هو نفسه الفاصل الرأسيّ، ولكن تم تغيير خاصيّة Dock من "اليسار" أو "اليمين" إلى "أعلى" أو "أسفل".

**أداة تلميحات الشاشة ToolTip**

* [**وظيفتها:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20أداة%20تلميحات%20الشاشة\1-%20وظيفتها.htm)
* [**أهمّ الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20أداة%20تلميحات%20الشاشة\2-%20أهمّ%20الخصائص.htm)
* [**أهمّ الوسائل:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\4-%20أداة%20تلميحات%20الشاشة\3-%20أهمّ%20الوسائل.htm)

**وظيفة أداة تلميحات الشاشة:**

من التغييرات التي لحقت بأدوات VB، إلغاء خاصيّة ToolTipText التي كانت نماذج وأدوات VB6 تمتلكها، واستحداث الأداة ToolTip بديلا لها.

أعرف أنّك ستهمهم في ضجر: "مجرّد تغيير من أجل التغيير، ولتعقيد ما لا لزوم لتعقيده!".

ولكنّ الأمر ليس كذلك.. إنّ هذا التغيير سيعطيك تحكّما أكبر في أداء تلميحات الشاشة، كما سترى بعد قليل.

ابدأ بضغط أيقونة هذه الأداة في صندوق الأدوات.. ستجد أنّ نسخة منها قد أضيفت لصينيّة المكوّنات Component Tray أسفل النموذج.

الآن يمكنك استخدام هذه الأداة لعرض تلميحات الشاشة للعدد الذي تريده من الأدوات.

انقر سطح النموذج بالفأرة، وفي حدث تحميل النموذج اكتب الجملة التالية:

ToolTip1.SetToolTip(Me, "نموذج")

إنّ الوسيلة "وضع تلميح شاشة" SetToolTip تأخذ معاملين.. النموذج أو الأداة التي سيتمّ عرض التلميح كلّما حلّقتَ بالفأرة فوقها، وطبعا نصّ هذا التلميح.



شغّل التطبيق، وحلّق بالفأرة فوق النموذج.. سرعان ما سيظهر لك مستطيل أصفر مكتوب به كلمة "نموذج".

آه.. عُدتَ تهمهم في ضجر: "كلّ هذا العنت من أجل هذه الحركة الساذجة؟".

ليس الأمر كذلك.. إنّ هذه التلميحات هامّة جدّا بالنسبة للمستخدم، فبمجرّد أن يشير بالفأرة للأزرار والقوائم وغير ذلك من عناصر تطبيقك، فستظهر له تلميحات توضّح له وظيفة كلّ أداة.. حاول أن تتخيّل كميّة الامتنان التي سيشعر بها المستخدم تجاهك حينما تبدو له الأمور بهذه السلاسة!

ما علينا.. أضف زرّا للنموذج، وأضف السطر التالي لحدث تحميل النموذج:

**ToolTip1.SetToolTip(Button1, "اضغط هذا الزرّ لإغلاق البرنامج")**

شغّل التطبيق مرّة أخرى.. حلّق بالفأرة فوق النموذج.. سيظهر لك التلميح "نموذج".. حلّق فوق الزرّ.. سيظهر لك التلميح "اضغط هذا الزرّ لإغلاق البرنامج".

إنّ معنى هذا أنّ أداة تلميحات الشاشة لا تقتصر على عرض تلميح واحد لأداة واحدة، بل تمكّنك من عرض ما تريد من تلميحات لما تريد من أدوات.

أتوقّع منك السؤال المشاغب التالي:

* وماذا لو وضعت أكثر من تلميح لنفس الأداة؟
* عادي.. ستعرض الأداة الأخير منهما فقط، فلا يمكن عرض أكثر من تلميح لنفس الأداة!
* ولكن.. ألا توجد طريقة لوضع تلميحات الشاشة للأدوات، في وقت التصميم؟
* والله...... لحسن حظّك يوجد!
* حقّا.. كيف ذلك؟
* بعد وضع نسخة من أداة تلميحات الشاشة للنموذج، ستظهر خاصيّة في نافذة الخصائص لكلّ أداة، اسمها:

ToolTipOnX

حيث تشير X إلى اسم أداة التلميحات.. في هذه الخاصيّة، يمكنك كتابة التلميح الخاص بكلّ أداة!

* نعم هكذا.. هذه هي البساطة.
* أيّ خدمة!

**خصائص أداة التلميحات:**

تمتلك هذه الأداة الخصائص التالية:

**فعّالة Active:**

إذا جعلت هذه الخاصيّة False، فسيتوقّف عرض تلميحات الشاشة لكلّ الأدوات.. لاحظ أنّ بإمكانك أن تضع أكثر من أداة تلميحات على نفس النموذج.. إنّ هذه الخاصيّة لن تؤثّر إلا على الأدوات المرتبطة بالأداة الحاليّة فقط.

**عرض دائم ShowAlways:**

اجعل قيمة هذه الخاصيّة True إذا أردت أن تعرض تلميح الشاشة حتّى لو كان النموذج الذي يحتوي على أداة التلميحات غير فعّال حاليّا.. إنّ ذلك سيفيدك عندما تتعامل مع مجموعة من النماذج التي يظهر بعضها كمربّع حوار، بحيث يجبر المستخدم على التعامل معه فقط إلى أن يُغلقه.

**التأخير المبدئيّ InitialDelay:**

تمثّل هذه الخاصيّة الزمن الذي سينقضي قبل ظهور التلميح، بدئا من لحظة سكون مؤشّر الفأرة فوق الأداة.. وكما هي القاعدة في قياس الزمن في لغتنا، فإنّ قيمة هذه الخاصيّة تقاس بأجزاء من الألف من الثانية Milliseconds.

**مدّة ظهور التلميح AutoPopDelay:**

تمثّل هذه الخاصيّة المدّة التي سيظلّ التلميح معروضا خلالها، طالما لم يحرّك المستخدم مؤشّر الفأرة.

**مدّة التأخير قبل إعادة عرض التلميح ReshowDelay:**

تمثّل هذه الخاصيّة الوقت الذي سينقضي بين ظهور تلميحين لأداتين مختلفتين.

**تأخير تلقائيّ AutomaticDelay:**

يمكنك بوضع قيمة هذه الخاصيّة، تحديد قيم الخصائص الثلاث السابقة، حيث ستصبح قيمة خاصيّة InitialDelay مكافئة لهذه الخاصيّة، بينما ستصبح قيمة خاصيّة AutoPopDelay عشرة أضعاف هذه الخاصيّة، وقيمة خاصيّة ReshowDelay خُمس قيمة هذه الخاصيّة.

**أهمّ وسائل أداة التلميحات:**

**اقرإ التلميح GetToolTip:**

أرسل لهذه الوسيلة كمعامل، اسم الأداة التي تريد أن تقرأ التلميح المرتبط بها.

**إزالة كلّ التلميحات RemoveAll:**

استخدم هذه الوسيلة لإزالة كلّ التلميحات التي تمّ وضعها للأدوات.

**المنزلق :ScrollBar**

* [**وظيفته:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\5-%20المنزلق\1-%20وظيفته%20المنزلق.htm)
* [**أهمّ الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\5-%20المنزلق\2-%20أهمّ%20الخصائص.htm)
* [**أهمّ الأحداث:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\5-%20المنزلق\3-%20أهمّ%20الأحداث.htm)
* [**مشروع الألوان Colors:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\5-%20المنزلق\4-%20مشروع%20الألوان%20Colors.htm)

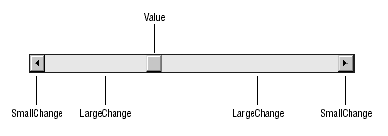
**وظيفة المنزلق:**

تمكّن هذه الأداة المستخدم من اختيار قيمة ما، بتحريك منزلقٍ بين قيمة عظمى وقيمة صغرى.. ويسمح المنزلقَ للمستخدم باختيار أيّ قيمة بحرّيّة (مجال القيم متّصل).

**ملاحظة لمبرمجي VB6:**

ليس هناك أيّ اختلاف في أداتي الانزلاق الرأسيّ والأفقيّ.

ويوجد نوعان من المنزلقات: منزلق رأسيّ VScrollBar، ومنزلق أفقيّ HScrollBar، وهما متماثلان إلا في طريقة رسمهما على النموذج فأحدهما بالطبع رأسيّ والآخر أفقيّ.



**أهمّ الخصائص:**

**الصغرى Minimum:**

القيمة الصغرى التي يقبلها المنزلق، وقيمتها الافتراضيّة 0.. ولا يوجد ما يمنعك من جعلها سالبة.

**الكبرى Maximum:**

القيمة الكبرى التي يقبلها المنزلق، وقيمتها الافتراضيّة 100، ولا يوجد ما يمنع أن تكون أكبر بكثير.

**القيمة Value:**

القيمة الحاليّة التي انزلق المستخدم إليها.. ويمكنك تغيير قيمتها من الكود، ليتغير تبعا لذلك موضع علامة الانزلاق إلى القيمة الجديدة.

**ملاحظة:**

نظرًا لأنّ الخصائص الثلاث السابقة لا تقبلُ إلا أعدادا صحيحة، فسيبدو لك أنّك لا تستطيع تمثيل الكسور.. يمكنك ذلك، ولكن بخدعة صغيرة.. فلو أردت مثلا أن تجعل المنزلق يمثّل رقما بين 2.5 و 8.5، يمكنك أن تجعل قيمة المنزلق الصغرى 25، والكبرى 85، وأيّ قيمة تحصل عليها من المنزلق تقسمها على 10.

**أهمّ الأحداث:**

لدينا هنا حدثان في غاية الأهمّيّة: القيمة تغيّرت ValueChanged وانزلاق Scroll.

وكلا الحدثين يحدث في كلّ مرّة يتغيّر فيها موضع المنزلق، ولكنّ الفارق الوحيد، هو أنّ حدث الانزلاق لا يتمّ استدعاؤه إذا غيّرتْ قيمة المنزلق من الكود.

ويمكن أن يتغيّر وضع المنزلق لأكثر من سبب:

- إذا ضغط المستخدم سهم المنزلق العلويّ (الأيسر) أو السفليّ (الأيمن).

- إذا ضغط المستخدم مسار الانزلاق بالفأرة.

- إذا سحب المستخدم مؤشّر الانزلاق إلى أيّ موضع.

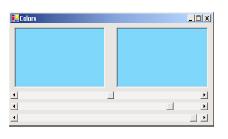
- إذا ضغط المستخدم أزرار الأسهم من لوحة المفاتيح.

ولو أردت أن تعرف سبب الانزلاق، فاستخدم خاصيّة النوع الخاصّة بالمعامل e (e.Type)، والتي تأخذ القيم التالية:

|  |  |
| --- | --- |
| القيمة | معناها |
| نهاية الانزلاق EndScroll | المستخدم قد توقّف عن تحريك المنزلق. |
| البداية First | المنزلق وصل لقيمته الصغرى. |
| إنقاصٌ كبير LargeDecrement | المستخدم قد ضغط مسار المنزلق بالفأرة بين المؤشّر والقيمة الصغرى، ممّا عمل على تقليل قيمة المنزلق بمعّدل كبير. |
| زيادة كبيرة LargeIncrement | المستخدم قد ضغط مسار المنزلق بالفأرة بين المؤشّر والقيمة الكبرى، ممّا عمل على زيادة قيمة المنزلق بمعّدل كبير. |
| النهاية Last | المنزلق وصل لقيمته الكبرى. |
| إنقاص صغير SmallDecrement | المستخدم قد ضغط سهم المنزلق الأيسر أو العلوي، أو ضغط سهما لوحة المفاتيح المناظرين، ممّا عمل على إنقاص قيمة المنزلق بقيمة صغيرة. |
| زيادة صغيرة SmallIncrement | المستخدم قد ضغط سهم المنزلق الأيمن أو السفليّ، أو ضغط سهما لوحة المفاتيح المناظرين، ممّا عمل على زيادة قيمة المنزلق بقيمة صغيرة. |
| تم سحب المؤشّر ThumbPosition | انتهي المستخدم من تغيير موضع مؤشّر الانزلاق. |
| المؤشّر يُسحب ThumbTrack | ما زال المستخدم يسحب مؤشّر المنزلق. |

**مشروع الألوان Colors:**

ذكرنا من قبل أنّك تستطيع تكوين الألوان من مكوناتها الرئيسيّة الثلاثة: الأحمر والأخضر والأزرق.. سترى الآن تطبيقا رائعا يسمح لك بتكوين ملايين الألوان من مختلف الدرجات، باختيار قيم مكوناتها عن طريق ثلاثة منزلقات، كل منها يتحرّك بين 0 و 255 (القيم المتاحة لكل مكوّن من مكونات اللون)، وقد وضعنا القيمة المبدئيّة لكل منها 128.



ولكن أين سنعرض الألوان؟

لدينا أداة تسمّى مربّع الصورة PictureBox، مخصّصة لعرض الألوان والرسوم والصور.

ولكي نوضّح الحدثين "انزلاق" Scroll و"تغيّر القيمة" ValueChanged، سنعرض اللون الناتج عن المنزلقات في مربعي صورة، أحدهما سيعرض اللون عند انطلاق الحدث "انزلاق"، والآخر سيعرضه عند انطلاق الحدث "تغيّر القيمة".. وسنكتفي هنا بذكر كود هذين الحدثين لمنزلق واحد فقط، فالآخران مماثلان له:

**Private Sub RedBar\_Scroll(ByVal sender As Object, \_**

**ByVal e As ScrollEventArgs) Handles RedBar.Scroll**

**' لن نغير لون مربع الصورة الأوّل إلا بعد التوقّف عن الانزلاق**

**If e.Type = ScrollEventType.EndScroll Then ColorBox1()**

**End Sub**

**Private Sub RedBar\_ValueChanged(sender As Object, \_**

**e As System.EventArgs) Handles RedBar.ValueChanged**

**' سنغيّر لون مربع الصورة الثاني حتّى أثناء الانزلاق**

**ColorBox2()**

**End Sub**

وها هما ذان الإجراءان اللذان يغيران لوني مربعي الصورة:

**Sub ColorBox1()**

**Dim Clr As Color**

**' تكوين اللون بناء على قيم المنزلقات الثلاثة**

**Clr = Color.FromARGB(RedBar.Value, \_**

**GreenBar.Value, BlueBar.Value)**

**PictureBox1.BackColor = Clr**

**End Sub**

**Sub ColorBox2()**

**Dim Clr As Color**

**Clr = Color.FromARGB(RedBar.Value, \_**

**GreenBar.Value, BlueBar.Value)**

**PictureBox2.BackColor = Clr**

**End Sub**

**شريط القيم TrackBar**

* [**وظيفته:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\6-%20شريط%20القيم\1-%20وظيفته.htm)
* [**مشروع Inches:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\6-%20شريط%20القيم\2-%20مشروع%20البوصة.htm)

**وظيفته:**

وهو مماثل للمنزلق، حيث يمكّن المستخدم من اختيار قيمة ما، بتحريك منزلقٍ بين قيمة عظمى وقيمة صغرى.. إلا إنّه لا يسمح للمستخدم إلا باختيار مجموعة محدّدة (منفصلة) من القيم.

ولهذا الشريط خاصيتي القيمة الصغرى والعظمى، تماما كما للمنزلق.

**ملاحظة لمبرمجي VB6:**

شريط القيم TrackBar هو الاسم البديل للاسم Slider في VB6.

ويمكنك أن تحدّد عدد الخطوات التي يتحرّكها المؤشّر عند ضغط أزرار الأسهم، عن طريق خاصيّة "أصغر تغيير" SmallChange، كما يمكنك أن تحدّد عدد الخطوات التي يتحرّكها المؤشّر عند ضغط الشريط بالفأرة، عن طريق خاصيّة LargeChange.

**مشروع Inches:**

يوضّح لك هذا المشروع كيف يمكن أن تستخدم شريط القيم للسماح للمستخدم باختيار أي طول يقع بين 0 أو 10 بوصات، بمعدل زيادة 0.2 بوصة.



ولو فتحت المشروع، فسترى أنّ هناك علامات تقسيم Tick Marks كثيرة.. يمكنك التحكّم في هذه العلامات باستخدام خاصيّة "تكرار العلامات" TickFrequency، التي توضّح عدد الذي يظهر من هذه العلامات.. فلو كانت قيمة هذه الخاصيّة 5، فهذا يعني أن العلامة الخامسة هي فقط التي ستظهر من بين كل 5 علامات.. ولكنّ هذا لا يمنع المستخدم أن يتوقّف عند العلامات غير الظاهرة.

ومن المستحسن أن تضع بعض اللافتات أسفل بعض العلامات، لتوضّح للمستخدم معناها.

وقد استخدمنا في هذا المشروع هذه القيم لخصائص شريط القيم:

|  |  |
| --- | --- |
| Maximum | 50 |
| SmallChange | 1 |
| LargeChange | 5 |
| TickFrequency | 5 |

وهذه الخصائص توضّح التالي:

- عدد علامات التقسيم = القيمة العظمى ÷ تكرار العلامات = 50 ÷ 10 = 10 علامات.. كل علامة منها تمثل 1 بوصة.

- عدد التنقلات الصغيرة بين علامات التقسيم = 5.. أيّ أنّ البوصة الواحدة مقسّمة بدقّة 1 ÷ 5 = 0.2 بوصة.

وعندما يغيّر المستخدم موضع المؤشّر، سنعرض القيمة التي اختارها في لافتةٍ على النموذج، وذلك باستخدام الحدث "القيمة تغيّرت" ValueChanged:

**Private Sub TrackBar1\_ValueChanged(ByVal sender As Object, \_**

**e As EventArgs) Handles TrackBar1.ValueChanged**

**lblInches.Text = "الطول بالبوصة = " & \_**

**Format(TrackBar1.Value / 5, "#.00")**

**End Sub**

وسيكون ذكيا منك، أن تسمح للمستخدم بتغيير قيمة شريط القيم، عند ضغط أي لافتة من اللافتات التي وضعناها تحت علامات التقسيم:

**Private Sub Label\_Click(ByVal sender As Object, e As EventArgs) Handles \_**

**Label1.Click, Label2.Click, Label3.Click, Label4.Click, \_**

**Label5.Click, Label6.Click, Label7.Click, Label8.Click, \_**

**Label9.Click**

**TrackBar1.Value = sender.text \* 5**

**' من الأذكى كتابة الجملة السابقة باستخدام التحويل الصريح كالتالي**

**' TrackBar1.Value = 5 \* CInt( CType(sender, Label).Text )**

**End Sub**

**السحبُ والإسقاط Drag And Drop**

* [**ما هو؟**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\0-%20ما%20هو.htm)
* [**السحب Drag:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\1-%20السحب.htm)
* [**مشكلة غير متوقّعة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\2-%20مشكلة%20غير%20متوقّعة.htm)
* [**تأثير السحب:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\3-%20تأثير%20السحب.htm)
* [**الإسقاط Drop:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\4-%20الإسقاط.htm)
* [**مشكلة جديدة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\5-%20مشكلة%20جديدة.htm)
* [**إلغاء عمليّة السحب:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\06-%20Basic%20Controls%20الأدوات%20الأساسيّة\7-%20السحبُ%20والإسقاط\6-%20إلغاء%20عمليّة%20السحب.htm)

**ما هو؟:**

من أشهر سماتِ الويندوز، والتي تجعله سلسلا في التعامل للغاية، قدرة مستخدمه على سحب أيّ عنصر بالفأرة من أيّ موضع، وإسقاطه في الموضع الذي يريده.. هذه العمليّة في الغالب تؤدّى لنقل العنصر Move، وأحيانًا لنسخه Copy (خاصّةً إذا ضغطت زرّ Ctrl من لوحة المفاتيح)، وأحيانا لإنشاء اختصار للعنصر Shortcut (خاصّةً إذا ضغطت زرّ Alt).. كلّ ما على المستخدم أن يفعل، هو أن يضغط أيّ عنصر من أيّ موضع بزرّ الفأرةِ الأيسر، ويحرّك الفأرة مع استمرار الضغط.. هنا ستظهر أيقونة صغيرةٌ بجوار مؤشّر الفأرة، تمثّل أيقونة العنصر المسحوب دليلا على نقله.. ولو ضغط المستخدم زرّ Ctrl من لوحة المفاتيح، تظهر مع هذه الأيقونة علامة "+" دليلا على نسخ العنصر.. ولو ضغط المستخدم زرّ Alt، تظهر علامة تمثّل سهما ملتفّا، دليلا على إنشاء نسخة مختصرة من العنصر.. بعد ذلك لا يتبقّى إلا أن يتّجه المستخدم بالفأرة إلى المكان الذي يريد إسقاط العنصر فيه، فإذا لم يتحوّل مؤشّر الفأرة إلى علامة "ممنوع التوقّف"، فإنّ العمليّة تكتملُ بمجرّد ترك المستخدم لزرّ الفأرة، حيث يتمّ إسقاط العنصر المنقول أو إسقاط نسخةٍ منه أو إسقاط اختصار له.

والسؤال الآن: هل تريد إضافة هذه الإمكانيّة لتطبيقاتك؟

أبشر إذن، فمعظم الأدوات تمتلك الوسائل والأحداث التي تؤهّلها للتعامل مع هذه العمليّة.. ما رأيك أن نرى هذا بطريقة عمليّة؟

**السحب Drag:**

ابدأ مشروعًا جديدًا أسمه DragAndDrop، وضع على نموذجه مربّع نصّ وقائمة ListBox.. أعدّ مربّع النصّ لتتمكّن من كتابة أكثر من سطر به، باستخدام خاصّيّة MultiLine.

اتبع الآن هذه الخطوات:

1- غيّر قيمة خاصيّة "السماح بالإسقاط" AllowDrop لكلّ من مربّع النصّ والقائمة إلى True.

2- في نافذة محرّر الكود، حدّد اسم مربّع النصّ في القائمة المنسدلة العلويّة اليسرى، ومن القائمة المنسدلة العلويّة اليمنى اضغط الحدث "تحرّك الفأرة" MouseMove.

3- في الإجراء المستجيب لها الحدث سنبدأ عمليّة السحب إذا كان زرّ الفأرة الأيسر مضغوطا، وذلك باستخدام الوسيلة "نفّذ السحب" DoDragDrop، التي تمتلك الصيغة التالية:

DoDragDrop(البيانات, تأثير السحب المسموح به)

حيث يمثّل المعامل الأول البيانات التي سيتمّ سحبها، والثاني نوع الأيقونة التي ستقترن بمؤشّر الفأرة عند السحب، وهو يأخذ قيمةً من قيم المرقّم DragDropEffects التالية:

|  |  |
| --- | --- |
| Copy | نسخ. |
| Move | نقل. |
| Link | اختصار. |
| Scroll | انزلاق. |
| All | سيتمّ نقل العنصر من مصدره، مع انزلاقه. |
| None | الإسقاط غير ممكن. |

تعالَ نكتب الكود الذي يسحب النصّ المحدّد في مربّع النصّ:

**Private Sub TextBox1\_MouseMove(ByVal sender As Object, e As \_**

**MouseEventArgs) Handles TextBox1.MouseMove**

**If e.Button = MouseButtons.Left Then**

**If TextBox1.SelectionLength > 0 Then**

**TextBox1.DoDragDrop(TextBox1.SelectedText, \_**

**DragDropEffects.Move)**

**End If**

**End If**

**End Sub**

الأمر بسيط: هل الزرّ الأيسر مضغوط؟.. هل هناك نصّ محدّد في مربّع النصّ؟.. إذن اسحب هذا النصّ المحدّد.

هيّا.. جرّب هذا الكود.

**مشكلة غريبة:**

لاحظ أنّ قصورا سيبرز لك في هذه العمليّة.. إنّ مجرّد البدء في عمليّة السحب يمنع VB عن الاستجابة لأحداث مربّع النصّ (ما عدا تلك الخاصّة بعمليّة السحب والإسقاط).. معنى هذا أنّ المستخدم لو أراد نسخ النصّ، يجب عليه أن يضغط Ctrl أولا ثمّ يسحب النصّ، لأنّه بعد أن يبدأ عمليّة السحب لن يكون لضغط المستخدم أو تركه لزرّ Ctrl أيّ تأثير!!

وللأسف: لا أملك حلا بسيطا لذلك.. لقد جرّبت كلّ الوسائل التالية وفشلت:

1- جرّبت استخدام الحدث KeyDown لمربّع النصّ والقائمة، فلم يفلح الأمر.

2- جعلت خاصيّة KeyPreview للنموذج True، وجرّبت استخدام الحدث KeyDown الخاصّ بالنموذج، فلم يفلح الأمر.

3- جرّبت تغيير التأثير في الحدث GiveFeedback، فاكتشفت أنّ خاصيّة e.Effect للقراءة فقط!!.. هذا بالرغم من أنّ ملفّات الإرشاد تقول إنّ هذا الحدث موجود خصّيصا لتغيير شكل مؤشّر الفأرة، ليتفاعل مع كلّ أداة يعبر فوقها!.. عامّةً فكّرتُ أن بالإمكان إنشاء مؤشّرات للفأرة Cursors، تتناسب مع تأثيرات عمليّة السحب التي نريدها، ونضعها في خاصيّة Cursor للأداة التي يعبر فوقها العنصر المسحوب.. للأسف: لم يفلح هذا الأمر أيضا، فتغيير مؤشّر الفأرة لا يتمّ إلا بعد انتهاء عمليّة السحب!!

4- جرّبت استخدام الحدث DragOver الذي ينطلق عند حركة الفأرة بالعنصر المسحوب فوق الأداة، حيث قمت بتغيير قيمة الخاصيّة e.Effect في هذا الحدث، فكانت النتيجة أنّ ظهرت علامة عدم التوقّف دليلا على عدم القدرة على إسقاط النصّ على القائمة..

المدهش أنّني حينما جرّبت ذلك في تطبيق جديد، عمل بطريقة صحيحة!!.. هناك شيء لا أدري ما هو، يعوق هذا الوظيفة في تطبيقنا! (خاصة مع اكتمال باقي التطبيق)!

سامح الله ميكروسوفت، التي جعلت من عمليّة ساذجة مأساة!

**تأثير السحب:**

نريد الآن أن نتحكّم في تأثير السحب.. ولكنّ هذا يقتضي أن نعرف إذا كان Ctrl مضغوطا أم لا، لنعرف إن كنّا سننسخ النصّ أم ننقله.. لحسن الحظّ أنّ مربع النصّ يرث من خليّة الأداة Control Class خاصيّة ModifierKeys، التي تستطيع استخدامها لمعرفة حالة الأزرار Shift و Ctrl و Alt.. الآن يجب تعديل جملة السحب في حدث حركة الفأرة لتصبح كالتالي:

**If TextBox1.ModifierKeys And Keys.Control Then**

**TextBox1.DoDragDrop(T, DragDropEffects.Copy)**

**Else**

**TextBox1.DoDragDrop(T, DragDropEffects.Move)**

**End If**

نريد الآن أن نعرض تأثير السحب، عندما يعبر مؤشّر الفأرة فوق القائمة.. هنا سنستخدم الحدث DragEnter، حيث يمنحنا المعامل e الخصائص التالية:

**التأثير المسموح به AllowedEffect:**

تخبرك هذه الخاصيّة بالتأثير الذي تمّ وضعه من مصدر السحب.

**التأثير Effect:**

تمكّنك هذه الخاصيّة من تغيير التأثير المعروض فوق الأداة الحاليّة.. وكان من الممكن أن سنستخدم هذه الخاصيّة لحلّ المشكلة السابقة جزئيّا، حيث نتحقّق من ضغط زرّ Ctrl لوضع التأثير المناسب.. ولكن للأسف، لن يعمل هذا بطريقة صحيحة!.. إنّك تشاهد مأساة من حالات القصور البرمجيّ في هذا التطبيق!!

**حالة الأزرار KeyState:**

استخدم هذه الخاصيّة لمعرفة حالة الأزرار الخاصّة Alt و Ctrl و Shift.. الجدير بالذكر أنّ هذه الخاصيّة تقبل عددا صحيحا.. إذن فلن تستطيع استخدام مرقّم الحروف Keys كما كنت متعوّدا.. فكيف يمكن اختبار الأزرار هنا؟

إنّ هذه الخاصيّة ثنائيّة Bit Wise، بمعنى أنّ كلّ زر له خانة تقابله، فإذا كان الزرّ مضغوطا تكون قيمة هذه الخانة 1، وإذا لم يكن مضغوطا تكون قيمتها 0 (راجع تكوين الأقنعة Masking في الفصل الثالث).. الخلاصة أنّ زرّ Shift يضيف إلى الرقم 4، وضغط زرّ Ctrl يضيف إلى الرقم 8 وضغط زرّ Alt يضيف إلي الرقم 32.. فإذا أردت أن تختبر زر Ctrl، فاستخدم الجملة التالية:

**If (e.KeyState And 8) <> 0 Then ' الزرّ مضغوط**

**البيانات Data:**

هذه الخاصيّة كائن من النوع IDataObject، وهي تمنحك مجموعة من الوسائل الهامّة منها:

**اقرإ البيانات الحاليّة GetDataPresent:**

تمكّنك هذه الوسيلة من التحقّق من نوع البيانات المسحوبة، حتّى تقرّر إذا كانت مناسبةً لإسقاطها على الأداة أم لا.

**اقرإ البيانات GetData:**

تمكّنك هذه الوسيلة من قراءة البيانات المسحوبة.

ها هو ذا كود الحدث.. كلّ ما سنفعله ببساطة، هو أن نتأكّد أنّ البيانات المسحوبة نصّيّة، ثم نجعل تأثير السحب فوق القائمة، هو نفس تأثير السحب المسموح به في مربّع النصّ:

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

**DragEventArgs) Handles ListBox1.DragEnter**

**If e.Data.GetDataPresent(DataFormats.Text) Then**

**e.Effect = e.AllowedEffect**

**Else**

**e.Effect = DragDropEffects.None**

**End If**

**End Sub**

**الإسقاط Drop:**

ولكن.. إلى الآن لم نكتب كود الإسقاط.. حسنا.. سنكتبه في الحدث DragDrop، الذي يحدث بعد إسقاط العنصر المسحوب:

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

**DragEventArgs) Handles ListBox1.DragDrop**

**If e.Effect = DragDropEffects.Copy Then**

**ListBox1.Items.Add(e.Data.GetData(GetType(String)))**

**ElseIf e.Effect = DragDropEffects.Move Then**

**ListBox1.Items.Add(e.Data.GetData(GetType(String)))**

**TextBox1.SelectedText = ""**

**End If**

**End Sub**

الآن يمكنك أن تجرّب سحب النصوص من مربّع النصّ وإسقاطها على القائمة، مع ضغط Ctrl أو عدمه.

**مشكلة غير متوقّعة:**

للأسف.. ستكتشف أنّ الكود لن يعمل، ليس لعيبٍ فيه، ولكن لقصورٍ قاتلٍ في مربّع النصّ، فعندما تضغط زرّ الفأرة، يتمّ إزالة التحديد ونقل مؤشّر الكتابة لموضع الضغطة أولا، ثمّ بعد ذلك يتمّ استدعاء حدث ضغط الفأرة وحدث حركة الفأرة!!.. وبهذا يستحيل أن يكون هناك نصّ محدّد بعد ضغط زرّ الفأرة الأيسر!!

ستسألني متوتّرا:

- ما الحلّ إذن؟

- بسيطة.. أرسل لميكروسوفت هذه المشكلة، وبإذن الله ستحلّها لك في الإصدار القادم من VB!

- مت يا حمار!

- من تعني بمثل هذا الكلام؟

- آسف.. لم أكن أعنيكَ حقيقةً بهذا الكلام.. إنّه مجرّد مثل شعبي!

- ما علينا!.. وإن كنت أنصحك بحسن اختيار أمثالك الشعبيّة فيما بعد!

- المهمّ الآن.. هل سنيأس ونهجر هذا التطبيق إلى الإصدار التالي؟

- ليس عندنا مثلُ هذا الكلام.. طبعا سنفكّر في حلّ يعالج هذا القصور.. إنّك لن تجد أبدا لغة البرمجة التي صمّمت لتعمل مثلما أردتها، فإذا انتظرت من شركة البرمجة علاج كلّ قصور يصادفك، فلن تكتب تطبيقا واحدا في حياتك!

- دعك من الخطابة، وادخل في لبّ الموضوع.

- حسنا.. سنفكّر كالتالي: لماذا لا نراقب نحن النصّ المحدّد في مربّع النصّ ونخزّنه في متغيّر.

- فكرةٌ خلاّقة.. ولكن كيف سننفّذها؟

- المفروض أن نجد لمربّع النصّ حدثا اسمه SelectionChanged يحدث كلّما يتغيّر التحديد.

- جميل.

- للأسف: هذا الحدث غير موجود إلا في مربّع النصّ المنسّق RitchTextBox!!

- فلنستخدم إذن مربّع النسق المنصّص هذا أو أيّا كان اسمه!

- بنيّتي.. هذا أمر شرحه يطول، وأنا الليلة مشغول!

- من تقصد ببنيّتي هذه؟

- آسف.. إنّها مجرّد تيمة لأحد البرامج الإذاعيّة!

- ما علينا!.. وإن كنت أنصحك بحسن اختيار برامجك الإذاعيّة فيما بعد!

- إنّ مربّع النصّ المنسّق مشروح بالتفصيل في فصل الأدوات المتقدّمة، ولا محلّ لإدخاله في مثال ها هنا.. وإن كنت أحبّ أن أبشّرك أنّه يقوم ببدء عمليّة السحب آليّا وبدون كتابة أيّ كود.. كلّ ما على المستخدم أن يحدّد النصّ ويسحبه.. وكلّ ما عليك أنت هو تحديد تأثير السحب (نسخ، نقل، غير مسموح... إلخ).

- إذن فلنؤجّل هذا المثال إلى فصل مربّع النصّ المنسّق.

- قلت لك ليس عندنا مثل هذا الكلام.. إنّ انهزامك كلّما عرضت لك بعض التحدّيّات، ليس أبدا من الخصال البرمجيّة الحميدة.. وإن كنت أعدك أنّني ـ بإذن الله تعالى ـ سأعيد كتابة هذا المثال في فصل الأدوات المتقدّمة، لترى السمات المتقدّمة التي يمتلكها مربّع النصّ المنسّق والشجرة TreeView وقائمة العرض ListView في موضوع السحب والإسقاط.. أمّا الآن، فلا مناص من إجراء عمليّة السحب والإجراء على مربّع النصّ اللعين هذا.. ولنرَ: نحن أو هو!

- ألا ترى أنّك تزيد الموضوع تعقيدا.. لقد بدأت أتعب!

- سلامتك.. خذ هذا العلاج: سنستخدم حدث حركة الفأرة MouseMove لمراقبة النصّ المحدّد، حيث سنضع هذا النصّ وبداية التحديد وطول التحديد في متغيّرات.. فإذا ضغط المستخدم زرّ الفأرة الأيسر، فسنتأكّد من الموضع الجديد لمؤشّر الكتابة، فإن كان ضمن النصّ الذي كان محدّدا سابقا، نعيد تحديد هذا النصّ، ونقوم بعمليّة السحب.. وإذا لم يكن كذلك، نقوم بإزالة قيم التحديد السابقة من المتغيّرات.. هه.. ما رأيك؟

- شغّال.. أرني إذن هذا الكود.

- كنت أفضّل أن .... .

- حذار أن تقولَ سأترك لك هذا الكود للتدريب!

- حسنا حسنًا.. لا تكن عصبيّا هكذا!.. هذا هو الكود:

**Private Sub TextBox1\_MouseMove(ByVal sender As Object, e As \_**

**MouseEventArgs) Handles TextBox1.MouseMove**

**' يجب أن تكون المتغيّرات التي سنخزّن بها بيانات التحديد ثابتة، حتّى لا تفقد قيمها**

**Static St Intege ' بداية التحديد**

**Static L As Integer ' طول التحديد**

**Static T As String ' النصّ المحدّد**

**Dim S As Integer = TextBox1.SelectionStart**

**If e.Button = MouseButtons.Left Then**

**If S >= St AndAlso S < St + L Then ' الضغطة فوق النصّ المحدّد**

**If T <> "" Then ' تأكّد أنّ النصّ غير فارغ**

**' نظرا لأنّ التحديد قد اختفى، فسنعيده نحن**

**' سيتسبّب ذلك في رعشة خفيفة، لأنّ التحديد سيختفي ثمّ يظهر**

**' ولكن ما باليد حيلة!**

**TextBox1.SelectionStart = St**

**TextBox1.SelectionLength = L**

**' وأخيرا.. سحب النصّ**

**TextBox1.DoDragDrop(T, \_**

**DragDropEffects.Move)**

**End If**

**Else ' الضغطة خارج النصّ المحدّد.. ألغ بيانات التحديد السابقة**

**St = S**

**L = 0**

**T = ""**

**End If**

**Else ' خزّن بيانات النصّ المحدّد**

**St = S**

**L = TextBox1.SelectionLength**

**T = TextBox1.SelectedText**

**End If**

**End Sub**

لاحظ أنّ هناك مشكلة صغيرةً في هذا الكود.. سأقولها لك كتدريب ذهنيّ، رغم أنّ اكتشاف المستخدم لها أمر مستبعد الحدوث.. هذه المشكلة تكمن في أنّ الكود السابق يفترض أنّ المستخدم يقوم بتحديد النصّ بالفأرة، أو يقوم بتحديده بلوحة المفاتيح لكنّه يحرّك الفأرة بعدها.. في مثل هاتين الحالتين يتمّ تخزين بيانات التحديد مع حركة الفأرة.. ولكن ماذا لو قام المستخدم بتحديد النصّ بلوحة المفاتيح (بضغط Shift مع الأسهم)، وكان مؤشّر الفأرة في هذه اللحظة فوق النصّ المحدّد، فضغط زرّ الفأرة دون أن يحرّكها؟.. طبعا لن يتمّ سحب النصّ.

واضح جدّا أنّها حالة معقّدة صعبة التحقيق، فعلى الأقلّ، لا بدّ أنّ يد المستخدم سترتعش وهو يمسك الفأرة!.. ولكنّي سأفترض أنّك ممّن ينشدون الكمال، وسأخبرك بحلّ هذه المشكلة.. كلّ ما عليك هو حفظ بيانات التحديد في حدث ضغط الأزرار، لو كان زرّ Shift مضغوطا.. وطبعا لا بدّ من حذف المتغيّرات الثابتة Static وإعادة تعريفها كمتغيّرات عاديّة على مستوى النموذج، بحيث تكون متاحة لحدثي ضغط الأزرار وحركة الفأرة.

هناك مشكلةٌ أخرى.. ماذا لو ضغط المستخدم زرّ الفأرة فوق النصّ المحدّد ولم يحرّك الفأرة؟.. طبعا سيختفي التحديد، وسيظلّ مختفيًا إلى أن يحرّك المستخدم الفأرة ليبدأ عمليّة السحب.

إذن فعليك أن تنقل الكود الخاصّ بإعادة تحديد النصّ من الحدث MouseMove إلى الحدث MouseDown، ولكن مع ترك الكود الذي يحفظ بيانات التحديد في المتغيّرات في موضعه، حتّى يتمّ تحديثه مع حركة الفأرة (ومع ضغط أزرار الأسهم لو أردت).. وإن كنت لا أنصحك بهذا، لأنّه سيزجّ بك في مشاكل جديدة أنت في غنى عنها.. ألا قد بلّغت.. اللهمّ فاشهد.

كانت كلّ هذه الحدوتة هي إحدى خطوات تطبيقنا.. تعال نواصل ما انقطع بيننا من حديث.

**إلغاء عمليّة السحب:**

بقي شيءٌ أخير، هو إلغاء عمليّة السحب لو ضغط المستخدم زر Esc.. طبعا سنستخدم الحدث QueryContinueDrag الذي أشرنا إليه من قبل.. هذا هو الكود:

**Private Sub TextBox1\_QueryContinueDrag(ByVal sender As Object, \_**

**ByVal e As QueryContinueDragEventArgs) Handles \_**

**TextBox1.QueryContinueDrag**

**If e.EscapePressed Then e.Action = DragAction.Cancel**

**End Sub**

ونظرًا لأنّنا في يوم القصور العالميّ، أحبّ أن أزفّ إليك هذه البشرى: رغم أنّ إلغاء عمليّة السحب يؤدّى إلى إخفاء أيقونة السحب من مؤشّر الفأرة، إلا إنّ ذلك لا يوقف العمليّة نفسها، ولو تركت زرّ الفأرة فسيتمّ استدعاء الحدث DragDrop وسيتمّ إسقاط البيانات!!.. شيءٌ غير منطقيٍّ على الإطلاق.. أعتقد أنّ ميكروسوفت يجب أن تراجع مفاهيمها حول السحب والإسقاط من جديد!!

ما علينا.. سنحلّ هذه المشكلة الجديدة بتعريف متغيّر منطقيّ نستخدمه في إلغاء العمليّة بطريقتنا.. عرّف المتغيّر التالي على مستوى النموذج:

**Dim** **CancelDrag** **As** **Boolean**

ثمّ أعد كتابة حدث إلغاء السحب كالتالي:

**Private Sub TextBox1\_QueryContinueDrag(ByVal sender As Object, \_**

**ByVal e As QueryContinueDragEventArgs) Handles \_**

**TextBox1.QueryContinueDrag**

**If e.EscapePressed Then**

**e.Action = DragAction.Cancel**

**CancelDrag = True**

**End If**

**End Sub**

والآن أضف الكود التالي لبداية الحدث DragDrop:

**If** **CancelDrag Then**

**CancelDrag = False**

**Exit Sub**

**End If**

وهكذا تكون عمليّة السحب والإدراجِ قد تمّت أخيرا بحمد الله.. حاول أنت كتابة الكود الذي ينقل أو ينسخ العناصر من القائمة إلى سطر جديد في مربّع النصّ، مع مراعاة إمكانيّة نقل النصوص من مربّع النصّ إلى نفسه، ومن القائمة إلى نفسها.

عامّةً ستجد الكود الكامل في تطبيق DragAndDrop في مجلّد برامج هذا الفصل.

وخذها نصيحةً جادّةً منّى: لا تحاول استخدام مربّع النصّ العاديّ في أيّ عمليّة سحب وإسقاط بعد ذلك، واستعِضْ عنه بمربّع النصّ المنسّق RichTextBox، إلى أن تصدر ميكروسوفت تعديلا لكلّ أوجه القصور التي صادفناها ها هنا.. وكفي بهذا المثال تدريبا لنا على حلّ المشكلات.