**الفصل الخامس**

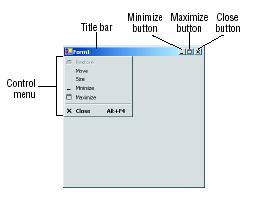
**النماذج Forms**

* [**تصميم النماذج:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\05-%20Forms%20النماذج\1-%20تصميم%20النماذج.htm)
* [**برمجة النماذج:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\05-%20Forms%20النماذج\2-%20برمجة%20النماذج.htm)
* [**تصميم القوائم Designing Menus:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\05-%20Forms%20النماذج\3-%20تصميم%20قوائم%20النماذج.htm)
* [**بناء نماذج مرنة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\05-%20Forms%20النماذج\4-%20بناء%20نماذج%20مرنة.htm)
* [**واجهة الوثائق المتعدّدة MDI:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة.htm)
* [**النماذج المترابطة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\05-%20Forms%20النماذج\6-%20النماذج%20المترابطة.htm)

**تصميم النماذج**

* [**أجزاء النموذج:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\1-%20تصميم%20النماذج\1-%20أجزاء%20النموذج.htm)
* [**أهمّ خصائص النموذج في وضع التصميم:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\1-%20تصميم%20النماذج\2-%20أهمّ%20خصائص%20النموذج%20في%20وضع%20التصميم.htm)
* [**تحديد ترتيب الانتقال بين الأدوات Setting TabOrder:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\1-%20تصميم%20النماذج\2-%20تحديد%20ترتيب%20الانتقال%20بين%20الأدوات.htm)
* [**أزرار الوصول Access Keys:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\1-%20تصميم%20النماذج\3-%20أزرار%20الوصول.htm)
* [**نماذج رائعة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\1-%20تصميم%20النماذج\4-%20نماذج%20رائعة.htm)

**أجزاء النموذج:**

يعتبر شريط العنوان من أهم أجزاء النموذج، حيث يظهر عنوان النموذج، وحيث تظهر أزرار التحكم (إغلاق Close، تكبير Maximize، تصغير Minimize)، وحيث تظهر كذلك أيقونة النموذج، والتي عند ضغطها بالفأرة، أو ضغط زر Alt ثم مسطرة المسافات من لوحة المفاتيح، تظهر قائمة التحكّم Control Menu، والتي تحتوي على هذه الأوامر:

|  |  |
| --- | --- |
| أوامر قائمة التحكّم | وظيفتها |
| استعادة Restore | يستعيد هذا الأمر حجم النموذج الأصليّ، حينما يكون في أقصى حجم له Maximized. |
| تحريك Move | يسمح للمستخدم بتحريك النموذج على الشاشة باستخدام لوحة المفاتيح، حيث سيظهر رسم يمثّل أربعة أسهم متعامدة في منتصف شريط العنوان، ليخبرك أنّ ضغط الأسهم من لوحة المفاتيح في هذه اللحظة سيحرّك النموذج.. لإنهاء التحريك اضغط زر الإدخال Enter ليستقر النموذج في الموضع الجديد، أو زر الإلغاء Esc ليبقى النموذج في موضعه القديم. |
| تغيير الحجم Size | يشبه الأمر السابق، إلا إنّ ضغط الأسهم من لوحة المفاتيح يؤدّي لتغيير حجم النموذج. |
| تكبيرMaximize | يجعل النموذج يصل لأقصى حجم له ليملأ الشاشة. |
| تصغير Minimize | يجعل النموذج يختفي من على الشاشة، ليصير مجرّد أيقونة على شريط مهام الويندوز Task Bar، (ذلك الموجود في الجزء السفليّ من الشاشة).. ولاستعادة النموذج مرة أخرى، اضغط على هذه الأيقونة في شريط المهام بالفأرة، أو اضغط Alt+Tab من لوحة المفاتيح ولا ترفع إصبعك عن زر Alt.. سيظهر لك شريط عليه أيقونات كل النماذج المفتوحة حاليا في الويندوز، اضغط زر Tab أكثر من مرّة دون أن ترفع إصبعك عن زر Alt أبدا، حتّى تجد أن المستطيل الأسود قد انتقل ليحيط بأيقونة نموذجك.. عند هذا اترك الزّرين، وسيظهر لك نموذجك مرة أخرى. |
| إغلاق Close | يغلق النموذج نهائيّا. |

وسنتعرّف الآنَ على الخصائص التي تمنحها لنا لغة البرمجة للتحكّم في هذه العناصر، لتصميم النموذج بالمظهر المناسب.

**أهمّ خصائص النموذج في وضع التصميم:**

لقد رأيت بالفعل كل أنواع النماذج أثناء تعاملك مع الويندوز.. فمثلا، أشرطة الأدوات الحرّة Floating Toolbars ما هي إلا نماذج ذات أشرطة عناوين ضيّقة.. كذلك فإنّ مربعات الحوار التي تعرض رسائل التحذير أو تطلب منك اختيار ملف ما، هي أيضا نماذج.. ولكن لماذا تبدو هذه النماذج مختلفة في الشكل؟.. إنّ هذا يرجع للعديد من خصائص النموذج، هي التي سنتعرفها الآن:

**زرا الموافقة والإلغاء AcceptButton, CancelButton:**

زر الموافقة AcceptButton هو ذلك الزر الذي يتم ضغطه تلقائيًّا عند ضغط زر الإدخال Enter، بغض النظر عن الأداة النشيطة في تلك اللحظة.. وعادة ما يكون هذا الزر مكتوبا عليه "نعم" أو "موافق" أو "حسنا" أو "فليكن" أو "OK" أو "Yes".. إلخ.

بالمثل، فإنّ زر الإلغاء CancelButton، هو ذلك الزر الذي يتم ضغطه تلقائيًّا عند ضغط زر الإلغاء Esc.. وعادة ما يكون مكتوبا عليه "إلغاء" أو "لا" أو "إغلاق" أو "Cancel" أو "No" أو "Close".

وعندما تستطلع هاتين الخاصّيتين في نافذة الخصائص، ستجد في خانة القيمة الخاصة بكل منهما زر إسدال، يعرض قائمة بكل أسماء الأزرار الموجودة على نموذجك، بحيث يمكنك أن تختار منها الزر الذي تريده أن يكون زر الموافقة أو الإلغاء.

ويمكن تحديد قيمتيهما كذلك في الكود، بجملتين كالتاليتين:

**Me.AcceptButton = Button1**

**Me.CancelButton = Button2**

**تحجيم تلقائي AutoScale:**

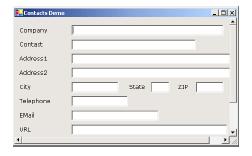
لو جعلت قيمة هذه الخاصّيّة "صواب" True، فسيتغير حجم الأدوات الموضوعة على النموذج تلقائيّا ليتناسب مع ارتفاع الخط الذي كتبت به نصوصها.. والقيمة الافتراضيّة لهذه الخاصّيّة "صواب".. هذا هو السبب الذي يمنعك من تصغير الأدوات عن حجم معيّن.

ويتمّ ضبط الأدوات على الخط الذي اكتسبتْه من النموذج عند وضعها عليه، ولكن لو تم تغيير خط النموذج فيما بعد، فلن تتأثّر الأدوات بهذا التغيير.

.

**انزلاق تلقائي AutoScroll:**

لو اخترت لهذه الخاصّية القيمة صواب، فستظهر أحد أو كلا شريطي الانزلاق Scroll Bars تلقائيّا، في حالة تغيير حجم النموذج للدرجة التي تختفي فيها بعض أدواته.



وبفضل هذه الخاصّيّة يمكنك أن تصمّم نماذج ضخمة متخمة بالأدوات، دون أن تقلق بخصوص حجم الشاشة ودقّتها.

وتنزلق هذه المنزلقات تلقائيا لتعرض الأداة النشيطة حاليا، وكلما انتقل المستخدم لأداة أخرى بضغط زر Tab، يتم عرضها آليا.

**هامش الانزلاق التلقائي AutoScrollMargin:**

هذا هو الهامش الذي يُترك حول الأدوات الموجودة على النموذج، بحيث لو صار النموذج أصغر من المستطيل الذي تشكّله الهوامش، تظهر المنزلقات.

وهذه الخاصّيّة هي كائن، يحتوي على خاصيتين فرعيتين: العرض والارتفاع، حيث يتوجّب عليك وضع قيمتيهما.

ولفعل ذلك من الكود، استخدم جملتين كالتاليتين:

**Me.AutoScrollMargin.Width = 40**

**Me.AutoScrollMargin.Height = 40**

**أصغر حجم للانزلاق التلقائيAutoScrollMinSize :**

تسمح لك هذه الخاصية بتحديد أصغر حجم ممكن للنموذج، لو صغر عنه تظهر المنزلقات.

وهي أيضا كائن، له خاصيتا الارتفاع والعرض.

**موضع الانزلاق التلقائي AutoScrollPosition:**

تسمح لك هذه الخاصية بقراءة أو تغيير موضع المنزلق، ذلك الذي انزلقه من موضع الصفر (الأعلى بالنسبة للمنزلق الرأسي، واليسار بالنسبة للمنزلق الأفقي).

**طراز الإطار FormBorderStyle:**

تتحكّم هذه الخاصية في مظهر النموذج، وهي تأخذ واحدة من القيم الموضحة في الجدول التالي:

|  |  |
| --- | --- |
| القيمة | معناها |
| بدون None | سيظهر النموذج بدون إطار وبدون شريط عنوان، بحيث لا يمكن تغيير حجمه أو تحريكه.. حاول أن تتجنب هذه القيمة. |
| متغير الحجم Sizable | وهي القيمة الافتراضية، حيث يتم عرض النموذج التقليدي، القابل للتحريك وتغيير الحجم. |
| مثبت مجسم Fixed3D | نموذج بإطار مرئي، مرتفع عن مساحة النموذج الرئيسية، ولكن لا يمكن تغيير حجمه. |
| مثبت حواري FixedDialog | نموذج يستخدم في مربعات الحوار، لا يمكن تغيير حجمه. |
| مثبت مفرد FixedSingle | نموذج لا يمكن تغيير حجمه، بإطار عبارة عن خط مفرد. |
| نافذة أدوات مثبتة FixedToolWindow | نموذج لا يمكن تغيير حجمه، بزر إغلاق فقط، وبدون زري تصغير وتكبير، وبدون الأيقونة ومربع التحكم. |
| نافذة أدوات متغيرة الحجم SizableToolWindow | مماثل للطراز السابق، لكنه قابل لتغيير حجمه. |

**مربع التحكّم ControlBox:**

يمكنك جعل هذه الخاصية "خطأ" False، لإخفاء أيقونة النموذج، وبالتالي لا يتمكن المستخدم من إظهار مربع التحكم، كما ستختفي أزار التكبير والتصغير.. ولو جعلت خاصية النص Text نصا فارغا ""، فسيختفي شريط العنوان كلّيّة!

**مراجعة الأزرار KeyPreview:**

ماذا تفعل عندما تريد تخصيص زر معين لأداء وظيفة ما، كأن يتم إغلاق البرنامج عند ضغط زر F12؟

إنّ الطبيعيَّ أن الأداةَ التي تستقبل مؤشّر الكتابة، هي الأداة التي ستستجيب لضغط هذا الزرّ.. ونظرا لأنّ النموذج يحتوي على العديد من الأدوات، فلا بد أن تكتب الكود الذي يغلق البرنامج فيها كلها!

طبعا هذا شيء سخيف وغير عمليّ، ولهذا وجدت هذه الخاصّيّة، فبجعل قيمتها "صواب"، سيستجيب النموذج أولا لضغطات أزرار لوحة المفاتيح، بغضّ النظر عن الأداة النشيطة حاليّا، حيث سيتم إطلاق الأحداث "الزر مضغوط" KeyDown و"ضغط الزر" KeyPress و"ترك الزر" KeyUp الخاصّة بالنموذج، قبل أن يتم تمريرها إلى الأداة النشيطة على النموذج.

ويمكنك أن تمنع الأدوات من استقبال هذه الأحداث ومن التأثّر بضغط الزرّ، لو كتبت في أيٍ من أحداث النموذج سالفة الذكر، الجملة التالية:

**e.Handled = True**

حيث تخبر هذه الجملة الويندوز بأنّ الزر قد تمّت الاستجابة له، ولا داعي لأداء أيّ شيء آخر، وهنا سيتصرّف الويندوز وكأنّ الزر لم يضغط أساسا!

**زر التصغير، وزر التكبير MinimizeBox, MaximizeBox:**

اجعل أيا من هاتين الخاصيتين خطأ، لإيقاف عرض زر تصغير النموذج أو زر تكبيره.

**عرض في شريط المهامّ ShowInTaskbar:**

إذا جعلت هذه الخاصيّة True، فستظهر اسم النموذج في شريط مهامّ الويندوز (في الغالب يكون في الجزء السفليّ من الشاشة).

**أصغر حجم وأكبر حجم MinimumSize, MaximumSize:**

استخدم هاتين الخاصيتين لمنع المستخدم من تصغير النموذج أو تكبيره عن حد معيّن.

وهاتان الخاصيتان كائنان، يمكن تغيير قيمهما من الكود بإحدى الطريقتين التاليتين:

* **Me.MinimumSize = New Size(400, 300)**
* **Me.MinimumSize.Width = 400**

**Me.MinimumSize.Height = 300**

ويجب أن تراعي أن ارتفاع شريط العنوان داخل في أقل ارتفاع للنموذج، لهذا يجب أن تستخدم جملة كالتالية عند جعل أقل ارتفاع للنموذج 300 مثلا:

**MinimumSize = New Size(400, 300 + SystemInformation.CaptionHeight)**

**حالة النافذة WindowState:**

يمكنك بهذه الخاصيّة قراءة أو تغيير حالة النافذة.. ولهذه الخاصيّة 3 قيم: مكبّرة Maximized ومصغّرة Minimized وعاديّة Normal.. والحالة الأخيرة تعني عرض النموذج بالأبعاد التي تحدّدها خاصيتا العرض Width والارتفاع Height والقمّة Top واليسار Left.

**طراز علامة تغيير الحجم SizeGripStyle:**

تمكنك هذه الخاصية من عرض علامة في ركن النموذج السفليّ الأيمن، لتدل على قابلية النموذج لتغيير حجمه.. وهي تأخذ واحدة من هذه القيم:

|  |  |
| --- | --- |
| القيمة | التأثير |
| تلقائي Auto | علامة تغيير الحجم ستظهر عند الاحتياج إليها.. وهي القيمة الافتراضيّة. |
| عرض Show | سيتم عرض علامة تغيير الحجم دائما. |
| إخفاء Hide | علامة تغيير الحجم لا تعرض على الإطلاق.. ولكن هذا لا يمنع تغيير حجم النموذج بالفأرة كالمعتاد. |

**موضع البداية StartPosition:**

تحدد هذه الخاصيّة موضع النموذج عندما يتم عرضه للمرّة الأولى.. وهي تأخذ القيم التالية:

|  |  |
| --- | --- |
| القيمة | تأثيرها |
| مركز محتويه CenterParent | سيتم توسيط النموذج حول مركز النافذة التي تحتويه. |
| مركز الشاشة CenterScreen | سيتم توسيط النموذج حول مركز الشاشة. |
| يدوي Manual | سيعرض النموذج تبعا لموضعه وحجمه كما تحددهما أنت. |
| حدود الويندوز الافتراضيّة WindowsDefaultBounds | سيعرض النموذج تبعا للموضع والحجم الافتراضيين كما يحددهما الويندوز. |
| موضع الويندوز الافتراضي WindowsDefaultLocation | سيتم عرض النموذج تبعا للموضع الافتراضيّ كما يحدده الويندوز، ولكن بالحجم الذي حدّدته أنت. |

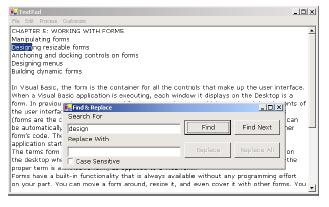
ما يجب أن تلاحظه هنا، هو أنّ استخدامك لأيّ قيمة غير القيمة Manual سيؤديّ لتجاهل الإحداثيّات التي وضعتها للنموذج ـ حتّى لو كانت من الكود ـ عند عرض النموذج لأوّل مرّة (سواء باستخدام الوسيلة Show أو الوسيلة ShowDialog أو جعل خاصيّة Visible = True)، ولكن بعد ذلك يمكنك تغيير موضع النموذج وحجمه من الكود بطريقة عاديّة، دون أن يكون لخاصيّة StartPosition أيّ تأثير.

**القمّة واليسار والعرض والارتفاع Top, Left, Width, Height:**

استخدم هذه الخصائص لتحديد موضع الحافة العليا والحافة اليسرى للنموذج وعرضه وارتفاعه.. وبهذا تكون قد وصفت حجم النموذج وموضعه بطريقة كاملة.

**يعلو الجميع TopMost:**

لو جعلت هذه الخاصية "صواب"، فستجعل هذا النموذج يعلو باقي نماذج تطبيقك، بحيث يظل مرئيا بكامله، حتّى لو نشّطت النماذج الأخرى بضغطها بالفأرة، مثلما يحدث مع نموذج البحث والاستبدال.



**تثبيت الهامش وانطباق الحافّة Anchoring and Docking:**

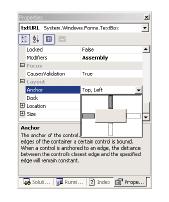
عندما تصمّم نموذجا قابلا لأن يغيّر المستخدم حجمه، تواجهك مشكلة خطيرة: كيف ستبدو الأدوات عندما يتغير حجم النموذج؟.. هل سيختفي بعضها؟.. هل ستبدو منفّرةً وغير متناسقة مع الأبعاد الجديدة للنموذج؟

لهذا كان عليك في الماضي أن تكتب بعض الكود لمعالجة هذه المشكلة، بتعديل موضع الأدوات كلما تغيّر حجم النموذج، وذلك باستخدام الحدث "تغيّر الحجم" Resize Event.

ولكن أبشر: إنّ VB.NET يقدّم لك خاصيتين جديدتين موجودتين في معظم الأدوات، لتريحاك من هذا العناء.. هاتان الخاصيتان هما: "تثبيت الهامش" Anchor و"انطباق الحافّة" Dock.

وتتيح لك خاصيّة Anchor، تثبيت واحد أو أكثر من هوامش الأداة بالنسبة لحافّة النموذج المقابلة لها، بحيث يظل هذا الهامش ثابتا، مهما غيّر المستخدم من مساحة النموذج، حيث تتغيّر أبعاد الأداة تلقائيّا، للمحافظة على هذا الهامش.

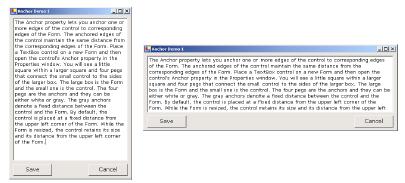
جرّب وضع مربع نص على نموذج جديد، وابحث عن خاصّية "تثبيت الهامش" في نافذة الخصائص.. اضغط زر الإسدال في خانة القيمة.. ستظهر لك الأداة كمربّع رماديّ، يتصل بحوافّ النموذج بمستطيلات رفيعة كما في الشكل التالي.



والآن كل ما عليك فعله، هو أن تضغط الوصلة التي تريد تثبيت الهامش من جهتها، حيث ستكتسب لونا رماديا دليلا على تثبيتها، ولو ضغطتها مرة أخرى، فسيزول التثبيت، وستعود للون الأبيض.

وفي الوضع التلقائيّ، ستجد أن الأداة مثبّتة بالنسبة للهامشين العلويّ والأيسر.

والآن جرّب أوضاعا مختلفة لتثبيت الهوامش، ثم غيّر أبعاد النموذج بالفأرة.. لن تحتاجَ لتشغيل البرنامج، فسيمكنك أن ترى النتيجة في وضع التصميم.



عامّة يمكنك تجريب ذلك في المشروع الذي يحمل اسم هذه الخاصّيّة في مجلد أمثلة هذا الفصل.

لدينا أيضا خاصّيّة "انطباق الحافّة" Dock، والتي تمكّنك من جعل أيّ من حوافّ الأداة منطبقة على حافّة النموذج المقابلة.. إنّ القيمة الافتراضيّة لهذه الخاصّيّة هي "ولا واحدة" None.

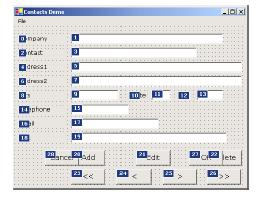
تعالَ نجرّب هذه الخاصّيّة على مربع نص على نموذج جديد.. اضغط زر الإسدال لهذه الخاصية.. ستظهر لك مجموعة من المربعات، يمكنك أن تضغط منها ما يناسبك.. فمثلا لو ضغطت المربع الموجود في المركز، فستنطبق كل حواف الأداة على حواف النموذج، بمعنى أنّ الأداة ستحتل كل مساحة النموذج.

جرّب القيم المختلفة لهذه الخاصّيّة (حيث سيظهر لك تأثيرها أيضا في وقت التصميم)، كما أنّ لديك مشروعا يحمل اسم هذه الخاصّيّة في مجلّد أمثلة هذا الفصل.

**تحديد ترتيب الانتقال بين الأدوات Setting the TabOrder:**

طبعا ستقول في نفسك: ما له سيكرر ما قاله سابقا؟.. ألم نتعرف على خاصيتيTabIndex و TabStop من قبل، لتحديد ترتيب الانتقال بين الأدوات عند ضغط Tab أو Shift+Tab؟

معك حقّ، ولكن ما زال هناك ما هو جديد: دعنا من هاتين الخاصيتين الآن، وتعال نتعرّف على طريقة مرئيّة سلسة، تمكّنك من تحديد هذا الترتيب بالفأرة: اضغط "عرض" / "ترتيب الانتقال" View.Tab Order.. ستجد أن رقم كل أداة في الترتيب قد كتب عليها.



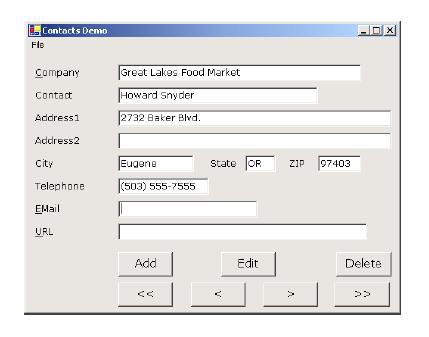
الآن وبمنتهى البساطة، اضغط الأدوات بالفأرة بالترتيب الذي تريده أن يضحي ترتيب التنقل بينها.. افعل ذلك ولا تتوقف قبل أن تنتهي من كل الأدوات.. لا تستثنِ شيئًا، حتّى اللافتات (رغمَ أنّها لا يُنتقلُ إليها، لأنّها لا تسمح للمستخدم بإدخال بيانات).

بعد أن تنتهي من فعل ذلك، اضغط "عرض" / "ترتيب الانتقال" View.Tab Order من جديد للعودة إلى تصميم النموذج التقليديّ.

**أزرار الوصول Access Keys:**

الآنَ نريد تخصيص بعض الأزرار من لوحة المفاتيح لضغط زر ما، أو الوصول مباشرةً إلى مربع نصٍّ ما، بمجرد ضغط زر Alt مع واحد من هذه الأزرار.

افترض أنّ لدينا نموذجا كالتالي:



نريد الآن إضافة حرف وصول لخانة "الشركة" Company.. كل ما عليك فعله، هو وضع العلامة "&" قبل الحرف الذي تريده، في خاصية "نص" Text الخاصة بلافتة الشركة.. فليكن هذا الحرف هو "C".. إذن يجب أن تكتب نص اللافتة كالتالي: "&Company".. ستجد أن حرف "C" في اللافتة قد كتب وتحته خط "Company"، معنى هذا أنّه حرف وصول، وأنّ ضغط Alt+C من لوحة المفاتيح، سيؤدّي للانتقال إلى هذا الحقل.

ولكن كيف؟.. ألم نَقُلْ إنّ اللافتات لا يتم الانتقال إليها؟!

نعم، ولهذا سيتم الانتقال إلى الأداة التالية في ترتيب الانتقال Tab Order، وهي هنا مربع النص الخاص بإدخال اسم الشركة.. عرفتَ الآنَ لماذا حرصنا على وضع اللافتات في ترتيب الانتقال، رغم أنّها لا يُنتقلُ إليها في حدّ ذاتها؟

ولا يوجد ما يمنع جعل حرف الوصول الخاص بخانة الشركة أي حرف آخر، مثل "o": C&ompany، أو "p": Com&pany... إلخ.

يمكن كذلك تخصيص حروف وصول للأزرار، مثل زر "Add": &Add، حيث سيتم ضغط الزر وتنفيذ وظيفته، بمجرّد ضغط Alt+A من لوحة المفاتيح.

**ملاحظة:**

ماذا لو أردت عرض الرمز "&" نفسه في اللافتة؟.. إنّ هذا الرمز هو اختصار كلمة And، وستحتاج لكتابته، كأن تريد أن تكتب مثلا "Tom & Jerry".. طبعا لو كتبت هذا النص في لافتة، فسيظهر لك كالتالي: "Tom \_Jerry"!

لا بأس: كل ما ستفعله هو أن تضيف رمز "&" زائد "Tom && Jerry"، وسيعرف VB أنّك تريد عرض الرمز وليس حرف الوصول.. هذا ما ستعرضه اللافتة في الحالة الأخيرة: "Tom & Jerry".

**نماذج رائعة:**

من الميزات الجديدة الرائعة التي يمنحها لك VB.Net قدرتك على تغيير شفافية النموذج وقدرتك على تغيير شكله.. وفي هذا الصدد لديك الخاصيّتان التاليتان:

**درجة الإعتام Opacity:**

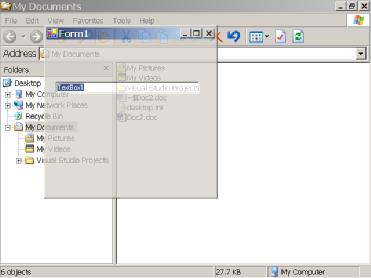
هذا الخاصيّة تأخذ نسبةً مئويّة، كلّما قلّت زادت شفافية النموذج، حتّى إنّك إذا جعلت هذه الخاصيّة صفرا، فلن ترى سطح النموذج على الإطلاق!.. وبزيادة هذه الخاصيّة، تقلّ شفافية النموذج.. وفي الوضع التلقائي، تكون قيمة هذه الخاصيّة 1 (100%)، بحيث يكون النموذج معتما تماما (غير شفّاف).

لاحظ أنّ هذه الخاصيّة تؤثّر على النموذج ككلّ، بما في ذلك شريط عنوانه والأدوات التي عليه.

جرّب أن تجعل هذه الخاصيّة في نافذة الخصائص صفرا وشغّل التطبيق.. لن ترى أيّ شيء، رغم أنّ النموذج معروض أمامك.. ولكي تتأكّد من ذلك، اضغط زر Alt ثمّ مسطرة المسافات.. ستجد أنّ قائمة التحكّم لهذا النموذج قد ظهرت.. اضغط Close لإغلاق هذا النموذج الخفيّ.

حاول تجريب نسبٍ مختلفةٍ لهذه الخاصيّة.

والصورة التالية تريك نموذجا درجة إعتامه 75%:

****

**مفتاح الشفافية TransparencyKey:**

هذه هي الخاصيّة الرائعة الثانية، وهي تسمح لك بتحديد لون على النموذج، يتمّ إزالة أيّ منطقة ملوّنة به من سطح النموذج.. إنّ روعة هذه الخاصيّة تتمثّلُ في أنّها تمكّنك من إعطاء النموذج أيّ شكل تريد.. ولكي تفعل ذلك، اتبع الخطوات التالية:

1- غيّر خاصية "لون الخلفيّة" BackColor للنموذج إلى أيّ لون ـ وليكن الأحمر.

2- اجعل قيمة "مفتاح الشفافية" TransparencyKey نفس لون خلفيّة النموذج ـ أحمر في حالتنا هذه.. الآن لن يظهر أيّ جزء من النموذج.

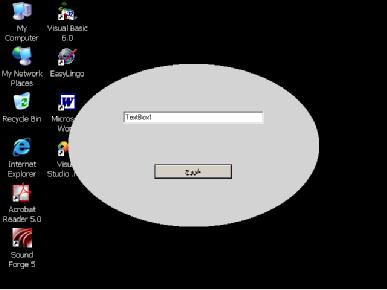
3- إذا لم تكن تريد أن تعرض شريط عنوان النموذج، فغيّر خاصيّة FormBorderStyle إلى None.

4- الآن لو وضعت أيّ أداة وجعلت لونها أيّ لون غير الأحمر، فستظهر بمفردها على الشاشة، وكأنّها بلا نموذج!

5- لتصميم نماذج بأشكال عجيبة، استخدم دوالّ الرسم ـ التي سنتعرّف عليها في فصل الرسم والتلوين ـ لرسم الشكل الذي تريده على النموذج، مع تلوينه بأيّ لون غير الأحمر.

6- أمّا لو شئت أن تكوّن نموذجا به ثقوب، فاعكس الخطوات السابقة.. بمعنى أن تجعل لون الشكل المرسوم أحمر حتّى تتمّ إزالته من النموذج، وتجعل لون خلفيّة النموذج أيّ لون غير الأحمر.

وفي مجلّد هذا الفصل، ستجد التطبيق EllipseForm، وهو يعرض لك نموذجا بيضاويّا، كما في الصورة التالية:



إن الكود الذي يفعل ذلك قد كتبناه في حدث إعادة رسم النموذج، ولن نشرحه هنا، فهو متعلّق بفصل الرسم والتلوين.. عامّةً هذا هو:

**Private Sub Form1\_Paint(ByVal sender As Object, ByVal e As \_**

**PaintEventArgs) Handles MyBase.Paint**

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

**G.Clear(Color.Red)**

**G.FillEllipse(Brushes.LightGray, 0, 0, Me.Width, Me.Height)**

**End Sub**

إنّك ستجد في معظم الأحيان أن ترك شريط العنوان سيفسد شكل النموذج، لذلك جعلنا خاصيّة FormBorderStyle بالقيمة None.. إذن كيف سيتم تحريك هذا النموذج؟

هنا سنضطرّ لكتابة الكود الذي يفعل ذلك.. والفكرة هي أن نسمح للمستخدم بسحب النموذج من أيّ نقطة عليه، وذلك بضغط زرّ الفأرة الأيسر على النموذج والتحرّك بها.. هنا يجب أن نستخدم الحدثين MouseDown و MousMove.. إنّ المعامل e لأحداث الفأرة يمنحنا الخصائص التالية:

**الزر Button:**

تسمح لنا هذه الخاصيّة بمعرفة زرّ الفأرة الذي تمّ ضغطه.

**الضغطات Clicks:**

تخبرك بعدد الضغطات التي ضغطها المستخدم.

**X و Y:**

إحداثيّا النقطة التي يوجد عندها مؤشّر الفأرة.

ولكي نحرّك النموذج، يجب أن نعرف النقطة التي بدأت عندها الحركة، حتّى نحرّك النموذج بمقدار الفارق بينها وبين موضع الفأرة الحاليّ.. ولفعل هذا ابدأ بتعريف متغيّرين على مستوى النموذج، كالتالي:

**Dim X As Integer, Y As Integer**

ثمّ استخدم حدث ضغط الفأرة MouseDown لحفظ إحداثيات نقطة الضغط في هذين المتغيّرين عند ضغط زرّ الفأرة الأيسر:

**Private Sub Form1\_MouseDown(sender As Object, e As \_**

**MouseEventArgs) Handles MyBase.MouseDown**

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

**X = e.X**

**Y = e.Y**

**End If**

**End Sub**

والآن استخدم حدث تحرّك الفأرة لتحريك النموذج كالتالي:

**Private Sub Form1\_MouseMove(ByVal sender As Object, ByVal e As \_**

**MouseEventArgs) Handles MyBase.MouseMove**

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

**' إزاحة النموذج أفقيّا بمقدار الفارق الأفقيّ بين**

**' موضع الفأرة الحالي والموضع الذي ضُغطت فيه**

**Me.Left = Me.Left + e.X - X**

**' إزاحة النموذج رأسيّا بمقدار الفارق الرأسيّ بين**

**' موضع الفأرة الحالي والموضع الذي ضُغطت فيه**

**Me.Top = Me.Top + e.Y - Y**

**End If**

**End Sub**

والآن لو جرّبت ذلك، فسيمكنك تحريك النموذج بضغطه من أيّ نقطة وسحبه.. ولكن للأسف يمكن أن يرتعش النموذج بعض الشيء أثناء الحركة.. إنّ ذلك يرجع إلى أنّ حدث حركة الفأرة يُستدعى عشرات المرّات في الثانية الواحدة، ممّا يؤدّى إلى إعادة رسم النموذج أكثر من اللازم.

**برمجة النماذج:**

* [**خصائص النموذج الوظيفيّة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\2-%20برمجة%20النماذج\1-%20خصائص%20النموذج%20الوظيفيّة.htm)
* [**وسائل النموذج:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\2-%20برمجة%20النماذج\2-%20وسائل%20النموذج.htm)
* [**أحداث النموذج Form’s Events:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\2-%20برمجة%20النماذج\3-%20أحداث%20النموذج.htm)
* [**تحميل وعرض النماذج Loading and Showing Forms:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\2-%20برمجة%20النماذج\4-%20تحميل%20وعرض%20النماذج.htm)
* [**نموذج بدء التشغيل Startup Form:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\2-%20برمجة%20النماذج\5-%20نموذج%20بدء%20التشغيل.htm)
* [**التحكم في النموذج من خارجه:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\2-%20برمجة%20النماذج\6-%20التحكم%20في%20النموذج%20من%20خارجه.htm)

**خصائص النموذج الوظيفيّة:**

**الأداة النشيطة ActiveControl:**

استخدم هذه الخاصيّة لتعرف الأداة التي تستقبل أحداث لوحة المفاتيح على هذا النموذج.

**النموذج النشيط ActiveForm:**

استخدم هذه الخاصيّة لتعرف النموذج الذي يستقبل أحداث لوحة المفاتيح ويعلو باقي النماذج في تطبيقك.

**يأسر الفأرة Capture:**

تعيد هذه الخاصيّة True، إذا كان مؤشّر الفأرة فوق النموذج في هذه اللحظة.

**مستطيل النموذج ClientRectangle:**

استخدم هذه الوسيلة للحصول على كائن مستطيل Rectangle يمثّل مساحة النموذج.. ولسبب ما قد لا تظهر هذه الخاصيّة في قائمة أعضاء النموذج في محرّر الكود!!.. لكن لا تجعل يفزعك، واستخدمها بلا تردّد!

**حجم النموذج ClientSize:**

استخدم هذه الوسيلة للحصول على كائن حجم Size يمثّل أبعاد النموذج.

**حدود النموذج DesktopBounds:**

تستطيع بهذه الخاصيّة قراءة وتغيير حدود وموضع النموذج على الشاشة.

**موضع النموذج Location:**

تستطيع بهذه الخاصيّة قراءة وتغيير موضع النموذج على الشاشة.

**أزرار الفأرة MouseButtons:**

تخبرك هذه الخاصيّة بالزر المضغوط في هذه اللحظة من أزرار الفأرة.. بالطبع حينما تكون الفأرة فوق النموذج.

**موضع الفأرة MousePosition:**

تخبرك بموضع الفأرة بالنسبة لإحداثيّات الشاشة.

**أزرار التحكّم ModifierKeys:**

تخبرك بحالة الأزرار Ctrl و Shift و Alt من حيث كونها مضغوطة أم لا في هذه اللحظة.. ونظرا لأنّ أكثر من زرّ من هذه الأزرار يمكن أن يكون مضغوطا في نفس اللحظة، فإنّ عليك استخدام عملية And مع الزرّ الذي تريد التحقّق من حالته.. فمثلا، للتأكّد من أنّ واحدا على الأقلّ من الزرّين Alt و Shift مضغوطا في هذه اللحظة، استخدم الكود التالي:

**If (Me.ModifierKeys And Keys.Alt) <> 0 OrElse \_**

**(Me.ModifierKeys And Keys.Shift) <> 0 Then**

**' الكود المناسب**

**End If**

ومن الجدير بالذكر، أنّ المعامل e في حدثي لوحة لمفاتيح (KeyDown و KeyUp) يمنحك خاصيّة مناظرة لهذه الخاصيّة، هي الخاصيّة Modifiers، التي يمكنك استخدامها بنفس طريقة استخدام الخاصيّة ModifierKeys، كما سيرد ذكره في الفصل السادس.

**الأدوات Controls:**

هذه الخاصيّة عبارة عن مجموعة Collection، تحتوي كلّ الأدوات الموجودة على النموذج.

**وسائل النموذج:**

من أهمّ وسائل النموذج:

**تنشيط Activate:**

تجعل النموذج الحالي هو النموذج النشيط (الفعّال) في البرنامج، حيث يعلو باقي النماذج، ويستقبل أحداث لوحة المفاتيح.

**إغلاق Close:**

تغلق النموذج.

**التخلّص من النموذج Dispose:**

تغلق النموذج وتحرّر أيّ مساحات كان يشغلها في الذاكرة.

**معرفة الأداة الموجودة GetChildAtPoint:**

استخدم هذه الوسيلة لكي تعرف الأداة التي تقع على النموذج عند نقطة معيّنة.

**PointToClient:**

يمكنك بهذه الوسيلة تحويل نقطة من إحداثيات الشاشة إلى الإحداثيات النسبيّة للنموذج (بمعنى أن يكون ركن النموذج العلويّ الأيسر هو نقطة الصفر، وتنسب النقطة إليها).. وتماثلها الوسيلة RectangleToClient، فهي تحوّل المستطيل من إحداثيّات الشاشة إلى إحداثيّات النموذج.

**PointToScreen:**

عكس الوسيلة السابقة: تحوّل النقطة من إحداثيّات النموذج إلى إحداثيات الشاشة.. ومثلها في ذلك الوسيلة RectangleToScreen، التي تحوّل المستطيل من إحداثيّات النموذج إلى إحداثيّات الشاشة.

**إنعاش Refresh:**

تعيد رسم النموذج لإنعاشه.

**تحديد حدود النموذج SetDesktopBounds:**

تحدّد أبعاد النموذج وموضعه على الشاشة.

**تحديد موضع النموذج SetDesktopLocation:**

تحدّد موضع النموذج على الشاشة.

**إيقاف العرض SuspendLayout:**

توقف تحديث رسم النموذج على الشاشة.. استخدم هذه الوسيلة عند إضافة عدد كبير من الأدوات إلى النموذج، أو رسم رسمٍ معقّد عليه، وذلك حتّى توفّر الوقت المستهلك في تحديث النموذج أثناء الرسم.

**استكمال العرضResumeLayout :**

بعد انتهاء الرسم أو وضع الأدوات على النموذج، استخدم هذه الوسيلة للعودة إلى رسم النموذج.

كما يمتلك النموذج مجموعة من الوسائل، تبدأ أسماؤها بكلمة Reset وتكتمل باسم صفة من صفات النموذج.. هذه الوسائل هي:

**ResetBackColor** ـ **ResetCursor** ـ **ResetFont** ـ **ResetForeColor** ـ **ResetImeMode** ـ **ResetRightToLeft**.

ومهمّة هذه الوسائل، هي إعادة الخاصيّة المرتبطة بها إلى قيمتها الموضّحة في الخاصيّة التي تبدأ بكلمة Defult، مثل: **DefultBackColor**، و **DefaultFont**... إلخ.. وفي حالة عدم وجود خاصيّة مناظرة تبدأ بكلمة Default لوسيلة منها، فإنّ هذا يعني أنّ هذه الوسيلة تعيد الخاصيّة المرتبطة بها إلى قيمتها الافتراضيّة.

**ملحوظة هامّة:**

معظم الخصائص والوسائل السابقة يرثها النموذج من خليّة الأداة Control Class، التي ترثها كذلك كلّ الأدوات Controls.. معنى هذا أنّك ستقابل معظم هذه الخصائص مع جميع الأدوات الأخرى.. تذكّر هذا لأنّنا لن نشرح هذه الخصائص مع كلّ أداة.

لاحظ كذلك أنّ كثيرا من هذه الخصائص والوسائل (مثل خاصيّة Capture)، لا تظهر أسماؤها في قائمة الأعضاء التي يعرضها لك محرّر الكود عند كتابة نقطة بعد كلمة Me.. معنى هذا أنّك تحتاج لكتابة هذه الأسماء بنفسك.. أمّا لو كنت مصرّا، فيمكنك أن تجبر VB على عرض كلّ أسماء الخصائص والوسائل في قائمة الأعضاء المنسدلة.. افتح القائمة Tools واضغط الأمر Options.. في النافذة التي ستظهر لك، اضغط العنصر Text Editor، ومن عناصره الفرعيّة اختر Basic ومن عناصره الفرعيّة اختر General.. على اليسار ستجد مربّعات اختيار.. أزل العلامة من الاختيار التالي "إخفاء الأعضاء المتقدّمة" Hide Advanced Members.. إنّ هذا سيؤدّي لظهور كلّ أعضاء النموذج والأدوات في قائمة الأعضاء، وهو ما لا ننصحك بعمله حاليّا.. والأفضل بدلا من ذلك أن تنظر في خصائص ووسائل خليّة الأداة Control Class.. انسخ الجملة التالية في نافذة فهرس الاستعلام Index (يمكن عرضها من القائمة Help)، واضغط Enter:

**Control class (System.Windows.Forms), all members**

**أحداث النموذج The Form’s Events:**

للنموذج العديد من الإجراءات، أهمُّها: "تنشيط" Activated، و"إخماد" Deactivated، و"إغلاق" Closing، و"تغيير الحجم"Resize ، و"رسم" Paint.

ولكي تضيف مستجيبات لهذه الأحداث، اختر "أحداث الخليّة الرئيسيّة" Base Class Events من القائمة المنسدلة اليسرى Class Name أعلى نافذة محرّر الكود، حيث ستظهر لك كل أحداث النموذج في القائمة المنسدلة اليمنى Method Name.

**حدثا التنشيط Activated والإخمادDeactivated :**

عندما يتم عرض أكثر من نموذج على الشاشة، واحد منها فقط يمكن أن يكون نشيطا في هذه اللحظة، بحيث يستقبل التأثيرات التي يؤدّيها المستخدم من لوحة المفاتيح.

ويمكن تنشيط أي نموذج آخر في أيّ لحظة، والانتقال إليه، بضغطه بالفأرة، أو باستخدام Alt+Tab كما شرحنا من قبل.

وفي كلّ مرّة يتمّ فيها تنشيط النموذج، يتم إطلاق الحدث Activated، وفي كل مرّة يتم فيها هجر النموذج إلى نموذج آخر، يتم إطلاق الحدث Deactivated.

**حدث الإغلاقClosing :**

يحدث هذا الحدث عند إغلاق النموذج.

ويمكنك الاستفادة منه، مثلا في إظهار رسالة تذكّر المستخدم بحفظ البيانات التي أدخلها قبل إغلاق النموذج.

كما يمكنك أن تلغي إغلاق النموذج، بالسطر التالي:

**e.Cancel = True**

حيث e هو المعامل الثاني، الموجود دائما في كل الإجراءات المستجيبة للأحداث، وإن كانت خصائصه ووسائله تختلف من حدث لآخر.

هذه القطعة من الكود تعرض الرسالة التقليديّة، التي تسأل المستخدم إذا ما كان يريد حفظ التغيرات أم لا، وتمنحه اختيارين: موافق وإلغاء.

**Public Sub Form1\_Closing(ByVal sender As Object, \_**

**ByVal e As System.ComponentModel.CancelEventArgs) \_**

**Handles Form1.Closing**

**Dim Reply As MsgBoxResult**

**Reply = MsgBox("هل تريد حفظ التغييرات التي طرأت على البرنامج؟", \_**

**MsgBoxStyle.OKCancel)**

**If Reply = MsgBoxResult.Cancel Then**

**e.Cancel = True**

**End If**

**End Sub**

وهناك حدث آخر مهمّ، مرتبط بهذا الحدث، هو "تمّ إغلاقه Closed، وهو يحدث بعد إغلاق النموذج بالفعل.

**حدث تغيير الحجمResize :**

ويحدث كلّما تغيّر حجم النموذج.

انظر لهذا المثال، الذي يحافظ على النسبة 3/4 بين عرض النموذج وارتفاعه:

**Private Form1\_Resize(ByVal sender As Object, ByVal e As \_**

**System.EventArgs) Handles MyBase.Resize**

**Me.Width = (0.75 \* Me.Height)**

**End Sub**

**ملحوظة:**

يمكن الإشارة لأيّ من أعضاء النموذج في الكود ، من خلال كلمة Me، على شرط أن تكون في نفس النموذج، لأنّ كلمة Me تدلّ على الخليّة Class الذي تستخدم داخلها.

ولكن من الممكن أيضا عدم كتابة هذه الكلمة، فهي افتراضيّة.. فمثلا في المثال السابق، يمكن كتابة:

**Width = (0.75 \* Height)**

بدلا من:

**Me.Width = (0.75 \* Me.Height)**

فكلاهما سواء!

**حدث رسم النموذج Paint:**

إنّك تستخدم الويندوز، حيث يمكن عرض نموذج آخر على الشاشة في أي لحظة، ليخفيَ لنموذجك أو جزءا منه.. فماذا سيحدث إذن لو تم إغلاق النموذج الثاني؟

في هذه الحالة يجب إعادة رسم النموذج الأوّل، لعرض الجزء الذي كان النموذج الثاني يُخفيه.

هذه العمليّة تسمّى إنعاش النموذج Refresh.. وفي كل مرة تحدث هذه العمليّة، يتم استدعاء حدث رسم النموذج.

ولكن فيمَ نستخدم هذا الحدث؟

إنّ النموذج سيُحدّث ويُنعش الأدوات التي يحملها آليّا، ولكنّه لن يعيد رسم الخطوط والمنحيات التي ترسمها أنت بدوال الرسوم، لذلك يجب عليك إعادة رسمها في هذا الحدث.

وسنرى الآن مثالا شيّقا، يسمح لك بتلوين خلفيّة النموذج بألوان متدرّجة، من الأحمر (عند ركن النموذج العلويّ الأيسر) إلى الأصفر (عند ركن النموذج السفليّ الأيمن).. وستجد هذا المثال في مجلد برامج هذا الفصل باسم GradientForm.



ولكن كيف سنفعل ذلك؟.. إنّ خاصيّة لون الخلفيّة BackColor تسمح فقط بتلوين خلفيّة النموذج بلون واحد متجانس.

إذن علينا أن نرسم نحن على النموذج.

عامةً هذا هو الكود، وإن كنّا لن نقومَ هنا بشرحه، فسنناقش هذا بالتفصيل في فصل الرسم والتلوين إن شاء الله:

**Sub RepaintForm()**

**Dim GrBrush As System.Drawing.Drawing2D.LinearGradientBrush**

**GrBrush = New System.Drawing.Drawing2D.LinearGradientBrush \_**

**(New Point(0, 0), New Point(Me.width, Me.height), \_**

**Color.Red, Color.Yellow)**

**Me.CreateGraphics.FillRectangle(GrBrush, New Rectangle(0, 0, \_**

**Me.Width, Me.Height))**

**End Sub**

**ملحوظة:**

بدلا من تكرار كتابة الجملة الطويلة System.Drawing.Drawing2D، اكتب الجملة التالية في بداية الملف، قبل تعريف النموذج نفسه:

**Imports System.Drawing.Drawing2D**

بعد ذلك يمكنك استخدام كل خصائص ووسائل هذا الاسم مباشرة.

ها هو ذا الإجراء السابق بعد استخدام هذا الاختصار:

**Sub RepaintForm()**

**Dim GrBrush As LinearGradientBrush**

**GrBrush = New LinearGradientBrush (New Point(0, 0), \_**

**New Point(Me.width, Me.height), Color.Red, Color.Yellow)**

**Me.CreateGraphics.FillRectangle(GrBrush, New Rectangle(0, 0, \_**

**Me.Width, Me.Height))**

**End Sub**

طبعا لاحظت أن عمليّة التلوين معتمدة على أبعاد النموذج، لهذا يجب استدعاء هذا الإجراء كلما تغيّرت أبعاد النموذج، بالإضافة لحدث الرسم، لأنّ حدث الرسم لا يستدعى عند تصغير حجم النموذج، ولكن يستدعى عند تكبيره فقط:

**Public Sub Form1\_Paint(ByVal sender As Object, ByVal e \_**

**As System.WinForms.PaintEventArgs) Handles Form1.Paint**

**RepaintForm()**

**End Sub**

**Public Sub Form1\_Resize(ByVal sender As Object, \_**

**ByVal e As System.EventArgs) Handles Form1.Resize**

**RepaintForm()**

**End Sub**

ويمكنك أن تطلب من VB أن يستدعي حادثة الرسم، عند تغيير حجم النموذج تكبيرا أو تصغيرا، باستخدام جملة كالتالية في حدث تحميل النمموذج:

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

**تحميل وعرض النماذج Loading and Showing Forms:**

كما ذكرنا من قبل، نحتاج لتعريف متغيّر من نوع النموذج، للتعامل معه من أي نموذج آخر (راجع الفصل الثاني).

وتعرفّنا كذلك على الوسيلة "عرض" Show، لعرض النموذج على الشاشة.. ولكن لا بدّ أنّك لاحظت أنّها تعرضُ النموذج حرًّا Modeless، حيث يمكنك التنقّل بينَه وبينَ أي نموذج آخر في برنامجك.

ولكنَّ هناك حالات تحتاج فيها إلى عرض نموذج إجباريّ Modal، مثل نموذج رسائل الخطإ، حيث لا يمكن للمستخدم استكمال تعامله مع البرنامج إلا بإغلاق هذا النموذج أولا، بضغط زر موافق أو إلغاء أو ما شابه.

لفعل هذا، يمكنك استخدام الوسيلة "عرض حواري" ShowDialog:

**Dim Frm As New Form2**

**Frm.ShowDialog**

ما يجب أن تلاحظه هنا، هو أن تنفيذ وسيلة العرض الحواري، يوقف تنفيذ البرنامج إلى أن يتم إغلاق النموذج الإجباريّ، حيث يتمّ بعدها تنفيذ السطور التالية لجملة ShowDialog.

ويمكن إغلاق النموذج بإحدى الطرق التالية:

1- "غلق" Close: وهي تُنهي النموذج تمامًا، بحيث تحتاج لتحميله من جديد للتعامل مع وسائله وأدواته.

2- "إخفاء" Hide: وهي تخفي النموذج من على الشاشة فقط، بجعل خاصية "مرئي" Visible خطأ، ولكن يظلّ بإمكانك التعامل مع النموذج وما عليه من أدوات، فهو ما زال محملا في الذاكرة.. وننصحك باستخدام هذه الوسيلة مع النماذج التي يتم عرضها كثيرا في البرنامج، وذلك لتوفير الوقت الذي تحتاجه عملية التحميل، كما أنّ حالة النموذج عند إعادة عرضه، ستكون كما تركها المستخدم قبل إخفائه، فمثلا ستكون مربعات النصوص ما زالت محتفظة بنصوصها، أما عند إغلاق النموذج وإعادة تحميله، فإنّ النموذج يظهر في حالته الابتدائيّة كما صمّمتَها أنت.

3- إذا كان النموذج قد تم عرضه كمربع حوار باستخدام الوسيلة ShowDialog، فإن وضع أي قيمة في خاصية "نتيجة الحوار" DialogResult يؤدّي إلى إغلاق النموذج:

**Me.DialogResult = DialogResult.OK**

حيث يمكن استخدام قيمة هذه الخاصّيّة من البرنامج الذي عرض النموذج كالتالي:

**If Frm.ShowDialog = DialogResult.OK Then Exit Sub**

حيث تقوم دالة ShowDialog بعرض النموذج، وعند غلقه تعيد القيمة التي تم وضعها في خاصية "نتيجة الحوار" DialogResult.

**ملحوظة:**

لو جعلت أحد أزرار مربع الحوار هو زر قبول النموذج AcceptButton، فسيؤدّى ضغطه لوضع قيمة خاصيّة "نتيجة الحوار" DialogResult بـ DialogResult.OK وسيتم إغلاق مربع الحوار تلقائيّا، دونَ أن تكتب أي كود.. نفس الأمر سيتمّ لو جعلتَ زرا آخر هو زر إلغاء النموذج CancelButton، ولكن في هذه الحالة سيتم وضع قيمة DialogResult بـ DialogResult.Cancel.. وهذا يقودنا لأن نذكرَ أنّ خاصّيّة DialogResult هي إحدى خصائص الأزرار أيضا، وتغيير قيمتها لأيّ زرّ يؤدّي لإغلاق النموذج المعروض كمربع حوار، وإرجاع قيمتها لدالة ShowDialog:

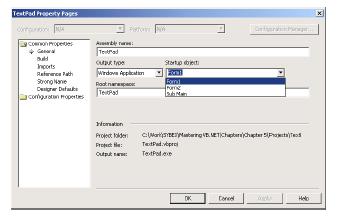
**BtOk.DialogResult = DialogResult.OK**

ولكي تعرف إذا ما كان النموذج الحالي قد تمّ عرضه كمربّع حوار أم لا، استخدم خاصيّة "حواري" Modal:

**If Me.Modal = True Then MsgBox("هذا مربّع حوار")**

**نموذج بدء التشغيل Startup Form:**

ذكرنا مرارا من قبل، أنّك تستطيع اختيار نموذج بدء التشغيل، بضغط اسم المشروع في متصفّح المشاريع Solution Explorer بزر الفأرة الأيمن، وضغط خصائص Properties من القائمة الموضعيّة، ليظهر لك مربع حوار خصائص المشروع، حيث يمكّنك من اختيار نموذج بدء التشغيل.



ما يهمّنا هنا، هي الحالة التي تريد أن تبدأ فيها برنامجك بإجراء، وليس بنموذج.

أضف قالبا Module لبرنامجك، وأسمِه StartUp.. أضف للقالب إجراءً وسمّه Main.. في هذا الإجراء يمكنك أن تكتب كل الكود الذي تريد تنفيذه عند تحميل البرنامج، ومن خلالِه يمكنك عرض النماذج التي تريد.

**Module StartUpModule**

**Sub Main()**

**System.Windows.Forms.Application.Run(New Form1())**

**End Sub**

**End Module**

والآن افتح مربع حوار خصائص المشروع، واجعل كائن بدء التشغيل، الإجراء الفرعيّ Main.

**التحكم في النموذج من خارجه**

حينما يكون لديك في التطبيق أكثر من نموذج، فلا ريب أنّك قد تحتاج للإشارة إلى نموذج في الكود من خلال نموذج آخر.. هنا تعرف كيف:

* [**المتغيرات المشتركة بين النماذج Shared Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\6-%20التحكم%20في%20النموذج%20من%20خارجه\1-%20المتغيرات%20المشتركة%20بين%20النماذج.htm)
* [**مشروع النماذج المتعدّدة MultipleForms Project:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\6-%20التحكم%20في%20النموذج%20من%20خارجه\2-%20مشروع%20النماذج%20المتعدّدة.htm)

**المتغيرات المشتركة بين النماذج Sharing Variables between Forms:**

قلنا من قبل إنّ تعريف متغير على مستوى النموذج (خارج أي إجراء)، مع استخدام المجال العام Public يجعله متاحا للاستخدام في كلّ المشروع.

فمثلا عند تعريف متغير في النموذج Form1 كالتالي:

**Public X As Integer**

فإن بإمكانك استخدامه من نموذج آخر Form2 كالتالي:

**Dim F1 As New Form1**

**F1.X = 5**

حيث يعتبر المتغير X حقلا Field من حقول النموذج.

ولكن هناك ما يجب أن تلاحظه في هذه الطريقة: إنّ X خاص فقط بالنسخة التي ينتمي إليها من النموذج، وأنت تعرف أنّ بإمكانك أن تنشئَ أكثر من نسخة من نفس النموذج، تتصرّف كل منها مستقلّةً عن الأُخريات:

**Dim F1 As New Form1**

**F1.X = 5**

**Dim F2 As New Form1**

**F2.X = 7**

**MsgBox (F1.X) ' تعرض الرسالة 5**

**MsgBox (F2.X) ' تعرض الرسالة 7**

إذن كيف يمكن تعريف متغيّر مشترك بين كل نسخ النموذج مهما مكان عددها؟

بسيطة: استخدم المجال "عام مشترك" Public Sharedعند تعريف المتغيّر:

**Public Shared X As Integer**

ويمكنك استخدام هذا المتغيّر من خارج النموذج كالتالي:

**Dim F1 As New Form1**

**F1.X = 5**

**Dim F2 As New Form1**

**F2.X = 7**

**MsgBox (F1.X) ' تعرض الرسالة 7**

تلاحظ طبعا أن تغيير قيمة هذا المتغيّر من أي نسخة من نسخة النموذج، يؤثّر في قيمته التي تراها كل النسخ، وذلك ببساطة لأنّ هذا المتغيّر يتم حجزه في الذاكرة مرة واحدة، وأي نسخة يتم إنشاؤها من النموذج يمكنها أن تتعامل معه.

ويمكن تطبيق نفس الطريقة على الأدوات الموجودة على النموذج، فبخلاف الطريقة المباشرة:

**Dim F1 As New Form1**

**F1.TextBox1.Text =""**

يمكن تعريف متغير عام مشترك من النوع مربع نص، ليصير مشتركا بين كل نسخ النموذج:

**Public Shared TBox As TextBox**

ويمكنك أن تضع فيه مرجعا لمربع النص الذي تريده، بكتابة الجملة التالية في حدث تحميل النموذج Load:

**TBox = TextBox1**

بعد ذلك يمكنك التعامل مع مربع النص من أي نسخة كالتالي:

**Dim F1 As New Form1**

**F1.TBox.Text = ""**

أو التعامل معه مباشرةً من النموذج الأصليّ نفسه:

**Fom1.TBox.Text = ""**

**حيلة مفيدة:**

باعتباري أحد من استخدموا VB6 لخمس سنوات متتالية، فأنا أعرف كم هو مزعج احتياج مبرمج VB.Net لتعريف متغيّر للنموذج في كلّ نموذج آخر قبل استخدامه.. ولحلّ هذه المشكلة، يمكنك اللجوء لخدعة صغيرة، وذلك إذا كنت ستستخدم نسخةً واحدةً فقط من النموذج في تطبيقك، كأن يكون النموذج هو النافذة الرئيسيّة للتطبيق، ولا يمكن عرض أكثر من نسخة منه في نفس اللحظة.. في هذه الحالة عرّف متغيّرا للنموذج، ولكن اجعله عاما مشتركا Public Shared كالتالي:

**Public Shared F1 As Form1**

وفي حدث تحميل النموذج اكتب الجملة التالية:

**F1 = Me**

الآن يمكنك في أيّ موضع من البرنامج استخدام نسخة النموذج F1 عبر اسم النموذج الأصليّ Form1، وذلك لأنّ المتغيّر F1 مشترك، ويمكن التعامل معه من خليّة النموذج الأصليّة Form1 وكذلك من كلّ نسخِها.. افترض الآن أنّ لديك نموذجا آخر اسمه Form2 وتريد من خلاله أن تعرض النموذج Form1.. استخدم الجملة التالية:

**Form1.F1.Show**

العيب الرئيسيّ في هذه الطريقة هو طول الجملة، فقد كان مستخدم VB6 معتادا على استخدام الجملة التالية مباشرة:

**Form1.Show**

ولكن ما باليد حيلة!

ولكن لماذا لا نعرّف متغيّرا آخر للنموذج Form1 على مستوى النموذج Form2 ونستريح؟.. ممكن، لكن في حالة واحدة.. لو وضعنا في هذه المتغيّر النسخة الأولي.. اكتب الجملة التالية في النموذج Form2:

**Dim F1 As Form1 = Form1.F1**

الآن في أيّ موضع في النموذج Form2 يمكنك عرض النموذج Form1 بجملة كالتالية:

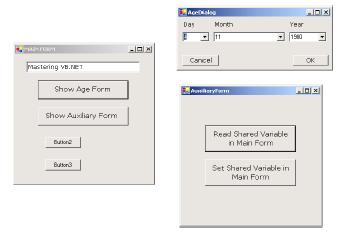
**F1.Show**

وبهذا نكون قد حقّقنا الاختصار غير المخلّ!

كما يمكن تعديل هذه الطريقة، بتعريف مجموعة من المتغيّرات عامّة مشتركة تحتوي على قيم الخصائص التي تريد التعامل معها من خارج النموذج، ولكنّ هذا سيحتاج لكتابة بعض الكود، بحيث يتمّ تعديل قيم هذا المتغيّرات كلّما تغيّرت قيم الخصائص المناظرة.. أنت بالخيار!

**مشروع النماذج المتعدّدة The MultipleForms Project:**

يتكوّن هذا المشروع من نموذج رئيسيّ، ونموذج إضافيّ AuxiliaryForm ومربع حوار.. ويحتوي النموذج الرئيسيّ على زر لإظهار النموذج الإضافيّ، وزر لإظهار مربع الحوار.



ويحتوي النموذج الرئيسيّ على تعريف المتغير المشترك StrProperty، كالتالي:

**Public Shared StrProperty As String = "Mastering VB.NET"**

ويحتوي النموذج الإضافي على زر لقراءة قيمة المتغيّر المشترك StrProperty، وكوده كالتالي:

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

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

**MsgBox(MainForm.StrProperty, MsgBoxStyle.OKOnly, \_**

**"قيمة المتغير المشترك")**

**End Sub**

كما يوجد زر لتغيير قيمة هذا المتغيّر، كوده كالتالي:

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

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

**Dim Str As String**

**Str = InputBox("أدخل قيمة جديدة للمتغير المشترك")**

**MainForm.StrProperty = Str**

**End Sub**

والدالة "مربع الإدخال" InputBoxهي نموذج يحتوي على زر موافق وزر إلغاء وربع نص، ولافتةٍ تقوم بعرض رسالة تطلب منك إدخال قيمة معيّنة في مربع النص.

ما رأيك الآن أن نطوّر وسيلة الاتصال بين النموذجين، بإطلاق حدث يخبر النموذج الرئيسيّ بتغيّر قيمة المتغيّر المشترك، حتّى يعرضها في مربع النص؟

لفعل ذلك، نحتاج أولا لتعريف الحدث.. اكتب ما يلي في قسم التعريفات من النموذج الإضافيّ:

**Event StrPropertyChanged()**

والآن يمكنك أن تطلقَ الحدث باستخدام تعبير RaiseEvent متبوعا باسم الحدث، عقب كل تغيير يحدث للمتغيّر.. لهذا سنعدّل إجراء تغيير قيمة المتغير المشترك إلى:

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

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

**Dim Str As String**

**Str = InputBox("أدخل قيمة جديدة للمتغير المشترك")**

**MainForm.StrProperty = Str**

**RaiseEvent StrPropertyChanged**

**End Sub**

الآن لم يتبقّ إلا أن نكتب الإجراء المستجيب للحدث، بالطريقة التي ألفناها سابقا.. ولكن كيف نكتب المستجيب لهذا الحدث في النموذج الرئيسيّ، واسمه ليس موجودا في أحداث النموذج؟

يجب أولا أن تعرّف المتغير المعبّر عن النموذج الإضافيّ بالطريقة التالية:

**Dim WithEvents FRM As New AuxiliaryForm()**

إن كلمة "بالأحداث" WithEvents تخبر VB أنّ هذا المتغيّر قادر على إطلاق الأحداث.. ولو نظرت الآن في القائمة المنسدلة اليسرى لمحرّر الكود، لوجدت فيها الاسم FRM.. اختره، وستجد أن الحدث StrPropertyChanged قد ظهر في القائمة المنسدلة اليمنى، ضمن باقي أحداث النموذج الإضافي التقليديّة.. اضغطه بالفأرة، وسيظهر لك تعريف المستجيب لهذا الحدث اكتب فيه التالي:

**Private Sub FRM\_StrPropertyChanged() \_**

**Handles FRM.StrPropertyChanged**

**TextBox1.Text = StrProperty**

**Beep() ' هذه الدالة تُصدر نغمة سريعة من الجهاز**

**End Sub**

ولكن ماذا عن مربع الحوار؟

في البداية سنعرّف متغيرا على مستوى النموذج الرئيسيّ لنتعامل به معه:

**Dim DLG As New AgeDialog()**

بعد هذا سنعرض مربع الحوار، مع وضع بعض القيم الافتراضية في خاناته:

**Protected Sub Button3\_Click(ByVal sender As Object, \_**

**ByVal e As System.EventArgs)**

**' تاريخ افتراضي 4/11/1980**

**DLG.cmbMonth.Text = "4"**

**DLG.cmbDay.Text = "11"**

**DLG.CmbYear.Text = "1980"**

**DLG.ShowDialog()**

**' عرض مربع الحوار**

**If DLG.DialogResult = DialogResult.OK Then**

**MsgBox(DLG.cmbDay.Text & "/" & DLG.cmbMonth.Text & "/" & DLG.cmbYear.Text)**

**Else**

**MsgBox("تم إلغاؤه")**

**End If**

**End Sub**

ويحتوي مربع الحوار على زري موافق وإلغاء.. هذا هو كودهما:

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

**ByVal e As System.EventArgs)**

**Me.DialogResult = DialogResult.OK ' هذه الجملة تغلق مربع الحوار**

**End Sub**

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

**ByVal e As System.EventArgs)**

**Me.DialogResult = DialogResult.Cancel ' هذه الجملة تغلق مربع الحوار**

**End Sub**

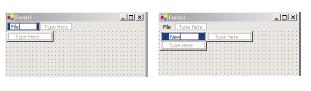
**تصميم القوائم Designing Menus**

* [**محرّر القوائم Menu Editor:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20تصميم%20قوائم%20النماذج\1-%20محرّر%20القوائم.htm)
* [**خصائص الكائن "عنصر القائمة" MenuItem Object:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20تصميم%20قوائم%20النماذج\2-%20خصائص%20عنصر%20القائمة.htm)
* [**برمجة القوائم:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20تصميم%20قوائم%20النماذج\3-%20برمجة%20القوائم.htm)
* [**إنشاء القوائم الموضعيّة Creating Context Menus:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20تصميم%20قوائم%20النماذج\4-%20إنشاء%20القوائم%20الموضعيّة.htm)

**محرّر القوائم Menu Editor:**

يمكنك أن تضيف القوائم لنموذجك عن طريق الأداة "القائمة الرئيسيّة" MainMenu، حيث يمكنك أن تضيف إليها كائنات من النوع "عناصر القائمة" MenuItem.

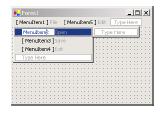
انقر بالفأرة مرتين على أيقونة القائمة الرئيسيّة MainMenu في صندوق الأدوات.. سيظهر لك هامش أسفل مصمم النماذج يسمّى "صينية المكوّنات" Components Tray، ستتم إضافة أيقونة القائمة الرئيسيّة إليه، بينما سيظهر أمر من أوامر القوائم في أعلى النموذج، مكتوب عليه "اضغط هنا" Click here.. اضغط كما يقول، واكتب عنوان القائمة ولتكن "ملف" File.. بمجرّد أن تبدأ الكتابة، سيظهر أمران جديدان من أوامر القائمة، أحدهما في مستوى الأمر الأوّل (على يساره)، والآخر فرّعيّ من الأمر الأوّل (أسفلَه).



اكتب في الأمر الفرعيّ "جديد" New.. ثمّ استمرّ في كتابة عناوين القوائم بنفس الطريقة.. بعد أن تنهي اضغط بالفأرة في أي موضع على النموذج.. ستختفي كل العناصر التي لم تُكتب بعد، وستتخذ القوائم شكلها النهائيّ.. ويمكنك العودة لإضافة المزيد من القوائم، بضغط أي عنصر موجود منها بزر الفأرة الأيمن، واختيار "إضافة عنصر جديد" Insert New.

ولكلّ عنصر في القوائم اسم، يمكن من خلاله أن تشير إليه في الكود.. ولتغيير الأسماء الافتراضيّة التي تكتسبها قوائمك، حدّد كل عنصر واضغطه بزر الفأرة الأيمن، واضغط "خصائص" Properties من القائمة الموضعيّة، حيث يمكنك تغيير خاصية Name من نافذة الخصائص.

وبطريقة أسهل، يمكنك اختيار الأمر "تحرير الأسماء" Edit Names من القائمة الموضعيّة، حيث سيسمح لك هذا الأمر بتحرير الأسماء على تركيب القوائم، تماما كأنّك تكتب عناوينها.



بعد أن تنتهي من تحرير الأسماء، اضغط الأمر "تحرير الأسماء" مرة أخرى للعودة لعرض العناوين.

ولإنشاء خط فاصل بين بعض أوامر قائمة ما، اضغط بزر الفأرة الأيمن، الأمر الذي تريد أن تعرض الفاصل قبله، واختر "إضافة فاصل" Insert Separator.. كما يمكنك أن تؤدّي نفس الغرض، بجعل عنوان الأمر إشارة الطرح "-"، حيث سيتحوّل إلى فاصل في هذه الحالة.

**خصائص الكائن "عنصر القائمة" MenuItem Object’s Properties:**

يمثل هذا الكائن، عنصر القائمة في أي مستوى: رئيسيا كان أو فرعيا.. ويحتوي هذا الكائن على الخصائص التالية:

**علامة الاختيار Checked:**

تعمل بعض القوائم كمفاتيح، يمكنها أن تكون مغلقة (بوضع علامة الاختيار)، أو مفتوحة (بإزالة علامة الاختيار) لتخبر المستخدم بقيامها بعمل ما أو عدم قيامها به.. ويمكنك أن تعكس حالة القائمة بجملة كالتالية:

**Mnu1.Checked = Not Mnu1.Checked**

**عنصر افتراضي DefaultItem:**

بجعل هذه الخاصية صواب، تصبح هذه القائمة الفرعيّة افتراضيّة، فتظهر بخط سميك، ويتم تحديدها تلقائيا عند فتح قائمتها الرئيسيّة.

**فعّالة Enabled:**

اجعل هذه الخاصيّة خطأ، حتّى تصبح هذه القائمة غير متاحة للاستعمال من قبل المستخدم، حيث ستظهر بلون رماديّ ولن يمكن ضغطها بالفأرة.

**إنّها رئيسيّة IsParent:**

هذه الخاصيّة للقراءة فقط، بمعنى أنّك لا تستطيع تغييرها.. وهي تخبرك إذا ما كانت هذه القائمة تحتوى على قوائم فرعيّة أم لا.

**حرف الوصول Mnemonic:**

هذه الخاصيّة للقراءة فقط، وهي تخبرك بحرف الوصول Access Key الذي تم وضعه للقائمة.. فإذا لم يكن للقائمة حرف اختصار، فستُرجع لك هذه الخاصيّة الحرف "0".

**مرئيّة Visible:**

بوضع هذا الخاصية بخطإ، يمكنك إخفاء هذا العنصر من القائمة.

**قائمة النوافذ MDIList:**

تستخدم هذه الخاصيّة في التطبيقات متعدّدة النوافذ، لسرد قائمة بأسماء كل النوافذ المفتوحة حاليا في التطبيق، وسنشرحها بالتفصيل في الفصل 19 إن شاء الله.

**نص Text:**

وهذه الخاصيّة غنيّة عن التعريف، وإن أحببت أن ألفت نظرك إلى أنّك تستطيع تغيير عنوان القائمة من الكود كذلك.

ويمكن تخصيص حرف وصول للقوائم أيضا، ليمكّن المستخدم من تحديد قائمة رئيسيّة بمجرّد ضغط زر Alt مع هذا الحرف.. فمثلا، يمكنك تحديد قائمة التحرير Edit في أي تطبيق ويندوز، بضغط Alt+E.. وبعد تحديد القائمة، يمكن إسدالها بضغط السهم السفليّ من لوحة المفاتيح.. بعد ذلك يمكن الانتقال للعناصر الفرعيّة بأزرار الأسهم، أو بضغط حرف الوصول الخاص بالعنصر (دون الحاجة لزر Alt).

ويمكن تخصيص زر الوصول بوضع العلامة & قبله في عنوان (نص) القائمة، تماما كما رأينا من قبل مع الأزرار واللافتات.

**ملاحظة:**

إذا لم تحدّد حرف وصول لإحدى القوائم الرئيسيّة، فسيعتبر VB أنّ أول حرف في عنوانها هو حرف الوصول، وإن كان لن يضع تحته خطّا.. وفي حالة وجود أكثر من قائمة رئيسيّة تبدأ بنفس الحرف، يمكن تحديدها على التوالي بإعادة ضغط Alt مع حرف الوصول أكثر من مرّة.

**حرف الاختصار Shortcut:**

بواسطة حرف الاختصار، يمكنك أن تنفّذ أمرا معينا من أوامر أي قائمة، بمجرّد ضغط هذا الحرف.

اختر عنصرا من أي قائمة، ومن نافذة الخصائص، أسدل القائمة المناظرة لخاصيّة "زر الاختصار" Shortcut واختر منها الاختصار الذي يناسبك.

**اعرض حرف الاختصار ShowShortCut:**

تأكّد من أنّ قيمة هذه الخاصيّة صواب، وذلك لعرض حرف الاختصار بجوار عنوان القائمة، حتّى يتمكّن المستخدم من رؤيته.

**تحذير:**

رغمّ أنّ القائمة الرئيسيّة تمتلك خاصيّة RightToLeft، إلا إنّ محاذاة القائمة الرئيسيّة من اليمين إلى اليسار قد تؤدّى إلى حدوث بعض الشعوذة في أدائها، كاختفاء بعض أو كلّ القوائم، أو حدوث لخبطة في ترتيبها، أو تغيّر هذا الترتيب عند المرور فوق القائمة بالفأرة!!.. عامّةً لا داعي لتبخير التطبيق ولا لتحضير (شمهورش)!!.. هكذا تبدأ ميكروسوفت أيّ إصدار جديد من إصداراتها، حيث ترهق المبرمجين العرب إلى أقصى حدّ للتعامل مع هذه الإصدارات، إلى أن يمطروها بمطالب التعديل والتطوير، التي سرعان ما تظهر في الإصدارات التالية وعليك خير!!

**برمجة القوائم**

* [**أحداث القوائم:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20برمجة%20القوائم\1-%20أحداث%20القوائم.htm)
* [**إنشاء القوائم القصيرة والطويلة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20برمجة%20القوائم\2-%20إنشاء%20القوائم%20القصيرة%20والطويلة.htm)
* [**القوائم المرنة Dynamic Menus:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20برمجة%20القوائم\3-%20القوائم%20المرنة.htm)
* [**التعبير عن عناصر القوائم في الكود:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20برمجة%20القوائم\5-%20التعبير%20عن%20عناصر%20القوائم%20في%20الكود.htm)
* [**مشروع خريطة القوائم MapMenu Project:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\3-%20برمجة%20القوائم\6-%20مشروع%20خريطة%20القوائم.htm)

**أحداث القوائم:**

عندما يضغط المستخدم أي أمر في القائمة، يتم تنفيذه.. فكيف إذن نكتبُ هذا الكود الذي يُنفّذ؟

ببساطة، انقر بالفأرة مرتين، ذلك الأمر الذي تريد كتابة كوده، وسيظهر لك الإجراء المستجيب لضغطه في محرّر الكود، تماما كما كنت تفعل مع الأزرار:

**Private Sub MenuItem9\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles MenuItem9.Click**

**End Sub**

فمثلا، لو لديك أمر لإغلاق البرنامج، فسيكون كوده كالتالي:

**Sub menuExit(ByVal sender As Object, ByVal e \_**

**As System.EventArgs) Handles menuExit.Click**

**End**

**End Sub**

ويمكنك أن تتعامل مع المستجيب لحدث ضغط القائمة، تماما كما كنت تتعامل مع المستجيب لضغط الزر، فمثلا يمكنك أن تجعل إجراء واحدا يستجيب لضغط أكثر من قائمة كالتالي:

**Private Sub Zoom200\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Zoom200.Click, \_**

**Zoom100.Click, Zoom75.Click**

**End Sub**

هذا مع الاستفادة من المعامل sender لمعرفة القائمة التي تم ضغطها (ولا تنس تحويل هذا المعامل إلى عنصر قائمة MenuItem:

**Console.WriteLine(CType(sender, MenuItem).Text)**

وهناك حدث آخر مهمّ من أحداث القائمة، ذلك هو حدث التحديد Select، وهو يحدث عندما يضغط المستخدم قائمة الرئيسيّة (مثل "ملفّ" أو "تحرير").. ويمكنك الاستفادة من هذا الحدث، عندما تريد إبطال بعض أوامر القائمة.. فمثلا لو لديك قائمة "تحرير" وبها أمر "تراجع".. في هذه الحالة يجب أن تتأكّد من أنّ هناك عمليّات تراجع يمكن القيام بها، وذلك في حدث تحديد Select الخاصّ بالقائمة "تحرير" ، وتبعا لذلك تجعل القائمة "تراجع" فعّالة أو غير فعّالة:

**Private Sub Edit\_Select(ByVal sender As Object, ByVal e As \_**

**System.EventArgs) Handles Edit.Select**

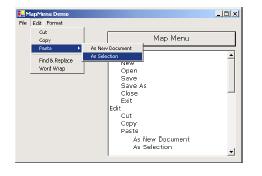
**EditUndo.Enabled = TextBox1.CanUndo**

**End Sub**

وهناك حدث ثالث هامّ ذلك هو حدث إسدال القائمة Popup، وهو يحدث مباشرةً قبل إنسدال القائمة لتعرض أوامرها.

**مشروع خريطة القوائم The MapMenu Project:**

هذا المشروع موجود في مجلد أمثلة البرنامج، وهو يريك كيف يمكن كتابة أسماء جميع عناصر القائمة الرئيسيّة، باستخدام الجمل التكراريّة.



هذا هو الكود الذي يتم تنفيذه عند ضغط الزر:

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

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

**Dim itm As MenuItem**

**For Each itm In Me.Menu.MenuItems**

**Console.WriteLine(itm.Text)**

**PrintSubMenu(itm)**

**Next**

**End Sub**

إنّ جملة التكرار "لكل عنصر في" For Each.. Next جديدة نوعا، وهي تقوم بوظيفة جملة "من إلى" العاديّة، ولكنّها لا تأخذ بداية ونهاية.. إنّ هذه الجملة خاصّة بالمجموعات Collections، تلك الكائنات التي يحتوي كلّ منها على مجموعة من العناصر من نوع ما، ويوضع اسم هذه المجموعة بعد كلمة In في رأس جملة "لكل عنصر في".

وتقوم جملة "لكل عنصر في" بالمرور على عناصر المجموعة واحدا بعد الآخر، وفي كل مرّة، يتم وضع هذا العنصر في متغيّر الجملة التكراريّة (ذلك الذي يلي كلمة Each)، والذي بالطبع يجب أن يكون من نوع عناصر المجموعة.

وفي برنامجنا هذا، تقوم جملة التكرار بالمرور على جميع العناصر الموجودة في مجموعة عناصر القائمة الرئيسيّة للنموذج الحالي (وهي ملف، تحرير،.... إلخ)، وكتابة عنوان كل عنصر في نافذة المخرجات، ثم استدعاء الإجراء PrintSubMenu، لطباعة أسماء عناصر القائمة الفرعيّة لهذا العنصر.

والآن لك أن تتخيّل ما سيفعله الإجراء PrintSubMenu:

إنّه يمرّ عبر كل عناصر المجموعة المرسلة إليه لطباعة أسمائها.

ولكن ماذا لو كان أحد عناصر هذه المجموعة يحتوي على عناصر فرعيّة؟

يمكن اختبار المجموعة MenuItems الخاصة بهذا العنصر، فإذا كان عدد العناصر الموجودة بها أكبر من صفر itm.MenuItems.Count > 0، إذن يجب طباعة هذه العناصر فورا.

ولكن كيف؟

بسيطة: استدع الإجراء PrintSubMenu مرّة أخرى، ولكن هذه المرّة أرسل له العنصر الحالي كمعامل:

**Sub PrintSubMenu(ByVal MItem As MenuItem)**

**Dim itm As New MenuItem()**

**For Each itm In MItem.MenuItems**

**Console.WriteLine(itm.Text)**

**If itm.MenuItems.Count > 0 Then PrintSubMenu(itm)**

**Next**

**End Sub**

لاحظ أنّ الإجراء PrintSubMenu يستدعي نفسه من جملة تكراريّة، وفي كل نسخة جديدة تُستدعى، يتم تنفيذ جملة تكراريّة جديدة، وفي كل لفّة منها يمكن أن يستدعي الإجراء نفسه، و.. و... و.... !!

جرّب أن تُضيف أي عدد من القوائم للنموذج.. وبأيّ عمق.. ستجد أن البرنامج سيحصيها كلها.

والآن حتّى لو كنت لا تفقه جيّدا في الإجراءات الارتداديّة Recursive Procedures، فيمكنك أن تستخدم الطريقة العامّة التالية في أيّ تطبيق:

ابدأ بتعريف مندوب Delegate عن أيّ إجراء تريد تنفيذه على القوائم:

**Delegate Sub MenuAction (ByVal MItem As MenuItem)**

والآن اكتب الإجراء الارتداديّ العامّ الذي يمرّ عبر كلّ عناصر القائمة الرئيسيّة المرسلة له كمعامل، وأرسل له معاملا آخر، هو الحدث الذي تريد تنفيذه على هذه العناصر:

**Sub ScanSubMenu(ByVal MItem As MenuItem, Action As MenuAction)**

**Dim itm As New MenuItem()**

**For Each itm In MItem.MenuItems**

**Call Action(itm)**

**If itm.MenuItems.Count > 0 Then PrintSubMenu(itm)**

**Next**

**End Sub**

الآن يمكنك استخدام هذا الإجراء في أيّ تطبيق.. كلّ ما عليك هو أن تكتب إجراءً فرعيّا أو أكثر، لكلّ منها معامل واحد من النوع MenuItem، بحيث يمكن إرسالها كمعامل للإجراء ScanSubMenu.. وفي هذه الإجراءات يمكنك أن تؤدّي أيّ عمليّة على عناصر المصفوفة.. فمثلا لطباعة عناصر المصفوفة، استخدم الإجراء التالي:

**Sub PrintMenu(ByVal MItem As MenuItem)**

**Console.WriteLine(MItem.Text)**

**End** **Sub**

ولاستدعاء هذا الإجراء لكلّ عناصر القائمة File استخدم الجملة التالية:

**ScanSubMenu(FileMenu, AddressOf PrintMenu)**

ولتطبيق ذلك على كلّ القوائم، استخدم جملة تكراريّة للمرور عبر كلّ القوائم الرئيسيّة كالتالي:

**Dim itm As MenuItem**

**For Each itm In Me.Menu.MenuItems**

**PrintMenu(itm)**

**ScanSubMenu(itm, AddressOf PrintMenu)**

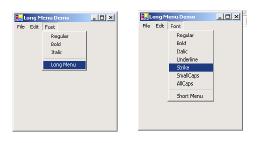
**Next**

لاحظ أنّنا استدعينا الإجراء PrintMenu بطرية عاديّة أولا لطباعة القائمة الرئيسيّة، ثمّ بعد ذلك أرسلناه كمعامل للإجراء ScanSubMenu لطباعة القوائم الفرعيّة.

وفي إمكانك كتابة أيّ إجراءات أخرى غير PrintMenu واستدعائها بنفس الطريقة.

**إنشاء القوائم القصيرة والطويلة:**

إذا كانت إحدى القوائم تحتوى على عدد كبير جدا من الأوامر، فيمكنك أن تعرض أكثر الأوامر استخداما، على أن يكون آخر أمرٍ في هذه القائمة القصيرة هو الأمر "اعرض القائمة الطويلة" Long Menu، والذي عند ضغطه، يتم عرض القائمة الكاملة "الطويلة".. أيضا سيكون آخر أمرٍ في هذه القائمة هو "اعرض القائمة القصيرة" Short Menu.



وستجد المشروع الذي يوضّح ذلك في مجلّد أمثلة هذا الفصل.

والجدول لتالي يوضّح عناصر القائمة الطويلة:

|  |  |
| --- | --- |
| اسم الأمر Name | عنوانه Text |
| FontMenu | Font |
| mFontBold | Bold |
| mFontItalic | Italic |
| mFontRegular | Regular |
| mFontUnderline | Underline |
| mFontStrike | Strike |
| mFontSmallCaps | SmallCaps |
| mFontAllCaps | AllCaps |
| Separator | - |
| MenuSize | Short Menu |

وها هو ذا الإجراء المستجيب لضغط أمر "تغيير الحجم" MenuSize:

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

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

**If MenuSize.text = "Short Menu" Then**

**MenuSize.text = "Long Menu"**

**Else**

**MenuSize.text = "Short Menu"**

**End If**

**' إظهار أو إخفاء بعض الأوامر (بعكس حالتها الحاليّة)**

**mFontUnderline.Visible = Not mFontUnderline.Visible**

**mFontStrike.Visible = Not mFontStrike.Visible**

**mFontSmallCaps.Visible = Not mFontSmallCaps.Visible**

**mFontAllCaps.Visible = Not mFontAllCaps.Visible**

**End Sub**

**القوائم المرنة Dynamic Menus:**

لا بد أنّك قد اعتدت على المرونة التي يمنحها لك VB.NET، لدرجة أنّك تتوقّع أنّ بإمكانك أن تنشئَ قوائمَ جديدةً من الكود أثناء تشغيل البرنامج.

هذا صحيح!

ويمكنك أن تستخدم مثل هذه الإمكانيّة، في عرض أسماء أخر ملفات فتحها المستخدم في برنامجك، تحت قائمة "ملف" File، حيث عند فتح البرنامج لأوّل مرّة، لا يوجد أي من هذه الملفات، ولكن كلما فتح المستخدم ملفا، ظهر اسمه في نهاية قائمة ملف.

وسيوضّح لك مشروع RunTimeMenu الموجود في مجلد أمثلة هذا الفصل، كيف تضيف وتزيل عناصر القائمة في وقت التشغيل.

واحدة من القوائم الرئيسيّة في هذا المشروع، هي القائمة Run Time Menu، وهي فارغة لا تحتوي على أيّة أوامر.. ولكن لدينا زران، لإضافة الأوامر إلى هذه القائمة وإزالتها منها، حيث سيتم إضافة الأمر الجديد في نهاية القائمة، وعند الإزالة سيتمّ إزالة آخر أمر في القائمة.

ها هو كود زر الإزالة، وهو يستخدم الوسيلة "إزالة" Remove الخاصّة بعناصر القائمة لإزالة أيّ عنصر ترسل لها رقمه.. هذا الرقم كما اتفقنا سيكون هو رقم آخر عنصر، ويمكن الحصول عليه من الوسيلة "عدد" Count والتي تخبرك بعدد العناصر الفرعية في القائمة.. وطبعا يحمل العنصر الأخير رقما يقل عن العدد الكلّيّ بواحدا، نظرًا لأنّ العنصر الأوّل يأخذ الرقم 0:

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

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

**With RunTimeMenu.MenuItems**

**If .Count > 0 Then .Remove(.Count - 1)**

**End With**

**End Sub**

تلاحظ وجود مقطع جديد هو مقطع "ضمن" With.. End With، وفائدته اختصار كتابة الكود، فبدلا من تكرار كتابة RunTimeMenu.MenuItems ثلاث مرات، استخدمنا هذه المقطع، لنخبر مترجم الكود Compiler أنّ أيّ جملة تبدأ بنقطة "." هي جملة مختصرة تبدأ بتعبير RunTimeMenu.MenuItems.. انظر للكود بدون مقطع "ضمن":

**If RunTimeMenu.MenuItems.Count > 0 Then**

**RunTimeMenu.MenuItems.Remove \_**

**(RunTimeMenu.MenuItems.Count - 1)**

**End If**

واضح طبعا مدى السهولة التي منحها لك تعبير "ضمن".. الطريف في الأمر، أنّك بمجرّد كتابة نقطة "." في أي موضع داخل مقطع "ضمن"، بحيث تكون هذا النقطة بداية كلمة جديدة لا يسبقها أي حرف، ستظهر لك قائمة الأعضاء الخاصّة بالكائن RunTimeMenu.MenuItems، تماما كما لو كنت قد كتبته صراحةً.

والآن ها هو ذا كود زر الإضافة:

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

**ByVal e As System.EventArgs)**

**With RunTimeMenu.MenuItems**

**.Add("اختيار " & .Count.ToString, \_**

**New EventHandler(AddressOf Me.OptionClick))**

**End With**

**End Sub**

ويستخدم هذا البرنامج الوسيلة "إضافة" Add، حيث يرسل لها معاملين:

* **المعامل الأوّل:**

وهو عنوان الأمر، وقد جعلناه "اختيار" متبوعا برقم، هو عدد الأوامر في القائمة، فالأمر الأوّل سيكون عنوانه "اختيار1" والثاني "اختيار2" وهكذا.

* **المعامل الثاني:**

وهو اسم الإجراء الذي سيستجيب لحدث ضغط هذا الأمر، وطبعا سنجعل لكل العناصر نفس الإجراء، وهو OptionClick().

هل أدهشك الأمر؟

نعم بإمكانك أن تعيّن الإجراءات المستجيبة للأحداث أثناء تنفيذ البرنامج!.. هل رأيتَ مرونةً أكثر من هذه؟

وتلاحظ أنّنا استخدمنا كلمة "عنوان" AddressOf قبل اسم الإجراء.. هذه الكلمة بديل للمؤشّرات Pointers في لغة C++، حيث إنّك تحاول أن ترسل عنوان الإجراء في الذاكرة، وليس مجرّد اسمه.

تلاحظ أيضا أنّنا أخبرنا VB أنّنا سنجعل هذا الإجراء يستجيب لحدث جديد، باستخدام تعبير New EventHandler().

ولكن أين هو الإجراء OptionClick؟

ها هو ذا:

**Private Sub OptionClick(ByVal sender As Object, ByVal e As EventArgs)**

**Dim ItemClicked As MenuItem = CType(sender, MenuItem)**

**Console.WriteLine("لقد ضغطت الأمر " & ItemClicked.Text)**

**End Sub**

أو يمكنك ان تكتبه مختصرا جدا كالتالي:

**Private Sub OptionClick(ByVal sender As Object, ByVal e As EventArgs)**

**Console.WriteLine("لقد ضغطت الأمر " \_**

**& CType(sender, MenuItem).Text)**

**End Sub**

طبعا لم نكتب التعبير Handles في هذا الإجراء، لأنّ الأحداث التي يستجيب لها ستضاف بطريقة مرنة أثناء تنفيذ البرنامج، من خلال الوسيلة Add كما رأينا من قبل.

**التعبير عن عناصر القوائم في الكود:**

بخلاف استخدام اسم القائمة مباشرةً، يمكنك استخدام هذه التعبيرات للتعامل مع عناصر القوائم:

- Me.Menu: للتتعامل مع القائمة الرئيسيّة.

- MenuItem: كل أمر من الأوامر يمثّله كائن من هذا النوع.

- MenuItems: مجموعة Collection تحتوي على كل الكائنات من النوع MenuItem، الفرعيّة من قائمة معيّنة.

فمثلا، للتعامل مع العنصر الأوّل في القائمة الرئيسيّة (وهو القائمة الرئيسيّة ملف)، يمكنك استخدام تعبير كالتالي:

**Dim F As MenuItem = Me.Menu.MenuItems(0)**

الآن يمكن الإشارة إلى قائمة ملف بالمتغيّر F، أو باستخدام اسم القائمة مباشرة (وليكن mFile).

وللتعامل مع العنصر الثاني في القائمة الرئيسيّة (وهو القائمة الرئيسيّة تحرير)، يمكنك استخدام تعبير كالتالي:

**Dim E As MenuItem = Me.Menu.MenuItems(1)**

وللتعامل مع أول أمر تحت قائمة ملف، يمكنك استخدام التعبير التالي:

**Dim N As MenuItem = Me.Menu.MenuItems(0).MenuItems(0)**

أو بطريقة أسهل، اعتمادا على المتغير F الذي عرّفناه من قبل:

**Dim N As MenuItem = F.MenuItems(0)**

وهكذا يمكن التعامل مع كل مستويات القوائم.

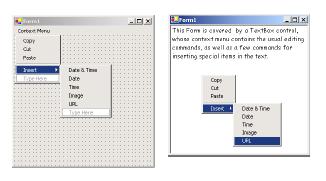
**إنشاء القوائم الموضعيّة Creating Context Menus:**

القوائم الموضعيّة هي تلك التي تظهر للمستخدم عندما يضغط زر الفأرة الأيمن.

ويمكنك أن تنشئ مثل هذه القوائم، بوضع واحدة أو أكثر من الأداة "القائمة الموضعيّة" ContextMenu على النموذج.

ولربط القائمة الموضعيّة بأداة ما على النموذج، بحيث تظهر القائمة الموضعيّة عند ضغط زر الفأرة الأيمن على هذه الأداة، اختر خاصيّة ContextMenu الخاصّة بالأداة، واجعل قيمتها القائمة الموضعيّة التي ترغب.

أمّا تصميم عناصر القائمة الموضعيّة، فستجده مماثلا لتصميم القوائم الرئيسيّة، فبمجرّد تحديد واحدة من القوائم الموضعيّة في صينيّة المكوّنات، ستظهر عناصر هذه القائمة في أعلى النموذج في وقت التصميم، حتّى تتمكّن من تحريرها، ولكنّها لن تظهر بالطبع في وقت التنفيذ، إلا عند ضغط زر الفأرة الأيمن على الأداة أو النموذج الذي ترتبط القائمة الموضعيّة به.



لا جديد على ما تظنّ؟

بل انتظر.. خذ هذه المعلومة الطريفة:

ماذا لو أردت أن تعرض في القائمة الموضعيّة بعض عناصر القائمة الرئيسيّة؟

هل يجب أن تعيد تصميم هذه العناصر من البداية؟

بالطبع لا.. إذن ما الحلّ؟

في الواقع هناك حلان:

الأوّل، هو أن تقوم بعرض عناصر القائمة الرئيسيّة، وضغط العنصر الذي تريده منها بزرّ الفأر الأيمن، واختيار الأمر Copy من القائمة.. الآن يمكنك أن تنتقل للقائمة الموضعيّة، للصق العنصر المنسوخ!

وخذ بالك: لن يكون للعنصر المنقول نفس اسم العنصر الذي نسخته منه.. كما أنّه لن ينفّذ نفس حدث الضغط عندما يضغطه المستخدم.. إذن فعليك أن تذهب لحدث ضغط العنصر الأصليّ، وفي جملة Handles أضف اسم العنصر الجديدة، بحيث يستجيب الإجراء لحدث ضغطه هو أيضا.

آه.. سيكون ذلك مرهقا، لا سيّما مع كثرة عدد العناصر!

ألا يوجد حلّ لهذا؟

بلى يوجد.. وتقدّمه لك الطريقة الثانية:

في هذه الطريقة لن نبذل أيّ مجهود على الإطلاق.. كلّ ما علينا هو استخدام الوسيلة "استنساخ القائمة" CloneMenu الخاصة بالعنصر الذي تريد إضافته للقائمة الموضعيّة، ثمّ إضافة النسخة الجديدة للقائمة الموضعيّة:

**ContextMenu1.MenuItems.Add(CopyMenu.CloneMenu)**

ولو أردت تحديد الموضع الذي سيضاف العنصر إليه فاستخدم الصيغة التالية:

**ContextMenu1.MenuItems.Add(1,CopyMenu.CloneMenu)**

ولكن... أين نكتب هذا الكود؟

طبعا يجب أن يكون أثناء تحميل النموذج.. الأفضل أن تكتبه في ختام الحدث New الخاص بالنموذج.

والآن إليك هذا الخبر السار: لن يقتصر الأمر على ظهور أمر النسخ Copy في القائمة الموضعيّة، بل سيقوم بتنفيذ حدث الضغط كذلك (بافتراض أنّ لديك أمر نسخ في القائمة الرئيسيّة وإجراء يستجيب لضغطه).

شيء آخر، غاية في الأهمّيّة:

هل سنحتاج لكتابة 10 سطور لإضافة 10 عناصر للقائمة الموضعيّة؟

بالطبع لا.. إننا نبرمج في القرن الواحد والعشرين يا صديقي (وهو على فكرة، قرن الاحتلال الأمريكيّ للعراق باسم الحريّة!!).

كلّ ما علينا هو استخدام الوسيلة AddRange لإضافة مصفوفة كاملة من العناصر للقائمة الموضعيّة:

**Dim I() As MenuItem = {CopyMenu.CloneMenu, PastMenu.CloneMenu}**

**ContextMenu1.MenuItems.AddRange(I)**

ويمكنك اختصار السطرين السابقين في سطر واحد كالتالي:

**ContextMenu1.MenuItems.AddRange(New \_**

**MenuItem() {CopyMenu. CloneMenu, PastMenu.CloneMenu })**

حيث قمنا بتعريف المصفوفة ووضع القيم الابتدائيّة لها داخل قوسي الوسيلة AddRange!!.. لاحظ أنّ قوسي المصفوفة يليان النوع MenuItem.. لاحظ كذلك عدم وجود علامة "=" بين تعريف المصفوفة وقيمها الابتدائيّة.

وبهذه الطريقة يمكنك أن تنقل نسخا طبق الأصل في الشكل والوظيفة من القائمة الرئيسيّ إلى القائمة الموضعيّة.

بقي شيء أخير.. ما زالت الكود السابق طويلا نسبيّا (رحم الله أيّام الشحططة في كتابة الكود!!).

فماذا لو أردت أن أنسخ كمّيّة كبيرة من العناصر مرّة واحدة؟

حسنا.. تعال نرى كيف ننقل القائمة Edit بكاملها من القائمة الرئيسيّة للقائمة الموضعيّة:

في البداية سننسخ قائمة Edit، ونضع مجموعة عناصرها الفرعيّة في متغيّر:

**Dim M As Menu.MenuItemCollection = \_**

**ToolsMenu.CloneMenu.MenuItems**

بعد ذلك، سنحاول أن نضع مجموعة العناصر الفرعيّة هذه في مصفوفة.. إذن فلنعرّف مصفوفة بعدد خانات مساوٍ لعدد العناصر الفرعيّة:

**Dim Itms(M.Count - 1) As MenuItem**

ثمّ سنستخدم الوسيلة CopyTo لنسخ العناصر من المجموعة للمصفوفة:

**M.CopyTo(Itms, 0)**

وأخيرا، سنستخدم الوسيلة AddRange الخاصة بالقائمة الموضعيّة لإضافة مصفوفة العناصر إليها:

**ContextMenu1.MenuItems.AddRange(Itms)**

مبروك يا عزيزي.. بهذه السطور الأربعة فقط، صارت لديك قائمة موضعيّة تحتوي على كلّ عناصر القائمة الرئيسيّة Edit (مهما كان عددها) وتقوم بتنفيذ كلّ وظائفها.

ألم أقل لك إنّ البرمجة مع VB.Net قد صارت متعة!

هيّا.. حاول الآن أن تحوّل القوائم الرئيسيّة التي أنشأناها في الأمثلة السابقة إلى قوائم موضعيّة.

**ملحوظة1:**

لا يمكن إضافة العناصر مباشرة من القوائم الرئيسيّة إلى القوائم الموضعيّة بدون استخدام الوسيلة CloneMenu، فالعنصر الواحد لا يمكن أن يتواجد في مكانين مختلفين.. ولكن يمكن أن توجد نسخ مختلفة من نفس العنصر (تشغل حيزا أكبر في الذاكرة) ترتبط كلّ منها بقائمة مختلفة..ولو حاولت أن تجرّب الكود السابق بعد حذف الوسيلة CloneMenu منه، فستجد أن العناصر اختفت من القوائم الرئيسيّة وظهرت في القائمة الموضعيّة!

**ملحوظة2:**

العناصر المستنسخة ليس لها أسماء برمجيّة.. فمثلا: الاسم CopyMenu يشير إلى أمر النسخ في القائمة الرئيسيّة فحسب، ولا علاقة له بمثيله الذي أضفناه للقائم الموضعيّة، مع أنّهما متماثلان تماما!.. فكيف إذن نتعامل مع أمر النسخ في القائمة الموضعيّة؟

في هذه الحالة عليك باستخدام مجموعة العناصر MenuItems كالتالي:

**ContextMenu1.MenuItems(0).Enabled = True**

هذا بافتراض أنّ أمر النسخ هو أوّل أمر في القائمة الموضعيّة (الأمر رقم 0).

**بناء نماذج مرنة**

* [**إضافة وإزالة الأدوات في وقت التنفيذ:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\4-%20بناء%20نماذج%20مرنة\1-%20إضافة%20وإزالة%20الأدوات%20في%20وقت%20التنفيذ.htm)
* [**مشروع ShowControls:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\4-%20بناء%20نماذج%20مرنة\2-%20مشروع%20عرض%20الأدوات.htm)
* [**مشروع DynamicForm:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\4-%20بناء%20نماذج%20مرنة\3-%20مشروع%20النموذج%20المرن.htm)
* [**إنشاء مستجيبات للأحداث في وقت التنفيذ:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\4-%20بناء%20نماذج%20مرنة\4-%20إنشاء%20مستجيبات%20للأحداث%20في%20وقت%20التنفيذ.htm)

**إضافة وإزالة الأدوات في وقت التنفيذ:**

يبدو أنّ VB لن يكفّ عن إدهاشك!

نعم: يمكنك أن تضيف وتزيل الأدوات من على النموذج في وقت التشغيل، بكيفيّة مشابهة لتك التي أضفنا وأزلنا بها القوائم.

إنّ النموذج يمتلك أيضا مجموعة الأدوات Controls Collection، والتي تحتوي على جميع الأدوات التي توجد على النموذج، حيث يمكنك أن تتعامل مع عناصر هذه المجموعة من خلال الكود.

ليس هذا فحسب، بل يمكنك استخدام الوسيلتين "إضافة" Add و"إزالة" Remove، لإضافة أو إزالة الأدوات أثناء تشغيل البرنامج.

فمثلا لإضافة زر جديد على النموذج، عرّف متغيرا من نوع الزر، وعدّل خصائصه كما تريد، ثم أضفه لمجموعة الأدوات كالتالي:

**Dim Bt As New Button**

**Bt.Text = "New Button"**

**Bt.Left = 100**

**Bt.Top = 60**

**Bt.Width = 80**

**Me.Controls.Add(Bt)**

ولإزالة هذا الزر من على النموذج، استخدم جملة كالتالية:

**Me.Controls.Remove(Bt)**

أو لو كنت تعرف رقم الزر في مجموعة الأدوات (وليكن Index)، أزله بإرسال هذا الرقم، كالتالي:

**Me.Controls.Remove(Index)**

وطبعا تتوقّع أنّ هذا الزرّ هو العنصر الأخير في المجموعة، لأنّه آخر ما أضيف، لهذا يمكن استخدام خاصيّة "عدد عناصر المجموعة" Count كالتالي:

**Controls.Remove(Controls.Count - 1)**

**ملاحظة:**

تحسب خاصية Count عدد الأدوات الرئيسيّة فقط على النموذج، فإذا كانت هناك أداة حاوية Container، مثل اللوحة Panel، فإنّ أيّة أدوات تنتمي لها لا يتم حسابها في العدّد، وذلك منطقيّ، فلو حذفت الأداة الحاوية، فستُحذف هي وما عليها من أدوات.. إنّ الأداة الحاوية وما عليها يعاملان كوحدة واحدة.. غنيّ عن الذكر أنّ الأدوات الحاوية، لها خاصيّة تدعى "أدوات" Controls ، وهي أيضا "مجموعة" Collection، تستطيع عبرها أن تتعامل مع كل أدوات هذه الأداة الأمّ.

وتوجد وسيلة أخرى شيّقة ضمن أعضاء مجموعة الأدوات، هي دالة "الجميع" All، وهي تُعيد مصفوفة كائنات من النوع Control، تحتوي علي جميع الأدوات الموجودة على النموذج، حتّى تلك التي تنتمي لأداة حاوية لغيرها Container Control.

كما يمكنك أن تزيل كل الأدوات الموجودة على النموذج بضربة واحدة، باستخدام الوسيلة "مسح" Clear.

ويمكنك استدعاء الوسيلة "إخماد العرض" SuspendLayout الخاصة بالنموذج قبل إضافة عدد من أدوات للنموذج في وقت التشغيل، وبعد الانتهاء من هذه العمليّة، تقوم باستدعاء الوسيلة "متابعة العرض" ResumeLayout.. بهذا يتحسّن أداء الكود بشكل كبير.

**Me.SuspendLayout**

**Dim Bt As New Button**

**Bt.Text = "New Button"**

**Bt.Left = 100**

**Bt.Top = 60**

**Bt.Width = 80**

**Me.Controls.Add(Bt)**

**'…………………..**

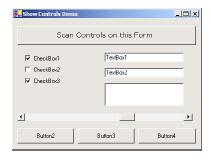
**' إضافة المزيد من الأدوات**

**Me.ResumeLayout**

**مشروع عرض الأدوات The ShowControls Project:**

افتح هذا المشروع من مجلد أمثلة الفصل، وأضف إلى النموذج أي عدد من الأدوات تريده.

والآن نريد أن نحصي كل الأدوات الموجودة على النموذج لنطبعها في نافذة المخرجات، تماما كما فعلنا مع القوائم من قبل.



هذا هو كود الزرّ الذي يفعل ذلك:

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

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

**Dim I As Integer**

**For I = 0 To Me.Controls.Count - 1**

**Console.WriteLine(Me.Controls(I).ToString)**

**If Me.Controls(I).GetType Is GetType(Panel) Then**

**Dim J As Integer**

**With Me.Controls(I).**

**For J = 0 To Controls.Count - 1**

**Console.WriteLine(.Controls(J).ToString)**

**Next**

**End With**

**End If**

**Next**

**End Sub**

ولكي تعرف نوع الأداة التي تتعامل معها في مجموعة الأدوات، يمكنك استخدام الدالة "اقرإ النوع" GetType الخاصّة بكل أداة:

**If Me.Controls(0).GetType Is GetType(TextBox) Then**

**MsgBox(CType(Controls(0),TextBox).Text)**

**End If**

لاحظ استعمال كلمة "هو" Is بدلا من علامة التساوي "=" في التعبير السابق، وذلك لأنّنا نتعامل مع كائنين، ولا يمكن استخدام علامة "=" في هذه الحالة.

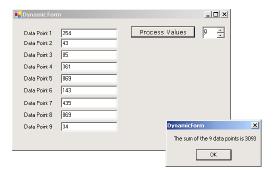
كما نحذرّك من استخدام تعبير كالتالي:

**If Me.Controls(I).GetType = "TextBox" Then**

فهو لن يعمل، بل لن تقبله اللغة أساسا.

**مشروع النموذج المرن The DynamicForm Project:**

ستجد هذا المشروع أيضا في مجلد أمثلة الفصل، وفيه سنسمح للمستخدم بتحديد عدد مربعات النصوص التي يريدها على النموذج، باستخدام الأداة "العداد الرقمي" NumericUpDown، وهي تعرض رقما، وتسمح لك بإنقاصه (بضغط السهم السفليّ) أو زيادته (بضغط السهم العلويّ).. وفي كلتا الحالتين، ستُطلق الأداة الحدث "القيمة تغيّرت" ValueChanged.



في هذا الحدث سنضيف أو نحذف مربعات النصوص:

**Private Sub NumericUpDown1\_ValueChanged(ByVal \_**

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

**Handles NumericUpDown1.ValueChanged**

**Dim TB As New TextBox()**

**Dim LBL As New Label()**

**Dim I, TBoxes As Integer**

**' احسب عدد مربعات النصوص على النموذج**

**For I = 0 To Me.Controls.Count - 1**

**If Me.Controls(I).GetType Is GetType(TextBox) Then**

**TBoxes += 1**

**End If**

**Next**

**' أضف مربع نص جديدا، إذا كان عدد المربعات الحاليّة**

**' أقلّ من العدد الذي يريده المستخدم**

**If TBoxes < NumericUpDown1.Value Then**

**TB.Left = 100**

**TB.Width = 120**

**TB.Text = ""**

**For I = TBoxes To NumericUpDown1.Value - 1**

**TB = New TextBox()**

**LBL = New Label()**

**If NumericUpDown1.Value = 1 Then**

**TB.Top = 20**

**Else**

**TB.Top =Controls(Controls.Count - 2).Top + 25**

**End If**

**Me.Controls.Add(TB) ' إضافة مربع نص**

**LBL.Left = 20**

**LBL.Width = 80**

**LBL.Text = "Data Point " & I**

**LBL.Top = TB.Top + 3**

**TB.Left = 100**

**TB.Width = 120**

**TB.Text = ""**

**' إضافة مستجيب لحدث دخول مربع النص وآخر لحدث مغادرته**

**Me.Controls.Add(LBL) ' إضافة لافتة**

**AddHandler TB.Enter, \_**

**New EventHandler(AddressOf TBox\_Enter)**

**AddHandler TB.Leave, \_**

**New EventHandler(AddressOf TBox\_Leave)**

**Next**

**Else ' إزالة مربعات النصوص الزائدة**

**' ستتم الإزالة من النهاية، وفي كل مرّة سنزيل مربع نص ولافتة، لهذا ستكون**

**' خطوة عداد التكرار هي -2**

**For I = Me.Controls.Count - 1 To Me.Controls.Count - \_**

**2\* (TBoxes - NumericUpDown1.Value) Step -2**

**Me.Controls.Remove(Controls(I))**

**Me.Controls.Remove(Controls(I - 1))**

**Next**

**End If**

**End Sub**

والآن عند ضغط الزر الأوّل، سنجمع القيم التي أدخلها المستخدم في كل مربعات النصوص، ونعرض له الناتج في رسالة.. لاحظ أنّنا نستخدم جملة التكرار التقليديّة "من إلى":

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

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

**Dim Ctrl As Control**

**Dim Sum As Double = 0, Points As Integer = 0**

**Dim I As Integer**

**For I = 0 To Controls.Count - 1**

**Ctrl = Controls(I)**

**If Ctrl.GetType Is GetType(TextBox) Then**

**If IsNumeric(CType(Ctrl, TextBox).Text) Then**

**Sum += CType(Ctrl, TextBox).Text**

**Points += 1**

**End If**

**End If**

**Next**

**MsgBox("عدد الخانات هو " & Points.ToString & \_**

**" ومجموع بياناتها هو " & Sum.ToString)**

**End Sub**

ويمكننا إعادة كتابة الإجراء السابق باستخدام الجملة التكراريّة "لكل عنصر في"، كالتالي:

**Private Sub BtProcess2\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtProcess2.Click**

**Dim TB As Control**

**Dim Sum As Double = 0, Points As Integer = 0**

**For Each TB In Me.Controls**

**If TB.GetType Is GetType(TextBox) Then**

**If IsNumeric(CType(TB, TextBox).Text) Then**

**Sum = Sum + CType(TB, TextBox).Text**

**Points = Points + 1**

**End If**

**End If**

**Next**

**MsgBox("عدد الخانات هو " & Points.ToString & \_**

**" ومجموع بياناتها هو " & Sum.ToString)**

**End Sub**

**إنشاء مستجيبات للأحداث في وقت التنفيذ Creating Event Handlers at Runtime:**

رأيتَ كيف حدّدنا مستجيبا لحدث ضغط القوائم التي أضفناها في وقت التشغيل.. بَدَهيٌّ جدًّا أنّ هناك وسيلةً مشابهةً، لتحديد المستجيبات لأحداث الأدوات التي نضيفها للنموذج في وقت التشغيل.

بالتأكيد ستبدأ العمل بكتابة الإجراء الذي سيستجيب للحدث، حتّى يمكنك فيما بعد، ربطه بالأداة المضافة.

وبالتأكيد أصبح محفوظا لديك شكل معاملي هذا الإجراء sender و e.. عامّة يمكنك أن تنسخ تعريف المعاملات، من أي مستجيب لحدث (من نفس نوع حدثك) سبقت كتابته.

بعد ذلك يمكنك أن تربط الإجراء بالأداة، عن طريق جملة "أضف مستجيبا" AddHandler، والتي لها الصيغة التالية:

**AddHandler أداة.حدث, New EventHandler(AddressOf اسم الإجراء)**

فمثلا، لربط الحدث ProcessNow، بالزر Calculate، ليستجيب لضغطه، يمكنك استخدام التعبير التالي:

**AddHandler Calculate.Click, \_**

**New EventHandler(AddressOf ProcessNow)**

تعالَ نطوّر المشروع DynamicForm، بحيث يتغيّر لون مربع النصّ عندما يستقبل مؤشّر الكتابة، ويعود للون الأبيض عندما يغادره مؤشّر الكتابة، بحيث يبدو جليّا للمستخدم، أيّ من مربعات النصوص هو الذي تتم الكتابة فيه الآن.

ها هو الإجراء المستجيب لدخول مربع النصّ:

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

**ByVal e As System.EventArgs)**

**CType(sender, TextBox).BackColor = Color.LightCoral**

**End Sub**

وها هو ذا الإجراء المستجيب لمغادرته:

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

**ByVal e As System.EventArgs)**

**CType(sender, TextBox).BackColor = Color.White**

**End Sub**

ولعلّك تذكر الجملتين التاليتين، فهما موجودتان في نهاية الجزء الذي يضيف مربعات النصوص واللافتات على النموذج في التطبيق DynamicForm:

**AddHandler TB.Enter, New EventHandler(AddressOf TBox\_Enter)**

**AddHandler TB.Leave, Neww EventHandler(AddressOf TBox\_Leave)**

وبالتأكيد تفهم الآنَ ما وظيفتاهما.

والآن شغّل تطبيق النموذج المرن DynamicForm، واستمتع بتجربته!

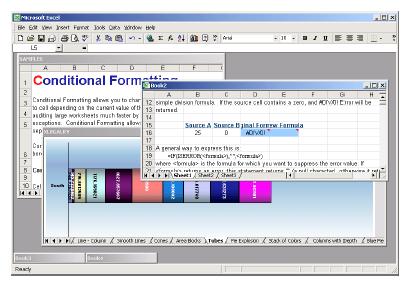
**واجهة الوثائق المتعدّدة MDI**

* [**واجهة الوثائق المتعدّدة؟**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة\0-%20ما%20هي%20واجهة%20الوثائق%20المتعدّدة.htm)
* [**بناء تطبيق متعدد الوثائق:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة\1-%20بناء%20تطبيق%20متعدد%20الوثائق.htm)
* [**القائمة "نافذة" Window:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة\2-%20نافذة.htm)
* [**مزج قوائم النوافذ الداخليّة بقوائم النافذة الرئيسيّة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة\3-%20مزج%20قوائم%20النوافذ%20الداخليّة%20بقوائم%20النافذة%20الرئيسيّة.htm)
* [**التعامل مع النماذج الداخليّة من خارجها:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة\4-%20التعامل%20مع%20النماذج%20الداخليّة%20من%20خارجها.htm)
* [**إنهاء التطبيق متعدّد الوثائق:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\5-%20واجهة%20الوثائق%20المتعدّدة\5-%20إنهاء%20التطبيق%20متعدّد%20الوثائق.htm)

**ما هي واجهة الوثائق المتعدّدة:**

في كلّ ما سبق، كنّا نتعامل مع نموذج واحد، وأحيانا كنّا نعرض نموذجا آخر.. حتّى في الحالة الثانية، كان النموذجان يظهران على الشاشة دون أن تربطهما واجهة واحدة.. إنّ هذا الوضع ليس مفضلا، فقد يزعج المستخدم أن يفتح 5 نماذج مثلا في تطبيقك، فتبدو مبعثرة على الشاشة بلا نظام.. فإذا أراد أن يصغرها أو يكبّرها، اضطرّ إلى أن يصغرّ أو يكبّر كل واحد منها على حدة!

وأفضل حل لهذه المشكلة، هو أن تكون هناك نافذة رئيسيّة للتطبيق، تحتوي كلّ النماذج الأخرى التي يفتحها المستخدم في هذا التطبيق، وتحتوي أيضا على القوائم الرئيسيّة Menus للتطبيق.. هذه النافذة الرئيسيّة هي ما يسمّى بواجهة الوثائق المتعدّدة Multiple Document Interface (MDI).. فإذا أراد المستخدم تصغير كل نماذج التطبيق، فما عليه سوى تصغير النافذة الرئيسيّة، لتختفي من على الشاشة بكلّ ما تحويه من نماذج.. وإذا حرّكها، تحرّكت بكلّ ما تحويه من نماذج، وإذا أغلقها، أُغلقت بكلّ ما تحويه من نماذج (يُغلق التطبيق نهائيّا).



ويجب هنا أن نقرّر هذه الحقائق:

- تسمّى واجهة الوثائق المتعدّدة النافذة الرئيسيّة أو النافذة الأمّ.

- تسمّى أي نافذة أخرى النافذة الداخليّة (الابنة).

- من البَدَهِيّ أن يكون للتطبيق نافذة رئيسيّة واحدة، ولكن لا قيود على عدد النوافذ الداخليّة التي يمكن أن يحتويها.

- في وقت التصميم تظهر لك كلّ نافذة مستقلّة، سواء الرئيسيّة أو الداخليّة، ولكن عند تشغيل البرنامج، تظهر النوافذ الداخليّة محتواة بالكامل داخل النافذة الرئيسيّة.

- توجد قائمة رئيسيّة Menu للنافذة الرئيسيّة، ويمكن أن توجد قوائم للنماذج الداخليّة، ولكن عند عرض النموذج الداخليّ، يتمّ دمج عناصر قائمته مع عناصر القائمة الرئيسيّة، ليظهرا معا على النافذة الرئيسيّة.

ولكي يتضح لك الأمر، نتعال نتعرّف على كيفيّة بناء التطبيقات متعددة الوثائق.

**بناء تطبيق متعدد الوثائق:**

ابدأ مشروعا جديدا، وأسمِه MDIProject.. وفي متصفّح المشاريع حدّد النموذج، وغيّر اسمه من Form1.vb إلى MDIForm.vb.. ولكي تجعل هذا النموذج هو النافذة الرئيسيّة MDI، حدّد خاصيّة "إنّه يحتوي على وثائق متعدّدة" IsMdiContainer في نافذة الخصائص، واجعل قيمتها True.. لاحظ تغيّر شكل النموذج بعد قيامك بهذا.

غيّر كذلك قيمة خاصيّة Text إلى "النافذة الرئيسيّة".

الآن نحنُ بصدد إنشاء النماذج الداخليّة.. إنّ بإمكانك أن تنشئ أيّ عدد من هذه النماذج، وإن كان الغالب أن تنشئ نموذجا واحدا تستخدمه كقالب، حيث يمكنك أن تعرض منه أيّ عدد تريده من النسخ.

الآن أضف نموذجا جديدا للمشروع وأسمه ChildForm.

ولعرض هذا النموذج كنموذج داخليّ ضمن النافذة الرئيسيّة، سننشئ القائمة File، ومن خلالها يمكن للمستخدم عرض نسخة جديدة من هذا النموذج باستخدام الأمر New.. ابدأ أولا بإنشاء القوائم التالية:

|  |  |
| --- | --- |
| عنوان القائمة | اسمها البرمجيّ |
| ملفّ File | FileMenu |
| جديد New | FileNew |
| خروج Exit | FileExit |
| نافذة Window | WindowMenu |
| تجاور أفقيّ Tile Horizontally | WindowTileH |
| تجاور رأسيTile Vertically | WindowTileV |
| تتالي Cascade | WindowCascade |
| ترتيب الأيقونات Arrange Icons | WindowArrange |

والآن اكتب الكود التالي في كود الأمر New:

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

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

**' عرفّ متغيّرا تحفظ فيه عدد نسخ النموذج**

**Static nWindow As Integer**

**' عرّف نسخة جديدة من النموذج الابن**

**Dim child As New ChildForm()**

**' اجعل النافذة الرئيسيّة تحتوى النموذج الابن**

**child.MdiParent = Me**

**' اعرض على النافذة رقمها**

**child.Text = “النافذة رقم # “ & nWindow.ToString**

**' اعرض النافذة الداخليّة**

**child.Show()**

**' لقد زاد عدد النوافذ بنافذة جديدة**

**nWindow = nWindow + 1**

**End Sub**

الجديد بالنسبة لك في هذا الإجراء، هو الأمر:

**Child.MdiParent = Me**

هذا هو الأمر الذي يجعل أيّ نافذة تنتمي للنافذة الرئيسيّة، حيث سيتمّ عرضها داخلها، وستظهر قوائمهما متداخلة.

شغّل التطبيق لترى ناتج ما فعلناه حتّى الآن.. جرّب أن تضغط الأمر New أكثر من مرّة، لترى كيف يمكنك فتح أكثر من نافذة داخل النافذة الرئيسيّة.

طبعا سترى هذا النوافذ فارغة.. لا غبار عليك لو وضعت على النموذج ChildForm مربّع نصّ وبعض الأزرار.. إنّ هذا بالطبع يرجع لنوعيّة الوظيفة التي تريد أن يؤدّيها هذا النموذج، ولكنّ الأمر هنا يندرج تحتَ بند التجريب.

**ملحوظة:**

على خلاف ما كان في VB6، يمكنك في VB.NET أن تضع أيّ أداةٍ على النافذة الرئيسيّة، ولكن مع ملاحظة أنّها ستظهر لك دائما، حتّى لو تمّ عرض نافذة داخليّة أمامها.. بمعنى آخر، ستعلو هذه الأداة كلّ النوافذ الداخليّة.. إنّ لهذا حكمة، وهي تمكينك من وضع شريط أدوات على النافذة الرئيسيّة، بحيث لا يمكن أن تخفيه أيّ نافذةٍ داخليّة.. لهذا فلا تتعجّب لو وضعت أداة على النافذة الرئيسيّة ومنحتها حجم النافذة كلّها، ثمّ حاولت أن تعرض أيّ نافذة داخليّة فلم تظهر، ففي هذه الحالة ستكون مختفيةً تماما تحت الأداة.

**القائمة "نافذة" Window:**

لتجدنّ في كلّ تطبيقات الوثائق المتعدّدة قائمة تسمّى "نافذة" أو "إطار" Window.. هذه القائمة تحتوي على نوعين من الأوامر: تلك الخاصّة بترتيب النوافذ الداخليّة المفتوحة حاليّا (بجعلها تتجاور أفقيّا أو رأسيّا أو على التوالي)، وتلك التي تعرض أسماء هذه النوافذ، مع وضع علامة (✓) أمام النافذة الفعّالة حاليّا Active.



طبعًا النوعَ الأوّل أنشئنا له أوامر في قائمة Window.. ولكن ماذا عن النوع الثاني؟

اطمئنّ، فلن تكتب أيّ كود لذلك.. كل ما ستفعله هو أن تضغط بزرّ الفأرة الأيمن على قائمة Window، ومن القائمة الموضعيّة تختار Properties.. والآن ببساطة ستغيّر قيمة الخاصيّة "قائمة الوثائق المتعدّدة" MDIList من False إلى True.. إنّ هذا سيجعل كلّ عناوين النوافذ الداخليّة المفتوحة في تطبيقك تظهر تلقائيّا تحت قائمة Window، مع تحديد النافذة الفعالة فيها.. وعندما يضغط المستخدم بالفأرة اسم أيّ نافذة منها، ستصبح هي النافذة الفعّالة التي تظهر فوق باقي النوافذ وتستقبل أحداث لوحة المفاتيح.

بقيَ أن نكتب كود أوامر الترتيب.. إنّ كل ما سنحتاجه لفعل ذلك هو الوسيلة "مظهر الوثائق المتعدّدة" LayoutMdi، وهي تأخذ معاملا عبارة عن إحدى قيم المرقّم MdiLayout، وهي:

|  |  |
| --- | --- |
| محاذاة أفقيّة  TileHorizontal | سيتمّ تغيير حجم النوافذ الداخليّة حتى تظهر كلّها متجاورة أفقيّا، بحيث تملأ كلّ مساحة النافذة الرئيسيّة. |
| محاذاة رأسيّة  TileVertical | سيتمّ تغيير حجم النوافذ الداخليّة حتى تظهر كلّها متجاورة رأسيّا. |
| تتالي Cascade | سيتمّ عرض النوافذ بعضها فوق بعض، مع إزاحة كلّ منها قليلا رأسيّا وأفقيّا، بحيث تظهر أجزاء من كلّ منها للمستخدم. |
| ترتيب الأيقونات  ArrangeIcons | عندما تقوم بتصغير نافذة داخليّة، فإنّها ستنكمش كأيقونة صغيرة في جانب النافذة الرئيسيّة السفليّ الأيسر.. وعندما يكون لديك أكثر من نافذة مصغّرة، يمكنك استخدام هذا الأمر لترتيب أيقوناتها. |

وباستخدام هذه الوسيلة، ستجد أنّ كلّ أمر من أوامر قائمة القائمة Window لن يزيد عن سطر واحد، كما يلي:

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

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

**' محاذاة أفقيّة**

**Me.LayoutMdi(MdiLayout.TileHorizontal)**

**End Sub**

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

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

**' محاذاة رأسيّة**

**Me.LayoutMdi(MdiLayout.TileVertical)**

**End Sub**

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

**ByVal e As EventArgs) Handles WindowCascade.Click**

**' تتالي**

**Me.LayoutMdi(MdiLayout.Cascade)**

**End Sub**

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

**ByVal e As EventArgs) Handles WindowArrange.Click**

**' ترتيب الأيقونات**

**Me.LayoutMdi(MdiLayout.ArrangeIcons)**

**End Sub**

**مزج قوائم النوافذ الداخليّة بقوائم النافذة الرئيسيّة:**

افترض أنّ تطبيقك هو محرّر نصوص، يستطيع المستخدم أن يفتح فيه أكثر من وثيقة في نفس الوقت.. وافترض أنّه لا توجد في هذه اللحظة أيّة نوافذ مفتوحة في التطبيق.. في هذه الحالة ليس من المنطقيّ أن تعرض النافذة الرئيسيّة قائمة تحرير التي تقوم بعمليّات النسخ والقصّ واللصق، بينما لا توجد وثيقة مفتوحة!

إنّ أنسب حلّ لهذا، هو أن تضع قائمة تحرير على نموذج الوثيقة نفسها، بحيث تضمن أنّها لن تكون متاحة للمستخدم إلا عندما تكون هناك وثيقة مفتوحة.. ولتجريب ذلك، أضف القائمة التالية للنموذج ChildForm:

|  |  |
| --- | --- |
| عنوان القائمة | اسمها البرمجيّ |
| تحرير Edit | EditMenu |
| نسخ Copy | EditCopy |
| قصّ Cut | EditCut |
| لصق Paste | EditPaste |

الآن لو شغّلت البرنامج، فستظهر لك النافذة الرئيسيّة بقائمتيها File و Window.. فإذا فتحت نافذة جديدة، فستجد أنّ القائمة Edit قد ظهرت بجانب هاتين القائمتين.

ولكن افترض أنّك تريد أن تعرض قائمة Edit بعد قائمة File.. في هذه الحالة استخدم خاصيّة "ترتيب المزج" MergeOrder لكلّ قائمة.. هذه الخاصيّة تأخذ رقما، يدلّ على ترتيب عرضها في مستواها (سواء أكانت قائمة رئيسيّة أو فرعيّة أو أمرا).. فمثلا، لعرض القوائم بالترتيب التالي: File ثمّ Edit ثمّ Window، اجعل قيمة هذه الخاصيّة في قائمة File = 0، وفي قائمة Edit = 1، وفي قائمة Window = 2.

ليس هذا كلّ شيء، فلا يقتصر الأمر على إضافة قائمة جديدة باسم جديد، ولكنّ بإمكانك أيضا إضافة أوامر جديدة لقوائم النافذة الرئيسيّة.. فمثلا، ليس من المنطقيّ أن تعرض في قائمة ملفّ الأمر "حفظ"، بينما لا توجد وثيقة معروضة حاليّا في التطبيق ليتمّ حفظ محتوياتها لملفّ على الجهاز.. في هذه الحالة يمكنك أن تنشئ قائمة اسمها "ملف" File على النموذج ChildForm.. في هذه القائمة أنشئ الأمر "حفظ" Save.. والآن شغّل التطبيق.. ستجد أنّ هناك قائمتين اسماهما File، إحداهما تحتوي على الأمر New والأمر Exit، بينما الأخرى تحتوي على الأمر Save.. طبعا ليس هذا ما نريده، فنحن نريد قائمة واحدة اسمها File، تحتوى على كلّ هذه الأوامر معا.

لا عليك.. إنّ لديك خاصيّة تسمّى "نوع المزج" MergeType، يمكنك بها أن تختار كيفية مزج القوائم، تبعا للقيم التالية للمرقّم MenuMerge:

|  |  |
| --- | --- |
| إضافة Add | يتمّ إضافة القوائم مستقلّة عن بعضها، حتّى لو تشابهت في الأسماء.. هذه هي القيمة الافتراضيّة. |
| دمج العناصر MergeItems | هذا هو الوضع الذي نريده، حيث يتمّ مزج عناصر القوائم ذات الأسماء المتماثلة لتصبح قائمة واحدة.. وفي هذه الحالة يمكنك تحديد ترتيب العناصر في القائمة الجديدة عن طريق خاصيّة MergeOrder.. لاحظ أنّ هذا لا ينطبق فقط على القوائم الرئيسيّة، بل وينطبق كذلك على القوائم الفرعيّة ذات الأسماء المتماثلة. |
| إزالة Remove | إذا تماثل اسما قائمتين، يتمّ عرض قائمة النافذة الرئيسيّة، ولا يتمّ عرض قائمة النافذة الداخليّة. |
| استبدال Replace | إذا تماثلت اسما قائمتين، يتمّ عرض قائمة النافذة الداخليّة، ولا يتمّ عرض قائمة النافذة الرئيسيّة. |

إنّ كلّ قائمة وكلّ أمر في قائمة يمتلك خاصيّة MergeType، وبهذا يمكنك أن تحدّد طريقة مزج مختلفة لكلّ أمر أو قائمة.. فمثلا في حالتنا هذه، سنجعل قيمة هذه الخاصيّة MergeItems في قائمة File الخاصّة بالنموذج ChildForm، وفي نفس الوقت سنترك قيمة هذا الخاصيّة Add للأمر Save، فنحن واثقون من أنّ هذا الاسم لن يتشابه مع أيّ أمر في قائمة File الخاصّة بالنافذة الرئيسيّة.

إنّ هذه المرونة تجعل تصميم القوائم أكثر كفاءة (بحيث لا يظهر للمستخدم قائمة أو أمر إلا إذا كانا قابلين للاستخدام)، كما تجعل كتابة الكود أسهل.

ولكي تتعامل مع القوائم الممتزجة من الكود، استخدم خاصيّة MergedMenu الخاصّة بالنموذج الرئيسيّ، للحصول على كائن القائمة الرئيسيّة لهذا المزيج، حيث من خلاله يمكنك التعامل مع باقي القوائم والأوامر.

**التعامل مع النماذج الداخليّة من خارجها:**

بينما أنت تكتب الكود في النموذج ChildForm، يمكنك استخدام كلمة Me للإشارة إلى النسخة الحاليّة من هذا النموذج.. فمثلا لو أردت أن تحدّد كلّ النصّ في مربّع نصّ موجود على النسخة الحالية، يمكنك استخدام الجملة التالية:

**Me.TextBox1.SelectAll**

ولكن كيف يمكن التعامل مع نسخ هذا النموذج المختلفة من خارجها (من النموذج الرئيسيّ MDIForm مثلا)؟

في هذه الحالة يمكنك أن تستخدم الخاصيّة "النافذة الداخليّة الفعّالة" ActiveMdiChild.. هذه الخاصيّة تسمح لك بالتعامل مع النافذة الفعّالة (التي تستقبل أحداث لوحة المفاتيح من المستخدم).. فمثلا لو أردت أن تغيّر عنوان النافذة الفعّالة الحاليّة من داخل النموذج MDIForm، استخدم الجملة التالية:

**Me.ActiveMdiChild.Text = "عنوان جديد"**

وللتعامل مع مربّع نصّ على هذا النموذج وتحديد كلّ نصّه، استخدم الجملة التالية:

**Me.ActiveMdiChild.Controls(0). SelectAll**

حيث استخدمنا مجموعة الأدوات Controls Collection للتعامل مع مربّع النصّ، مفترضين أنّ ترتيبه على النموذج هو رقم 0 .. وهنا يجب الانتباه للمخاطر التالية:

- لو لم تكن هناك نافذة مفتوحة في تلك اللحظة فسيحدث خطأ.

- يجب أن تعرف رقم الأداة بالضبط!

- لو كنت تتعامل مع نوافذ مختلفة، فستكتب كودا معقّدا للتعامل مع أدوات كلّ منها، حيث ستحتاج لمعرفة رقم كلّ أداة على كلّ نموذج.

- الجملة التي كتبناها تعتبر ربطا متأخّرا Late Binding، ولو كان اختيار التحويل الدقيق فعّالا Option Strict On، فسيرفض VB هذه الجملة، ويجب عليك إعادة كتابتها كالتالي:

**CType (Me.ActiveMdiChild.Controls(0), TextBox). SelectAll**

**ملحوظة:**

أعتقد أنّ هذه العيوب قد أجابت على سؤال لا بدّ أنّه قد دار في ذهنك، وهو: لماذا نرهق أنفسنا بتصميم قوائم للنوافذ الداخليّة، ثمّ نعرضها على النافذة الرئيسيّة؟.. لماذا لا نضع كلّ القوائم على النافذة الرئيسيّة ونريح أنفسنا، ولا ضرر بعد ذلك من إخفاء بعض القوائم والأوامر التي لا نحتاجها؟

والإجابة تكمن في صعوبة كتابة كود هذه القوائم عندما نضعها على النموذج الرئيسيّ ونحاول من خلالها التعامل مع النوافذ الداخليّة.. وأعتقد أنّك رأيت بنفسك الفارق بينَ تحديد النصّ من داخل النافذة ومن خارجها في المثال السابق!!

وهناك طريقة أخرى كالتالي:

**Dim F As ChildForm**

**F = CType (Me.ActiveMdiChild, ChildForm)**

**F.TextBox1.Text = ""**

ولا تنسَ أن تغيّر خاصيّة Modifier الخاصّة بمربّع النصّ TextBox1 من Private إلى Public، حتّى يعمل الكود السابق.

عامّةً، إنّ هذه الطريقة ستعمل جيّدا في تطبيقنا الحاليّ، لأنّ كل النوافذ الداخليّة هي نسخ من النموذج ChildForm.. أمّا لو كان تطبيقك يحتوى على نماذج داخليّة أخرى، فيجب أولا من التأكّد من نوع النموذج الفعّال حتّى لا تتسبّب جملة تحويل الأنواع في خطأ.. وطبعا يجب أن تتأكّد أن هناك نافذة فعالة أساسا:

**Dim F As ChildForm**

**If Not Me.ActiveMdiChild Is Nothing AndAlso \_**

**Me.ActiveMdiChild.GetType Is F.GetType Then**

**F = CType (Me.ActiveMdiChild, ChildForm)**

**F.TextBox1.Text = ""**

**End If**

وهناك حلّ ثالث، قد تجده أكثر عمليّة.. فبدلا من كتابة كلّ هذا الكود في كلّ موضع تريد التعامل فيه مع النوافذ الداخليّة، عرّف متغيّرا عامّا على مستوى النموذج MDIForm كالتالي:

**Public Shared activeChildForm As ChildForm**

هذا المتغيّر يجب أن يشير باستمرار إلى النافذة الحاليّة الفعّالة.. ولفعل هذا، سنستخدم الحدث "تفعيل نافذة داخليّة" MdiChildActivate، الذي يستدعى كلّما تمّ تفعيل إحدى نوافذ التطبيق الداخليّة:

**Private Sub MDIForm\_MdiChildActivate(ByVal sender As Object, \_**

**ByVal e As EventArgs) Handles MyBase.MdiChildActivate**

**activeChildForm = CType (Me.ActiveMdiChild, ChildForm)**

**End Sub**

الآن يمكنك استخدام المتغيّر activeChildForm في أيّ موضع في كود النافذة الرئيسيّة، مطمئنّا إلى أنّه يشير دائما إلى النافذة الفعّالة الحاليّة ـ إن كانت هناك واحدة.

وهناك طرق أخرى يمكنك بها التعامل مع النوافذ الداخليّة.. فمثلا، يمكنك أن تجعل إحدى النوافذ الداخليّة نشيطة (فعّالة)، باستدعاء الوسيلة ActivateMdiChild، كالتالي:

**Me.ActivateMdiChild(النافذة التي تريد تنشيطها)**

كما يمكنك الحصول على مصفوفة تحتوي على كلّ النماذج المفتوحة حاليّا داخل النافذة الرئيسيّة، عن طريق الخاصيّة MdiChildren.. فمثلا يمكنك معرفة عدد الوثائق المفتوحة في تطبيقك بالجملة التالية:

**MsgBox (Me.MdiChildren.Length)**

ويمكنك أن تعرف النافذة الرئيسيّة لأيّ نافذة داخليّة باستخدام الخاصيّة ParentForm.. فمثلا لو أردت أن تكتب الجملة السابقة لكن داخل النموذج ChildForm، فاستخدم الطريقة التالية:

**MsgBox (Me.ParentForm.MdiChildren.Length)**

كما أنّ لديك الخاصيّتين "إنّها نافذة رئيسيّة" IsMdiContainer، و"إنّها نافذة داخليّة" IsMdiChild لمعرفة نوع النافذة.

**إنهاء التطبيق متعدّد الوثائق:**

عند إغلاق أيّ نافذة داخليّة، يتمّ استدعاء حدث إغلاقها Closing.. في هذا الحدث يمكنك أن تحفظ إعدادات النافذة Settings، كموضعها وحجمها، أو أن تتحقّق إذا كانت هناك تغييرات على مربّع النصّ لتنبّه المستخدم لحفظها.... إلخ.

ولكن ماذا سيحدث لو أغلق المستخدم النافذة الرئيسيّة أولا؟

في هذه الحالة يتمّ استدعاء أحداث إغلاق النوافذ الداخليّة أولا، ثمّ يتمّ استدعاء حدث إغلاق النافذة الرئيسيّة في النهاية.

ولكي تدرك هذا، اكتب الجملة التالية في حدث إغلاق النموذج ChildForm:

**Private Sub ChildForm\_Closing(sender As Object, e As \_**

**ComponentModel.CancelEventArgs) Handles MyBase.Closing**

**MsgBox(Me.Text)**

**End Sub**

واكتب الجملة التالية في حدث إغلاق النافذة الرئيسيّة:

**Private Sub MDIForm\_Closing(sender As Object, e As \_**

**ComponentModel.CancelEventArgs) Handles MyBase.Closing**

**MsgBox("إنهاء التطبيق")**

**End Sub**

والآن جرّب فتح نوافذ عديدة وإغلاق بعضها وإغلاق النافذة الرئيسيّة بينما بعض النوافذ الداخليّة مفتوح، وانظر للرسائل التي ستظهر لك في أثناء ذلك.

بناءً على ما سبق، إذا أردت إنهاء التطبيق بسطر من الكود (كما في قائمة Exit)، فابتعد عن جملة End (وغنّ لها!!)، لأنّها تنهي التطبيق في الحال، دون استدعاء أيّ حدث على الإطلاق.. وكبديل، استخدم الوسيلة "إغلاق" Close، فهي تتصرّف كما لو أنّ المستخدم قد ضغط زر إغلاق النموذج، وبذلك يتمّ استدعاء أحداث الإغلاق:

**Me.Close()**

ولمزيد من التدريب على استخدام تطبيقات الوثائق المتعدّدة، افحص مشروع MDIProject ومشروع ScrollingPictureBox في مجلّد برامج هذا الفصل، ومشروع MDIPad في مجلّد برامج الفصل السابع.

**النماذج المترابطة:**

بخلاف واجهة الوثائق المتعدّدة، يمكنك أن تربط نموذجا بآخر، بحيث يبدو وكأنّ أحدهما يتسيّد الآخر، فإذا تمّ تصغير النموذج المالك Owner تمّ تصغير النموذج المملوك Owned معه، وإذا تمّ إغلاق النموذج المالك، تمّ إغلاق النموذج المملوك معه.. وفي كلّ الأحوال، سيظهر النموذج المملوك أمام النموذج المالك، (إلا إذا تمّ تصغير النموذج المملوك بالطبع).

إنّ هذا يختلف عن واجهة الوثائق المتعدّدة في نواحي كثيرة، أهمّها أنّ النموذج المملوك غير مقيّد بمساحة النموذج المالك، ولا يتمّ تحريكهما معا.. كما أنّ برنامجك يمكن أن يحتوي على أكثر من مجموعة من النماذج المترابطة (أكثر من نموذج مالك).

ولا يوجد ما يمنع أن يكون للنموذج أكثر من نموذج تابع (مملوك).

ويمكن أن تستفيد من هذه الإمكانيّة عند عرض مربّع حوار البحث والاستبدال، فليس من المنطقيّ أن يبقى هذا المربّع معروضا، بعد تصغير أو إغلاق النموذج الذي يبحث في النصّ الموجود فيه.

ولكي تجعل نموذجا سيّدا (مالكا) لآخر، استخدم الوسيلة AddOwnedForm.. افترض أنّ لدينا نموذجا اسمه Form1، ونريد أن نجعل النموذج Form2 تابعا له.. لفعل ذلك استخدم الكود التالي من داخل Form1:

**Dim F2 As New Form2**

**Me.AddOwnedForm(F2)**

**F2.Show**

والصورة التالية تريك كيف سيظهر النموذجان.. لاحظ أنّ النموذج Form2 ما زال أمام النموذج Form1 رغم أنّ Form1 هو النموذج الفعّال Active.



أمّا إذا أردت أن تنشئ هذا الترابط من داخل النموذج Form2، فاستخدم الخاصيّة "مالكي" Owner، لجعل النموذج Form2 تابعا للنموذج Form1:

**Me.Owner = F1**

**Me.Show**

هذا بافتراض أنّ F1 متغيّر عامّ للبرنامج كلّه، تمّ تعريفه في قالب كود Module كالتالي:

**Public F1 As New Form1**

ويمكن معرفة كلّ النماذج التي يمتلكها نموذج ما، باستخدام خاصيّة OwnedForms، التي تعيد مصفوفة تحتوي على كلّ النماذج التابعة:

**Dim OF() As Form = Me.OwnedForms**

ولفكّ هذا الارتباط بين نموذجين، يمكنك استدعاء الوسيلة "إزالة النموذج المملوك" RemoveOwnedForm، التي ترسل لها اسم النموذج المملوك الذي تريد تحريره، حتّى يمكن عرضه بحرّيّة بعد ذلك:

**Me.RemoveOwnedForm(F2)**