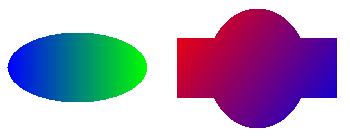
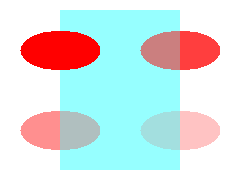
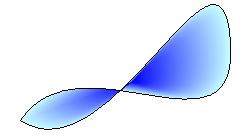
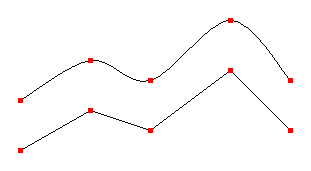
**الفصل الثاني عشر**

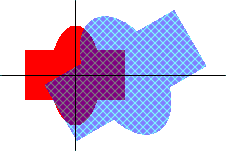
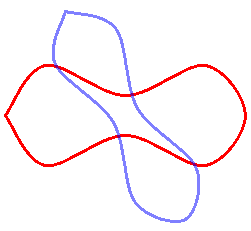
**الرسم والتلوين**

* [**قدرات مدهشة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\12-%20الرسم%20والتلوين\1-%20قدرات%20مدهشة.htm)
* [**التعامل مع الألوان:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\12-%20الرسم%20والتلوين\2-%20التعامل%20مع%20الألوان.htm)
* [**رسم الأشكال:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\12-%20الرسم%20والتلوين\3-%20الرسم.htm)
* [**رسم النصوص:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\12-%20الرسم%20والتلوين\3-%20رسم%20النصوص.htm)
* [**التعامل مع الصور:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\12-%20الرسم%20والتلوين\5-%20التعامل%20مع%20الصور.htm)
* [**رسم الدوال:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\12-%20الرسم%20والتلوين\6-%20رسم%20الدوالّ.htm)

**قدرات مدهشة:**



هل ترى هذه الرسوم الرائعة؟

هل تصدّق أنّك تستطيع أن ترسم مثلها في كود VB؟

إنّ إطار العمال .Net FrameWork يمنحك العديد من الكائنات التي تستطيع أن تستخدمها في VB لإنشاء تطبيقات رسوم رائعة.

وتنقسم الرسومات Graphics إلى نوعيّن:

1- الرسوم الرياضيّة Vector: وهي التي ترسمها باستخدام دوال VB، مثل دوال رسم الخطوط والمنحنيات التي تعتبر إحدى أنواع الرسوم الرياضيّة، مثل تلك التي رأيناها بأعلى.

2- الصور Bitmaps: حيث يمكنك عرضها في العديد من أدوات VB، كما يمكنك إجراء العمليّات عليها عن طريق معالجتها نقطة نقطة.

ولا غنى لك عن استخدام كلتا الوسيلتين، وإن كان استخدامك للصور سيكون أكثر تواترا.

ويمكنك الرسم في VB.NET، تقريبا على أيّ أداة تخطر ببالك، بما في ذلك مربّع النصّ والقائمة ListBox، وإن كان الأكثر شيوعا أن ترسم على النموذج ومربّع الصورة PictureBox.

ولمعظم الأدوات خاصيّة "الصورة" Image أو "صورة الخلفيّة" BackgroundImage، التي يمكنك تغيير قيمتها من نافذة الخصائص في وقت التصميم، أو بكتابة الكود الذي يغيّرهما في البرنامج.

**ملاحظة:**

عند وضع صورة في خلفيّة أيّ أداة أو نموذج في وقت التصميم، يتمّ نسخها من موقعها الأصليّ إلى ملفّات مشروعك، بحيث لا تقلق إذا تغيّر موقع برنامجك أو موقع الصورة الأصليّة.. أمّا لو استخدمت الكود في تحميل صورة، فإنّها تظلّ في مكانها الأصليّ.. لهذا يجب عليك تجنّب الأخطاء التي يمكن أن تحدث عند حذف هذه الصورة أو تغيّر موقعها.

**التعامل مع الألوان**

* [**كائن اللون Color Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\2-%20التعامل%20مع%20الألوان\1-%20كائن%20اللون.htm)
* [**تكوين الألوان بالطرق القديمة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\2-%20التعامل%20مع%20الألوان\2-%20تكوين%20الألوان%20بالطرق%20القديمة.htm)
* [**مربّع حوار اختيار اللون ColorDialog:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\2-%20التعامل%20مع%20الألوان\3-%20مربّع%20حوار%20اختيار%20اللون.htm)

**كائن اللون Color Object:**

يمكنك تعريف متغيّر من هذا السجلّ كما يلي:

**Dim myColor As Color**

**myColor = Color.Azure**

إنّ سجلّ الألوان Color Structure يمنحك 128 لونا بأسمائها الإنجليزيّة (ولا أدّعي أنّني أعرف باللغة العربيّة هذا العدد من أسماء الألوان!!)، مثل اللون اللازورديّ (الأزرق السماويّ) Azure.. ويمكنك أن تتعرّف على هذه الألوان بمجرّد كتابة:

**Color.**

حيث ستظهر لك قائمة بكلّ قيم المرقّم Color.

ومن أهمّ الألوان اللون الشفّاف Transparent، حيث يمكنك أن تستخدمه لجعل خلفيّة بعض الأدوات ـ كاللافتة ـ شفّافة:

**Lable1.BackColor = Color.Transparent**

**بالإضافة لهذا، فإنّ هذا السجلّ يمنحُك بعض الوسائل الهامّة، ومنها:**

**تكوين اللون من مركّباته FromARGB:**

يمكنك أن تكوّن ملايين الألوان باستخدام هذه الوسيلة، حيث ترسل لها نسب الأحمر Red والأخضر Green والأزرق Blue لتعيد لك اللون المكوّن منها.. (طبعا اتضح لك أنّ الحروف RGB هي الحروف الأولى من أسماء الألوان الثلاثة).

وفي المثال التالي نغيّر لون خلفيّة مربّع النصّ باستخدام دالة تكوين اللون:

**TextBox1.BackColor = Color.FromARGB(25,150,255)**

وهنا يجب أن تلاحظ شيئين:

1- أنّ كلّ مكوّن من المكوّنات الثلاثة تنحصر قيمته بين 0 و 255، فإذا أرسلت للمعاملات أيّ عدد أكبر من هذا فسيكون تأثيره كتأثير العدد 255.

2- أنّ معكوس أيّ لون (نيجاتيف اللون) ينتج بطرح كلّ مكوّن من مكوّناته الثلاثة من 255، وسيتّضح لك ذلك عندما تعرف أنّ مكونات اللون الأبيض هي (255، 255، 255) ومكونات اللون الأسود هي (0، 0، 0).

والجدول التالي يوضّح لك مكوّنات بعض الألوان الهامّة:

|  |  |  |  |
| --- | --- | --- | --- |
| اللون | نسبة الأحمر  R | نسبة الأخضر  G | نسبة الأزرق  B |
| الأسود | 0 | 0 | 0 |
| الأزرق | 0 | 0 | 255 |
| الأخضر | 0 | 255 | 0 |
| السماوي | 0 | 255 | 255 |
| الأحمر | 255 | 0 | 0 |
| الأحمر البنفسجيّ | 255 | 0 | 255 |
| الأصفر | 255 | 255 | 0 |
| الأبيض | 255 | 255 | 255 |
| الرمادي | 128 | 128 | 128 |

وإذا أردت الحصول على لون فاتح من الألوان الماضية، فاستبدل العدد 128 بالعدد 255 في كلّ مكوّنات اللون.

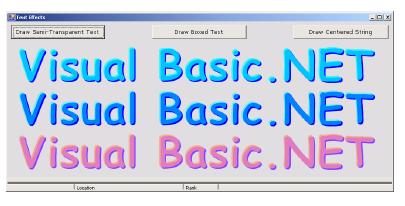
وهناك صيغة أخرى من هذه الوسيلة، تسمح لك بتحديد درجة شفافية اللون:

**FromARGB(شفافية اللون, نسبة الأحمر, نسبة الأخضر, نسبة الأزرق)**

حيث تتراوح الشفافية ما بين 0 (شفافية تامّة) و 255 (إعتام تامّ).

ويسمّى معامل الشفافيّة "خليط ألفا" Alpha Blinding.. ولا أعرف تحديدا سبب وجود الحرف الأبجدي اللاتيني ألفا في هذا الاسم، وإن كنت أخمّن أنّه اسم الخوارزميّة Algorithm التي تقوم بخلط لون الكائن مع لون ما خلفه، لإعطاء إحساس الشفافيّة.. إنّ الكثير من الخوارزميّات البرمجيّة تحمل أسماء حروف لاتينية ومعاملات ومتغيّرات تدخل في كتابتها، كالخوارزميّة K-means مثلا.. والله أعلم.

ومن أروع استخدامات الشفافية، استخدامها لكتابة نصوص تبدو للرائي مجسّمة، وذلك برسم النصّ بلون ما (سنعرف كيفيّة ذلك لاحقا)، ثمّ رسم نفس النصّ بلون آخر وجعله شبه شفّاف، على أن يكون النصّ في المرّة الثانية مزاحا قليلا رأسيّا وأفقيّا.. ويمكنك أن ترى هذه الطريقة في تطبيق TextEffects.



كما يمكنك استخدام الشفافية في رسم علامة مائيّة على الصورة عند نشرها على الإنترنت.. هذه العلامة المائيّة لن ترهق العين، ولكنّها ستعوق الآخرين عن استخدام هذه الصورة بدون تصريح منك.

والكود التالي يكتب جملة "MySite.com" على النموذج نصف شفّافة.

**Dim F As New Font("Comic Sans MS", 20, FontStyle.Bold)**

**Dim B As New SolidBrush(Color.FromARGB(50, 230, 80, 120))**

**Me.CreateGraphics.DrawString("MySite.com", F, B, 100, 40)**



**تكوين لون معروف FromKnownColor:**

تسمح لك هذه الوسيلة بتكوين اللون من قيم المرقّم KnownColor.. وهذه القيم تندرج تحت طائفتين:

- 128 لونا بأسمائها الإنجليزيّة المعروفة، مثل KnownColor.Red .

- الألوان الخاصّة بنظام الويندوز (والتي يغيّرها المستخدم من خصائص سطح المكتب لتكون عامّة للويندوز)، مثل لون العناوين KnownColor.ActiveCaption، ولون نصوص العناوين KnownColor.ActiveCaptionText.. إلخ.

**GetBrightness:**

للحصول على درجة إضاءة اللون.

**GetHue:**

للحصول على درجة تدرّج اللون.

**GetSaturation:**

للحصول على درجة تشبّع اللون.

**ToArgb:**

استخدم هذه الوسيلة للحصول على العدد الدالّ على اللون (وهو يتكوّن من 4 وحدات 4 Bytes)، حتّى تستطيع استخدامه مع التطبيقات الأخرى التي تتعامل مع الألوان كأرقام وليس ككائنات.. ومن أطرف استخدامات هذه الوسيلة، استخدامها لمقارنة لونين معا.. إنّ الجملة التالية مرفوضة برمجيّا:

**If Me.BackColor = Color.Red Then**

وذلك لأنّ اللون سجلّ Structure، ولا يمكن التأكّد من تساوي سجلين أو كائنين بعمليّة تساوي عاديّة "=".. في حالة الكائنات المرجعيّة نستخدم المعامل Is بدلا من العلامة "=".. ولكنّ السجلات كائنات قيميّة وليست مرجعيّة.. لهذا لا يمكن استخدام أيّ من المعاملين "=" أو Is.

فما العمل إذن؟

يمكنك استخدام الوسيلة Equals كالتالي:

**If Me.BackColor.Equals(Color.Red) Then**

أو يمكنك استخدام أيّ وسيلة تحوّل اللون من سجلّ إلى رقم أو نصّ، كالتالي:

**If Me.BackColor.ToArgb = Color.Red.ToArgb Then**

أو:

**If Me.BackColor.ToString = Color.Red.ToString Then**

**الخصائص:**

**درجة شفافية اللون A ـ نسبة الأحمر R ـ نسبة الأخضر G ـ نسبة الأزرق B.**

وأعتقد أنّها غنيّة عن التعريف.

والدالة التالية تعيد لك معكوس اللون الذي ترسله لها، مع المحافظة على درجة شفافيته كما هي:

**Function RevColor(Clr As Color) As Color**

**Return Color.FromARGB(Clr.A, 255 - Clr.R, \_**

**255 - Clr.G, 255 - Clr.B)**

**End Function**

**تكوين الألوان بالطرق القديمة:**

وقبلَ أن ننهيَ مناقشتنا لكائن اللون، يجب أن أخبرك أنّ هناك دالتين حفريّتين باقيتين من VB6 ما زال بإمكانك استخدامهما إلى الآن لتكوين الألوان.. هاتان الدالتان هما:

**QBColor:**

تتيح لك هذه الدالة الحصول على 16 لونا معرّفا، على الصيغة التالية:

**Dim Clr As Integer = QBColor(لون)**

حيث يأخذ المعامل عددا صحيحا من 0 إلى 15، يمثّل أحد الألوان.. والجدول التالي يبيّن لك الأعداد وما تدلّ عليه من ألوان:

|  |  |
| --- | --- |
| 0 | أسود |
| 1 | أزرق |
| 2 | أخضر |
| 3 | سماوي |
| 4 | أحمر |
| 5 | أحمر بنفسجي |
| 6 | أصفر |
| 7 | أبيض |
| 8 | رمادي |
| 9 | أزرق فاتح |
| 10 | أخضر فاتح |
| 11 | سماوي فاتح |
| 12 | أحمر فاتح |
| 13 | بنفسجي محمرّ فاتح |
| 14 | أصفر فاتح |
| 15 | أبيض ناصع |

ولا أعتقد أنّ نفسك ستساورك لاستخدام هذه الدالة يوما، فهي تحتوي على نقيصتين خطيرتين:

1- معاملها رقميّ وليس مرقّما يمنحك أسماء الألوان، ممّا يعني أنّ عليك حفظ أرقام الألوان أو الرجوع لملفّات الإرشاد في كلّ مرّة تريد فيها استخدام هذه الدالة.

2- القيمة العائدة من هذه الدالة عبارة عن رقم صحيح يمثّل اللون، وستحتاج لتحويله إلى كائن من النوع Color حتّى تستطيع استخدامه.

أعتقد أنّك ستجد استخدام سجلّ الألوان Color Structure أفضلَ وأسهل.

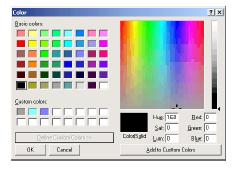
**RGB:**

هذه الدالة مماثلة لإحدى صيغتي الوسيلة Color.ARGB()، فهي تتلقّى منك نسب الألوان الأساسيّة (الأحمر، الأخضر، الأزرق)، وتعيد رقما صحيحا يمثّل اللون المركّب من هذه النسب.. ولا أنصحك باستخدامها هي الأخرى، فستحتاج لتحويل العدد الذي تعيده لك إلى كائن من النوع Color حتّى تستطيع استخدامه، وهو ما ستريحك الدالة Color.ARGB() من الاحتياج إليه.

كان هذا فقط للعلم بالشيء.

**مربّع حوار اختيار اللون ColorDialog:**

كثيرا ما تريد أن تمنح المستخدم الحرّيّة في اختيار ألوان خلفيّات النوافذ والأدوات.. في هذه الحالة لا بدّ أن تعرض مربّع حوار اختيار الألوان، ليختار منه المستخدم اللون الذي يناسبه.. ولحسن حظّك فإنّ VB يمنحك هذا المربّع جاهزا، وستجده في صندوق الأدوات باسم ColorDialog، حيث يمكنك أن تضيفه للنموذج كأيّ أداة أخرى.



تعال نرى خصائص مربّع الحوار هذا:

**السماح بفتح كامل AllowFullOpen:**

لمربّع حوار الألوان زرّ بعنوان "تعريف ألوان مخصّصة" Define Custom Colors.. لو جعلت هذه الخاصيّة False فسيكون هذا الزر غير فعّال ولن يتمكّن المستخدم من ضغطه، فإذا جعلت للخاصية AllowFullOpen القيمة True، فسيتمكّن المستخدم من ضغط هذا الزرّ، حيث سيظهر نموذج آخر، يستطيع المستخدم من خلاله تركيب أيّ لون يريده.

**أيّ لون AnyColor:**

لو جعلت هذه الخاصيّة True، فسيتمّ عرض جميع الألوان الأساسيّة المتاحة في مربّع حوار الألوان.

**اللون Color:**

هذه الخاصيّة تسمح لك باختيار اللون الذي سيكون محددا عندما يفتح المستخدم مربّع الحوار.. إنّك تحتاج لفعل هذا حتّى تعرض للمستخدم اللون الحالي للأداة التي سيغيّر لونها (لون خلفيّة مربّع النصّ مثلا)، أو لكي تعرض له آخر لون اختاره في المرّة السابقة.

كما أنّ هذه الخاصيّة تسمح لك بمعرفة اللون الذي اختاره المستخدم عند إغلاق مربّع الحوار.. انظر للمثال التالي:

**' اللون الحاليّ لخلفيّة النموذج هو الذي سيكون محددا عند فتح مربع الحوار**

**ColorDialog1.Color = Me. BackColor**

**' عرض مربّع الحوار**

**' ويجب التأكّد من أنّ المستخدم قد ضغط موافق وليس إلغاء**

**If ColorDialog1.ShowDialog = DialogResult.OK Then**

**' في هذه الحالة نجعل خلفيّة النموذج باللون الذي اختاره المستخدم**

**Me.BackColor = ColorDialog1.Color**

**End If**

**ألوان مخصّصة CustomColors:**

في الجزء السفليّ من مربّع حوار الألوان منطقة يمكنك فيها وضع 16 لونا خاصّا بك.. تضعها كلّها أو بعضها.. ولكي تفعل ذلك، عرّف مصفوفة من الأعداد الصحيحة، وضع فيها الأرقام الدالة على الألوان التي تريد ظهورها في المنطقة المخصّصة، وضعها في هذه الخاصيّة:

**Dim colors() As Integer = {222663, 35453, 7888}**

**ColorDialog1.CustomColors = colors**

ونظرا لاستحالة حفظ أرقام الألوان، استخدم كائن الألوان لتعريف الألوان بأسماء واضحة، ثمّ حوّلها لأعداد صحيحة باستخدام الوسيلة ToARGB.. ونظرا لأنّ أوّل وحدة Byte على يسار اللون مخصّصة للشفافية، فإنّ ذلك قد يؤدّى لأرقام سالبة.. لهذا استخدم دالة العدد المطلق من خليّة الدوالّ الرياضيّة Math.Abs للتأكّد من أنّ كلّ الأرقام موجبة:

**Dim colors() As Integer = {Math.Abs(Color.Gray.ToARGB), \_**

**Math.Abs(Color.Navy.ToARGB), Math.Abs(Color.Teal.ToARGB)}**

أو من الأسهل، أن تستخدم دالة VB القديمة RGB.. ها هي ذي فائدة تتّضح لهذه الدالة الحفريّة!

**الألوان المتجانسة فقط SolidColorOnly:**

اجعل هذه الخاصيّة True إذا كان برنامجك سيعمل على نظام يعرض 256 لونا فقط.

وللتدريب على استخدام هذه الأداة، افحص الأمر Page Color والأمر Text Color في القائمة Customize في مشروع TxtPad في مجلّد برامج الفصل السادس.

**رسم الأشكال الرياضيّة:**

* [**الرسم بواجهة تصميم الرسومات GDI+:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\2-%20الرسم%20بواجهة%20تصميم%20الرسومات.htm)
* [**كيف ترسم:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\3-%20كيف%20ترسم.htm)
* [**أهمّ الكائنات المستخدمة في الرسم:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم.htm)
* [**رسم الأشكال بوسائل كائن الرسوم:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\5-%20رسم%20الأشكال%20بوسائل%20كائن%20الرسوم.htm)
* [**كائن مسار الرسوم GraphicsPath Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\6-%20كائن%20مسار%20الرسوم.htm)
* [**الألوان المتدرّجة Gradients:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\7-%20الألوان%20المتدرّجة.htm)
* [**تحويل الإحداثيّات Coordinate Transformations:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\8-%20تحويل%20الإحداثيّات.htm)
* [**قص منطقة الرسم Clipping:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\3-%20الرسم\9-%20قص%20منطقة%20الرسم.htm)

**الرسم بواجهة تصميم الرسومات Graphics Design Interface (GDI+):**

إنّ GDI هي مجموعة من الخلايا التي تمكّنك من إنشاء الرسوم والنصوص والصور، وهي بمثابة المحرّك الذي يستخدمه الويندوز نفسه في هذه العمليّات.. ولقد تطوّر GDI عبر الزمن، حتّى جاءت أحدث نسخة منه مع VS.Net تحمل اسم GDI+.

ونظرا لقوّة وثراء وسرعة أداء هذا المحرّك، فلقد ذهبت كل وسائل الرسم في VB6 بلا رجعة، ولم يعد بإمكانك إلا استخدام GDI+.

وتقع خلايا هذا المحرّك في الفضاءات التالية، التي يجب عليك استيراد بعضها على الأقلّ قبل استخدامها في مشروعك:

**System.Drawing**

**System.Drawing.Drawing2D**

**System.Drawing.Imaging**

**System.Drawing.Text**

**كيف ترسم:**

لكي ترسم أيّ شيء يجب أن يتوافر لديك ما يلي:

**سطح الرسم Surface:**

يجب عليك قبل أن تبدأ الرسم، أن تختار السطح الذي سترسم عليه.. هذا السطح يمثّله كائن الرسوم Graphics object، الذي يمدّك بالوسائل التي ترسم الأشكال الأساسيّة وغير الأساسيّة.

وللرسم على نموذج أو أداة، يجب أن نتعامل مع كائن الرسوم الخاصّ بها، والذي تحصل عليه من الوسيلة "إنشاء رسوم" CreateGraphics.. كما يمكن الرسم على كائن صورة نقطيّة Bitmap Object، ووضع الصورة الناتجة بعد ذلك على النموذج أو الأداة، كما سنرى فيما بعد.

**أداة الرسم:**

لدينا أداتان رئيسيّتان نرسم بهما:

**1. القلم Pen:**

ويمكنك استخدامه في رسم الأشكال المكوّنة من خطوط، مثل الخطوط lines، والمستطيلات rectangles والمنحنيات curves.. وأهمّ خصائص القلم: لونه وسمك خطّه.

**2. الفرشاة Brush:**

ويمكنك استخدامها في رسم المساحات الملوّنة.. وأهمّ خصائص الفرشاة: لونها والشكل الذي ستلوّنه وطريقة التلوين.

والمثال البسيط التالي يوضّح لك كيف ترسم خطّا على النموذج:

**' تجهيز قلم أحمر سمك خطّه نقطتان**

**Dim redPen As Pen = New Pen(Color.Red, 2)**

**' نقطة بداية الخطّ**

**Dim point1 As Point = New Point(10,10)**

**' نقطة نهاية الخطّ**

**Dim point2 As Point = New Point(120,180)**

**' رسم الخطّ على النموذج**

**Me.CreateGraphics.DrawLine(redPen, point1, point2)**

ويمكن دمج كلّ الخطوات الأربع السابقة في الخطوة الوحيدة التالية:

**Me.CreateGraphics.DrawLine(New Pen(Color.Red, 2), \_**

**New Point(10,10), New Point(120,180))**

وهناك صيغة أسهل للوسيلة DrawLine كالتالي:

**Me.CreateGraphics.DrawLine(New Pen(Color.Red, 2), 10, 10, 120, 180)**

حيث أرسلنا إحداثيات الخطّ منفردة، بدلا من إرسالها كنقطتي بداية ونهاية، وإنّ كانت الصيغة الأولى أوضح عند قراءتها من الصيغة الثانية.

كما يمكنك تعريف كائن رسوم ثمّ ربطه بكائن رسوم النموذج كالتالي:

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

**G.DrawLine(redPen, point1, point2)**

**أهمّ الكائنات المستخدمة في الرسم**

تعتبر الكائنات التالية، كائنات أوّليّة في عمليّة الرسم:

* [**كائن النقطة Point Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم\1-%20كائن%20النقطة.htm)
* [**كائن الحجم Size Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم\2-%20كائن%20الحجم.htm)
* [**كائن المستطيل Rectangle Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم\3-%20كائن%20المستطيل.htm)
* [**كائن القلم Pen Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم\4-%20كائن%20القلم.htm)
* [**كائن الفرشاة Brush Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم\5-%20كائن%20الفرشاة.htm)
* [**كائن الرسوم Graphics Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\4-%20أهمّ%20الكائنات%20المستخدمة%20في%20الرسم\6-%20كائن%20الرسوم.htm)

**كائن النقطة Point Object:**

يمثّل هذه الكائن إحداثي نقطة: الإحداثيّ السينيّ X (موضع النقطة أفقيّا، بدئا من أقصى يسار الشاشة)، والإحداثيّ الصادي Y (موضع النقطة رأسيّا، بدئا من أعلى الشاشة).

ولإنشاء نقطة جديدة، يجب أن تحدد إحداثييها (الأفقيّ ثمّ الرأسيّ) كالتالي:

**Dim P1 As New Point**

**P1.X = 34**

**P1.Y = 50**

أو باختصار:

**Dim P1 As Point**

**P1 = New Point(34, 50)**

أو بشكل أكثر اختصارا:

**Dim P1 As New Point(34 50)**

ويقبل هذا الكائن الإحداثيات في صورة أعداد صحيحة Integer، فإذا أردت استخدام وحدة قياس غير الوحدة Pixel، فقد تكون قيم الإحداثيات أعدادا مفردة Single.. في هذه الحالة استخدم الكائن PointF، وهو مماثل لكائن النقطة Point، إلا إنّه يقبل الإحداثيات من النوع Single أو Double.. وبالمناسبة: الحرف F في اسم هذا الكائن هو اختصار لتعبير "علامة عشريّة" Floating-point.

**كائن الحجم Size Object:**

مشابه لكائن النقطة، ولكنّه يمتلك خاصيتين: العرض Width، والارتفاع Height، ويتمّ تعريفه كالتالي:

**Dim S As Size**

**S.Width = 100**

**S.Height = 20**

أو بطريقة أخرى:

**Dim S As New Size(100,20)**

وبطريقة ثالثة:

**Dim S As New Size(New Point(100,20))**

وهناك نوع آخر من كائن الحجم، هو كائن الحجم ذو الدقّة العشريّة SizeF، وهو مماثل لكائن الحجم Size، إلا إنّه يقبل معاملات من النوع Single و Double.

**كائن المستطيل Rectangle Object:**

يتطلّب تعريف المستطيل، موقع رأسه العلويّ الأيمن وعرضه وارتفاعه:

**Dim box As New Rectangle(X, Y, العرض, الارتفاع)**

وهناك صيغة أخرى باستخدام كائن النقطة وكائن الحجم:

**Dim P As New Point(1,1)**

**Dim S As New Size(100,20)**

**Dim box As New Rectangle(P, S)**

كما يمكنك أن تنشئ مستطيلا، ثمّ تضع قيم خصائصه كالتالي:

**Dim box As New Rectangle()**

**box.X = 1**

**box.Y = 1**

**box.Width = 100**

**box.Height = 20**

وهناك نوع آخر من كائن المستطيل، هو كائن المستطيل ذو الدقّة العشريّة RectangleF، وهو مماثل للمستطيل Rectangle، إلا إنّه يقبل معاملات من النوع Single و Double.

**كائن القلم Pen Object:**

ذكرنا من قبل أنّ تعريف هذا الكائن يتطلّب اللون والسمك (ولو حذفت السمك من التعريف فسيتمّ اعتباره 1):

**Dim P As Pen**

**P = New Pen(Color.Black, 3)**

وهناك صيغة أخرى تسمح لك بتعريف اللون من كائن فرشاة كالتالي:

**Dim patternPen As New Pen(brush, width)**

وفي هذه الحالة يمكنك استخدام هذا القلم لتلوين الحدّ الخارجيّ لشكل بطريقة معيّنة (تحدّدها الفرشاة).

ولديك مجموعة من الأقلام الجاهزة محدّدة الألوان وسمكها 1، يمنحها لك مرقّم الأقلام Pens Structure.. مثل القلم الأزرق Pens.Blue.. ويمكنك أن تستخدم هذه الأقلام مباشرة بدون تعريف، كمعامل في أيّ وسيلة تتطلّب قلما.

**خصائص القلم:**

**اتصّال الخط LineJoin:**

تحدّد الطريقة التي ستلتحم بها الأشكال المتجاورة.

**StartCap, EndCap:**

تحددان شكل بداية ونهاية الخطّوط المرسومة.

**DashCap**

تحدّد شكل بداية ونهاية التقطيعات في الخطّ المتقطّع.

**طراز التقطيع DashStyle:**

استخدم هذه الخاصيّة لرسم الخطّ متقطّعا بالطراز الذي تريده.. وتأخذ هذه الخاصيّة قيمة من قيم المرقّم DashStyle وهي:

متّصل **Solid** ـ شرطة **Dash** ـ منقّط **Dot** ـ شرطة ونقطة **DashDot** ـ شرطة ونقطتان **DashDotDot** ـ مخصّص **Custom**.

**نوع القلم PenType:**

تحدّد نوع القلم، الذي هو أحد القيم التالية:

ملء مظلّل **HatchFilled** ـ لون متدرّج **LinearGradient** ـ لون متشعّب التدرّج **PathGradient** ـ لون ثابت **SolidColor** ـ ملء بخامة **TextureFill**.

ستتساءل هنا، لماذا تتكلّم هذه القيم عن ملء الشكل باللون، مع أنّها خاصّة بالقلم وليس بالفرشاة؟

لا تنسَ أنّ القلم يرسم خطوطا، يمكن أن تكون سميكة.. ونحن نتكلّم عن كيفيّة ملء هذه الخطوط السميكة بالألوان.

**كائن الفرشاة Brush Object:**

تمكّنك الفرشاة Brush من ملء الأشكال بلون واحد أو بألوان متدرّجة أو بصورة.

ولا يمكنك تعريف فرشاة منفردة، فهذه الخليّة ترثها أنواع الفرش المختلفة، وهي:

**الفرشاة الصلبة SolidBrush:**

تملأ الشكل بلون واحد.

**Dim sBrush As New SolidBrush(لون الفرشاة)**

وسنتعرّف عليها بالتفصيل لاحقا.

**فرشاة التظليل HatchBrush:**

تملأ الشكل بتشكيلة معيّنة، من أكثر من 50 تشكيلة جاهزة، يمكنك اختيارها من خلال المرقّم HatchStyle.. ويتمّ تعريف هذه الفرشاة كما يلي:

**Dim hBrush As New HatchBrush (طراز الفرشاة, لون خطوط التظليل, لون الخلفيّة)**

وهذه بعض تشكيلات التظليل، التي يمكنك استخدامها كقيم للمعامل الأوّل:

|  |  |
| --- | --- |
| ForwardDiagonal | خطوط قطريّة من أعلى اليسار إلى أسفل اليمين. |
| BackwardDiagonal | خطوط قطريّة من أعلى اليمين إلى أسفل اليسار. |
| Cross | خطوط أفقيّة ورأسيّة متقاطعة. |
| DiagonalCross | خطوط قطريّة متقاطعة. |
| Horizontal | خطوط أفقيّة. |
| Vertical | خطوط رأسيّة. |

ويمكنك استعراض هذه التشكيلات المختلفة، في تطبيقي HatchBrushEnum و HatchBruchText في مجلّد برامج هذا الفصل.



**الفرشاة المتدرّجة LinearGradientBrush:**

تملأ الشكل بألوان متدرّجة.. ويبدأ التدرّج من لون ما وينتهي إلى لون آخر.

وسنتعرّف عليها بالتفصيل [لاحقا](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\7-%20الألوان%20المتدرّجة\1-%20المتدرّجات%20الخطّيّة.htm).

**فرشاة متشعّبة التدرّج PathGradientBrush:**

تملأ مسار الشكل بألوان متشعّبة التدرّج، حيث يبدأ التدرّج بلون واحد، ولكنّه ينتهي إلى ألوان مختلفة.

وسنتعرّف عليها بالتفصيل [لاحقا](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\7-%20الألوان%20المتدرّجة\2-%20المتدرّجات%20المتشعّبة.htm).

**فرشاة الخامة TextureBrush:**

تملأ الشكل بصورة.. ويتمّ تبليط Tiling الشكل بالصورة ليملأها كلّيا، بمعنى أنّ الصورة تتكرّر على حسب ما يتطلّب الأمر.. تخيّل مدى روعة الشكل الناتج من ملء دائرة أو منحنى بصورة معيّنة!

**ملحوظة1:**

في أيّ نوع من أنواع الفرش السابقة (والأقلام ومسار الرسوم كذلك) يمكن استخدام معامل الشفافية في تكوين الألوان المرسلة كمعاملات لهذه الخلايا، وبهذا يمكنك الحصول على أشكال شفافة بأيّ نسبة تريدها.. فمثلا: لو لوّنت أيّ شكل بالفرشاة التالية، فسيكتسب لونا أحمر نصف شفّاف، حيث سيمتزج لونه بلون خلفيّته:

**Dim sBrush As SolidBrush**

**sBrush = New SolidBrush(Color.FromARGB(128, 255, 0, 0))**

**ملحوظة2:**

يمنحك سجلّ الفرش Bruches Structure مجموعة من الفرش الجاهزة بألوان مختلفة.. مثل Brushes.Red التي تمنحك فرشاة تلوين حمراء.

**كائن الرسوم Graphics Object:**

أدوات كثيرة لا تتوقّعها تمنحك كائن رسوم، ممّا يمكّنك من الرسم عليها.. خذ مربّع النصّ مثلا!

ولكي تحصل على كائن الرسوم من الأداة، استخدم وسيلتها CreateGraphics.

ولكي تستخدم كائن الرسوم، استورد المكتبة Drawing2D في مشروعك أولا:

**Imports System.Drawing.Drawing2D**

بعد ذلك يمكنك تعريف متغيّر من هذا الكائن كالتالي:

**Dim G As Graphics**

ولكي ترسم بهذا الكائن على مربّع الصورة، استخدم الجملة التالية:

**G = PictureBox1.CreateGraphics**

**ملحوظة:**

يرتبط كائن الرسوم عند إنشائه بمساحة سطح النموذج أو الأداة، فإذا تغيّرت هذه المساحة لا يشعر كائن الرسوم بهذا التغيير.. لهذا فإنّ أنسب مكان تعرّف فيه هذا الكائن بالنسبة للنموذج هو حدث رسم النموذج Paint event، حتّى يعاد إنشاء كائن الرسوم كلّما حدث تغيير للنموذج يستدعي إعادة رسمه.

**خصائص كائن الرسوم:**

**DpiX و DpiY:**

تمثّل هاتان الخاصيتان عدد النقاط في كلّ بوصة Dots per inch (Dpi) أفقيّا (على المحور X) ورأسيّا (على المحور Y).

**وحدات الصفحة PageUnit:**

في الوضع التلقائيّ، تقاس كلّ الأطوال بالنقطة Pixel.. فإذا أردت تغيير ذلك، فاختر لهذه الخاصيّة أيّ وحدة قياس أخرى من المرقّم GraphicsUnit، الذي قيمه كالتالي:

|  |  |
| --- | --- |
| Document | 1.3 بوصة. |
| Inch | بوصة. |
| Millimeter | ملليمتر. |
| Pixel | نقطة على شاشة الكمبيوتر (وهي القيمة الافتراضيّة). |
| Point | نقطة في ورقة الطباعة، وهي تساوي 1.72 بوصة. |
| World | حدّد أنت القيمة التي تريد القياس بها. |

**كيفية رسم النصوص TextRenderingHint:**

تأخذ هذه الوسيلة واحدة من قيم المرقّم TextRenderingHint، لإضافة تأثيرات معيّنة تحسّن من شكل النصّ المرسوم.

**طراز التنعيم SmoothingMode:**

مماثلة للخاصيّة السابقة، ولكنّها تنطبق على كلّ الأشكال، وليس النصوص فقط.. وهي تأخذ قيمة من قيم المرقّم SmoothingMode.

**رسم الأشكال بوسائل كائن الرسوم:**

يمتلك كائن الرسوم العديد من الوسائل التي تمكّنك من رسم الأشكال.. ولكي نشرح هذه الوسائل، تعالَ نعرّف كائن رسوم يرسم على النموذج بقلم أزرق:

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

**Dim P As New Pen(Color.Blue)**

إنّ احتياجنا للقلم يرجع إلى أنّ أوّل معامل في وسائل رسم الأشكال هو القلم.. وطبعا باقي المعاملات هي إحداثيات الشكل.

دعنا نبدأ برسم مستطيل:

**G.DrawRectangle(P, 10, 10, 200, 150)**

والآن سنرسم قطري المستطيل:

**G.DrawLine(P, 10, 10, 210, 160)**

**G.DrawLine(P, 210, 10, 10, 160)**

جرّب ذلك في مشروع SimpleShapes.

والجدول التالي يوضّح لك الوسائل التي ترسم الأشكال المختلفة (تلك التي تبدأ بالمقطع Draw)، والوسائل التي تملأ كل شكل منها بلون معيّن ـ إذا كان الشكل مغلقا بالطبع ـ (وهي تبدأ بالمقطع Fill)، مع ملاحظة أنّ استخدام وسيلة الملء يملأ الشكل إذا كان موجودا على النموذج، أو ترسمه ممتلئا باللون إذا لم يكن موجودا من قبل.

|  |  |
| --- | --- |
| DrawArc | ترسم قوسا. |
| DrawBezier | ترسم منحنى بيزير، وهو منحنى له نقطتا بداية ونهاية، ويمرّ بنقطتين أخريين تتحكّمان في انحنائه، كما في الشكل: |
| DrawBeziers | ترسم مجموعة من منحنيات بيزير معا. |
| DrawClosedCurve  FillClosedCurve | ترسم منحنى مغلق. |
| DrawCurve | ترسم منحنى يمرّ بنقاط معيّنة (على الأقلّ 4 نقاط، بداية ونهاية ونقطتان على المنحنى):  **Graphics.DrawCurve(القلم, \_**  **مصفوفة من النقاط التي يمرّ بها المنحنى, درجة الانحناء)**  ولو لم ترسل المعامل الأخير، فسيتمّ رسم المنحنى بدرجة انحناء 1.  والشكل التالي يبيّن مجموعة من المنحنيات التي تمرّ بنفس النقاط، ولكنّ لكلّ منها درجة انحناء مختلفة: |
| DrawEllipse  FillEllipse | ترسم قطعا ناقصا، بمعلوميّة المستطيل الذي يحتويه.. والقطع الناقص هو شكل بيضاويّ له قطران غير متساويين (مساويان لضلعي المستطيل الذي يحتويه).    ونظرا لأنّه لا توجد وسيلة مستقلّة لرسم الدائرة، فيجب عليك أن تستخدم هذه الوسيلة لرسم الدوائر، وذلك برسم القطع الناقص داخل مربّع، وفي هذه الحالة سيكون طول ضلع المربّع مساويا لقطر الدائرة.. مثال: الجملة التالية ترسم دائرة زرقاء نصف قطرها 50:  **Graphics.DrawEllipse(Pens.Blue, \_**  **New Rectangle(50,20,100,100))**  ويقع مركز هذه الدائرة في مركز المربّع الذي هو (50 + [100] ÷ 2 ، 20 + [100] ÷ 2)، أي النقطة (100 ، 70).  وكقاعدة:  يقع مركز الدائرة عند النقطة (س + نصف طول ضلع المربّع ، ص + نصف طول ضلع المربّع). |
| DrawIcon | تحمّل أيقونة داخل كائن الرسوم.. هذه الأيقونة سيتمّ مطّها لتملأ مساحة كائن الرسوم. |
| DrawIconUnstretched | تحمّل أيقونة داخل كائن الرسوم، ولكن دون مطّها. |
| DrawImage | تحمّل صورة داخل كائن الرسم، مع مطّها لتملأ مساحة كائن الرسوم. |
| DrawImageUnscaled | تحمّل صورة داخل كائن الرسم، ولكن دون تغيير حجمها. |
| DrawLine | ترسم خطّا يصل بين نقطتين. |
| DrawLines | ترسم مجموعة من الخطوط. |
| DrawPath  FillPath | ترسم جميع الأشكال الموجودة في كائن مسار الرسوم GraphicsPath object.. وسنتعرّف عليه [لاحقا](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\6-%20كائن%20مسار%20الرسوم.htm). |
| DrawPie  FillPie | ترسم شريحة من الدائرة (قوس من قطع ناقص (أو دائرة) وضلعين يصلان نهايتي القوس بمركز الدائرة).. ولرسم هذا القوس أرسل لهذه الوسيلة المستطيل الذي يحتوي على القطع الناقص، مع زاوية بداية القوس، وزاوية انفراج ضلعي القوس.. مثال:  **Graphics.DrawEllipse(Pens.Blue, \_**  **New Rectangle(50,20,100,300), 50, 30)**  وأهمّ استخدام لهذا الشكل هو الرسوم البيانيّة التي توضّح النسب كشرائح ملوّنة من دائرة (مثل الرسم الذي يوضّح لك المساحة المشغولة والمساحة الفارغة من القرص الصلب في الويندوز). |
| DrawPolygon  FillPolygon | ترسم مضلّعا مغلقا له أيّ عدد من الرءوس.. هذه الرءوس ترسلها لهذه الوسيلة في صورة مصفوفة من النقط. |
| DrawRectangle  FillRectangle | ترسم مستطيلا. |
| DrawRectangles  FillRectangles | ترسم مجموعة من المستطيلات. |
| DrawString | ترسم نصّا بخطّ معيّن. |
| FillRegion | لملء كائن من النوع "منطقة" Region بلون معيّن. |

ولن تجد صعوبة في استخدام هذه الوسائل.. فقط لاحظ ما يلي:

- أوّل معامل في دوال رسم الأشكال هو كائن القلم Pen Object، وذلك لتحديد لون الخطّ وسمكه.

- أوّل معامل في دوال ملء وتلوين الأشكال المغلقة هو كائن الفرشاة Prush Object، وذلك لتحديد لون التلوين وطرازه.

- باقي معاملات وسائل الرسم والتلوين توضّح موضع الشكل ومقاييسه.

- الوسائل التي ترسم أكثر من شكل في نفس الوقت، تتطلّب ـ كمعامل ـ مصفوفة تعرّف مواضع ومقاييس هذه الأشكال.. فمثلا DrawRectangles تقبل مصفوفة تحتوي على كائنات من النوع "مستطيل" Rectangle objects.

**كائن مسار الرسوم GraphicsPath Object:**

يمثّل هذا الكائن رسما مغلقا يتكوّن من مجموعة من الخطوط والمستطيلات والمنحنيات.

وأبسط طرية لتكوين هذا الكائن، هي تعريف نسخة جديدة منه، واستخدام وسائله التالية لإضافة الرسومات إليه:

إضافة قوس **AddArc** ـ إضافة قطع ناقص **AddEllipse** ـ إضافة مضلّع **AddPolygon** ـ إضافة منحنى بيزير التكعيبيّ **AddBezier** ـ إضافة خطّ **AddLine** ـ إضافة مستطيل **AddRectangle** ـ إضافة منحنى **AddCurve** ـ إضافة منحنى مغلق **AddClosedCurve** ـ إضافة شريحة من دائرة **AddPie** ـ إضافة نصّ **AddString**.

وهناك المزيد من الوسائل تسمح لك بإضافة عدد من الأشكال دفعة واحدة، مثل:

إضافة منحنيات بيزير **AddBeziers** ـ إضافة خطوط **AddLines** ـ إضافة مستطيلات **AddRectangles**.

ليس هذا فحسب، بل يمكنك إضافة مسار رسومات آخر إلى المسار الحاليّ باستخدام الوسيلة **AddPath**.

طبعا لاحظت أنّ أسماء هذه الوسائل تتشابه مع أسماء وسائل الرسم بكائن الرسوم Graphics.. بل إنّهما يتشابهان في المعاملات.. والاختلاف الوحيد بينهما، هي أنّ وسائل الرسم في كائن الرسوم لها معامل زائد، هو المعامل الأوّل، الذي تحدّد فيه القلم الذي سترسم به.. ونظرا لأنّ كائن مسار الرسوم لا يرسم هذه الرسوم على النموذج أو الأدوات بنفسه، فإنّك لا تحتاج لتحديد القلم له.. ولرسم الأشكال الموجودة في هذا المسار استخدم الوسيلة DrawPath الخاصّة بكائن الرسوم Graphics Object.. حينئذٍ يمكنك تحديد القلم الذي سترسم به.

وفي المثال التالي ننشئ مسار رسوم ونضيف له قطعا ناقصا ومنحنى بيزير:

**Dim myPath As New GraphicsPath()**

**myPath.AddEllipse(New Rectangle(10, 30, 40, 50))**

**mypath.AddPie(100, 50, 100, 200, 90, 10)**

**Me.CreateGraphics.DrawPath(Pens.Aqua, myPath)**

إنّ هناك فوائدَ جمّةً لكائن مسار الرسوم، فقدرتك على رسم شكل معقّد يتكوّن من مجموعة مختلفة من الأشكال، والتعامل معه باعتباره شكلا واحدا، يمنحك القدرة على تكرار رسم هذا الشكل لأيّ عدد من المرّات باستدعاء وسيلة واحدة DrawPath، مع قدرتك على تغيير القلم (اللون والسمك) في كلّ مرّة، وقدرتك على ملء الشكل بألوان مختلفة أو صور مختلفة في كلّ مرّة.

وكذلك يستخدم مسار الرسوم في إنشاء كائن مسار متدرّج اللون PathGradient، وهو ما سيسمح لك برسم أشكال ساحرة، [كما سنرى فيما بعد](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\7-%20الألوان%20المتدرّجة\2-%20المتدرّجات%20المتشعّبة.htm).

**الألوان المتدرّجة Gradients**

لدينا نوعان من التدرّج:

* [**المتدرّجات الخطّيّة Linear Gradients:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\7-%20الألوان%20المتدرّجة\1-%20المتدرّجات%20الخطّيّة.htm)
* [**المتدرّجات المتشعّبة Path Gradients:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\7-%20الألوان%20المتدرّجة\2-%20المتدرّجات%20المتشعّبة.htm)

**المتدرّجات الخطّيّة Linear Gradients:**

يتمّ إنشاء هذا النوع من الألوان المتدرّجة بالفرشاة LinearGradientBrush.. ابدأ بتعريف متغيّر من هذه الفرشاة كالتالي:

**Dim lgBrush As New LinearGradientBrush(مستطيل, لون البداية, \_**

**لون النهاية, طراز التدرّج)**

هذه الفرشاة مؤهّلة لملء مساحة مماثلة لمساحة المستطيل المحدّد في المعامل الأوّل، بألوان تتدرّج من لون البداية حتّى تصل للون النهاية.. فإذا كان الشكل الذي ستملؤه الفرشاة أصغر من المستطيل، فإنّ جزءا فقط من التدرّج هو الذي سيملأ الشكل.. أمّا لو كان الشكل أكبر من المستطيل، فسيتمّ تكرار التدرّج حتّى يملأ كلّ الشكل.

ويحدّد طراز التدرّج اتّجاه التدرّج، ويمكن أن يأخذ قيمةً من القيم التالية:

|  |  |
| --- | --- |
| BackwardDiagonal | تدرّج قطريّ من أعلى اليمين إلى أسفل اليسار. |
| ForwardDiagonal | تدرّج قطريّ من أعلى اليسار إلى أسفل اليمين. |
| Horizontal | تدرّج أفقيّ. |
| Vertical | تدرّج رأسيّ. |

والجزء التالي من الكود موجود في مشرع GDIPlusGradients في مجلّد برامج هذا الفصل، وهو يملأ مستطيلين بألوان متدرّجة:

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

**Dim R As New RectangleF(20, 20, 300, 100)**

**Dim startColor As Color = Color.BlueViolet ' لون البداية أزرق بنفسجي**

**Dim EndColor As Color = Color.LightYellow ' لون النهاية أصفر فاتح**

**Dim LGBrush As New LinearGradientBrush (R, startColor, EndColor, \_**

**LinearGradientMode.Horizontal)**

**G.FillRectangle(LGBrush, New Rectangle(20, 20, 200, 100))**

**G.FillRectangle(LGBrush, New Rectangle(20, 150, 600, 100))**

وتوجد عدّة صيغ أخرى لتعريف هذه الفرشاة.. فمثلا، يمكنك استخدام نقطتين بدلا من المستطيل، هاتان النقطتان ستمثلان رأسي المستطيل المتقابلين (العلويّ الأيسر، والسفليّ الأيمن).. وهما في نفس الوقت تحدّدان اتجاه التدرّج، حيث سيكون في اتجاه الخطّ الواصل بين النقطتين (ممّا سيتيح لك اتجاهات غير ممكنة في صيغة المستطيل وطراز التدرّج).

**Dim lgBrush As New LinearGradientBrush(نقطة البداية, نقطة النهاية, \_**

**لون البداية, لون النهاية)**

كما أنّ هناك صيغة تتيح لك تحديد زاوية التدرّج، بدلا من طراز التدرّج:

**Dim lgBrush As New LinearGradientBrush(مستطيل, لون البداية, \_**

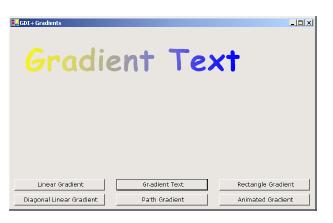
**لون النهاية, زاوية التدرّج)**

حيث تقاس هذه الزاوية بالنسبة للمحور الأفقيّ، في اتجاه عقارب الساعة.

ويمكنك إضافة معامل أخير على الصيغة السابقة.. لو جعلته True، فستتأثّر الزاوية بالتحويلات [Transform](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\8-%20تحويل%20الإحداثيّات.htm) المجراة على الإحداثيات (حيث يتمّ قياسها بالنسبة للمحاور بعد تدويرها)..أو False إذا كنت تريد أن يتمّ قياس الزاوية بطريقة مستقلّة.

ويمكن استخدام الألوان المتدرّجة لملء المنحنيات والمضلّعات الأخرى.. في هذه الحالة يتمّ ملء المستطيل الذي يحتوي المنحنى أو المضلّع، ولكن لا يتمّ عرض إلا الجزء الواقع داخل الشكل.

والطريف أنّك تستطيع رسم نصّ على النموذج بألوان متدرّجة.. هذا الكود أيضا موجود في مشروع GDIPlusGradients وهو يفعل ذلك:



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

**G.Clear (Me.BackColor) ' مسح أيّ رسوم على النموذج، مع المحافظة على لون الخلفيّة**

**G.TextRenderingHint = \_**

**System.Drawing.Text.TextRenderingHint.AntiAlias**

**Dim largeFont As New Font("Comic Sans MS", 48, \_**

**FontStyle.Bold, GraphicsUnit.Point)**

**Dim txt As String = "نصّ متدرّج"**

**' معرفة مساحة النصّ**

**Dim txtSize As SizeF = G.MeasureString(txt, largeFont)**

**' إنشاء مستطيل له مساحة النصّ**

**Dim R As New RectangleF(New PointF(0,0), txtSize)**

**' إنشاء الفرشاة**

**Dim grBrush As New LinearGradientBrush(R, Color.Yellow, \_**

**Color.Blue, LinearGradientMode.BackwardDiagonal)**

**G.DrawString(txt, largeFont, grBrush, 20, 20)**

الجدير بالذكر، أنّ المتدرّج الخطّي يمنحك هذه الإمكانيّات الإضافيّة:

1. استخدام أكثر من لونين للتدرّج:

وذلك باستخدام خاصيّة InterpolationColors.

1. تحديد كيفيّة مزج الألوان:

وذلك باستخدام خليّة المزج Blend Class أو الوسيلتين SetSigmaBellShape و SetBlendTriangularShape.

1. تحويل إحداثيات المتدرّج:

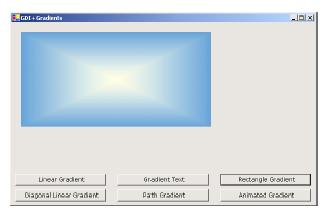
وذلك باستخدام خاصيّة [Transform](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\8-%20تحويل%20الإحداثيّات.htm).

1. إنشاء متدرّج يبدأ من لون في المركز، ويتدرّج إلى لون آخر على الطرفين.
2. إنشاء متدرّج خطّي غير منتظم التدرّج Nonuniform Linear Gradient.

ويمكنك التعرّف على كلّ هذه الإمكانيّات، بالاستعانة بملفات المساعدة المرفقة باللغة.. ألا تعتقد أنّ الوقت قد حان لتعتمد على نفسك قليلا؟

**المتدرّجات المتشعّبة Path Gradients:**

يمكنك استخدام الفرشاة PathGradientBrush لرسم ألوان متدرّجة، تبدأ من لون معيّن، وتنتشر في اتجاهات مختلفة، لتنتهي في كل اتجاه منها بلون مختلف.



ولاختبار هذه الإمكانية الرائعة، افحص المشروع GDIPlusGradients.

وهذه الفرشاة تناسب تلوين كائن مسار الرسوم GraphicsPath، لهذا فإنّك تحدد لها الكائن الذي ستستخدم في تلوينه عند تعريفها:

**Dim pgBrush As PathGradientBrush**

**pgBrush = New LinearGradientBrush(GraphicsPath )**

والآن استخدم الخاصيتين التاليتين من خصائص هذه الفرشاة لتحديد ألوان التدرّج:

**لون المركز CenterColor:**

لتحديد لون المركز، الذي يبدأ منه التدرّج.. ولو لم تحدّد هذه الخاصيّة، فستكون قيمتها الافتراضيّة هي مركز الشكل الذي سيتمّ تلوينه.

**الألوان المحيطة SurroundColors:**

مصفوفة من الألوان بعدد رءوس الأشكال الموجودة في كائن مسار الرسوم.. مثال:

**Dim Colors() As Color = {Color.Yellow, Color.Green, Color.Blue}**

**pgBrush.SurroundColors = Colors**

وبعد أن تنشئ كائن الفرشاة وتضبط خصائصه، استدع الوسيلة FillPath الخاصّة بكائن الرسوم لتلوين الأشكال الموجودة في مسار الرسوم.

والمثال التالي يريك كيف تلوّن مستطيلا بألوان متدرّجة من الأحمر في مركزه، إلى الأصفر والأخضر والأزرق والـ Cyan في أطرافه.. ويمكنك تجربة هذا المثال في مشروع GDIPlusGradients.. لاحظ أنّنا رسمنا المستطيل كأربعة خطوط مغلقة وأضفناها لمسار الرسوم:

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

**Dim path As New GraphicsPath()**

**path.AddLine(New Point(10, 10), New Point(400, 10))**

**path.AddLine(New Point(400, 10), New Point(400, 250))**

**path.AddLine(New Point(400, 250), New Point(10, 250))**

**Dim pathBrush As New PathGradientBrush(Path )**

**pathBrush.CenterColor = Color.Red**

**Dim surroundColors() As Color = \_**

**{Color.Yellow, Color.Green, Color.Blue, Color.Cyan}**

**pathBrush.SurroundColors = surroundColors**

**G.FillPath(pathBrush, Path)**

ولا يوجد ما يمنع أن تضيف مستطيلا (أو أيّ شكل آخر) لكائن مسار الرسوم ثم تلوّنه بالفرشاة متشعّبة التدرّج.

**تحويل الإحداثيّات Coordinate Transformations:**

لدينا ثلاثة أنواع من تحويلات الإحداثيّات:

1. **تغيير مقياس الرسم Scaling:**

بحيث يتغيّر حجم الشكل تكبيرا أو تصغيرا.. وتقوم بذلك الوسيلة ScaleTransform من وسائل كائن الرسوم Graphics Object:

**Graphics.ScaleTransformation(نسبة التكبير أفقيّا, نسبة التكبير رأسيّا)**

فإذا كانت النسبة أصغر من واحد كانت العمليّة تصغيرا، وإذا كانت أكبر من 1 كانت تكبيرا.

**2- تغيير الموضع Translation:**

بحيث نحرّك الشكل من موضعه.. وتقوم بذلك الوسيلة TranslateTransform من وسائل كائن الرسوم:

**Graphics.TranslateTransform(مقدار الانتقال أفقيّا, مقدار الانتقال رأسيّا)**

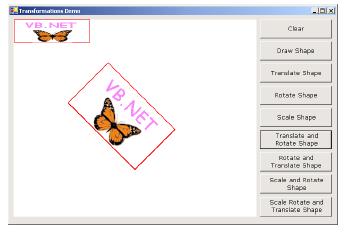
فإذا كان مقدار الانتقال موجبا انتقل الشكل لليمين أو لأسفل، وإذا كان سالبا، انتقل الشكل لليسار أو لأعلى.

1. **الدوران Rotation:**

بحيث ندير الشكل بأيّ زاوية.. وتقوم بذلك الوسيلة RotateTransform من وسائل كائن الرسوم.

**Graphics.RotateTransform(زاوية الدوران)**

وتنحصر زاوية الدوران بين 0 و 360 درجة.



إنّ هذه التحويلات تراكميّة، فمثلا: تدوير الشكل 30 درجة، ثمّ تدويره 40 درجة، يكافئ تدويره 70 درجة مباشرة.

هذا التراكم ناتج من أنّ هذه التحويلات يتمّ حفظها في مصفوفة تسمّى "مصفوفة التحويلات" Transformation matrix.

فإذا أردت أن تعيد الإحداثيّات لوضعها الأصليّ، فاستخدم الوسيلة ResetTransform.

ويمكنك اختبار هذه التحويلات في المشروع GDIPlusTransformations.

**قص منطقة الرسم Clipping:**

إذا أردت أن تقصّ منطقة من النموذج أو الأداة لتكون هي المنطقة التي يرسم عليها كائن الرسوم، فاستخدم الوسيلة SetClipالخاصّة بكائن الرسوم Graphics object.



وأوّل معامل تأخذه هذه الوسيلة، يحدّد منطقة الرسم.. في هذا الحالة يمكنك أن تجعل المعامل مستطيلا.. فإذا شئت أن تجعل للمنطقة أيّ شكل أكثر تعقيدا، فاستخدم كائن مسار الرسوم GraphicsPath، أو منطقة Region (المنطقة هي سجلّ يتكوّن من أشكال بسيطة، حيث تتكوّن المنطقة من اتحاد هذه الأشكال، أو من تقاطعها، أو من فرق أحدها من الآخر، أو من المنطقة التي تقع خارج التقاطع).

وهناك معامل آخر اختياريّ، يحدّد كيف ستتم إضافة هذه المنطقة للنموذج أو المنطقة المعرّفة سابقا في النموذج.. ويأخذ هذا المعامل قيمة من قيم المرقّم CombineMode، وهي:

|  |  |
| --- | --- |
| Complement | منطقة الرسم الجديدة هي التي تقع خارج اتحاد المنطقتين. |
| Exclude | منطقة الرسم الجديدة هي التي تقع في المنطقة الجديدة وحدها، خارج منطقة التقاطع. |
| Intersect | منطقة الرسم الجديدة هي منطقة التقاطع. |
| Replace | المنطقة الجديدة ستكون بديلة لسابقتها. |
| Union | منطقة الرسم الجديدة هي ناتج اتّحاد المنطقتين. |
| XOR | منطقة الرسم الجديدة هي ناتج اتحاد المنطقتين ما عدا منطقة تقاطعهما. |

وبعد تحديد منطقة الرسم، فإنّ أيّ شكل ترسمه لن يُرسم إلا في هذه المنطقة، وأيّ تحويل للإحداثيّات لن ينطبق إلا على هذه المنطقة.

ولإزالة المنطقة المقصوصة، والعودة للرسم على النموذج كلّه، استخدم الوسيلة ResetClip.

ويريك مشروع Clipping، كيف تعرض النصّ والصورة داخل قطع ناقص.

**رسم النصوص:**

لرسم نصّ على نموذج أو أداة، استخدم الصيغة التالية:

Graphics.DrawString(النصّ, الخطّ, الفرشاة, X, Y)

حيث X و Y يمثّلان نقطة بداية الكتابة (أعلى نقطة يسارا في المنطقة التي تحتوي النصّ).

ولمعرفة المساحة التي يشغلها النصّ على النموذج أو الأداة، استخدم الوسيلة "قياس النصّ" MeasureString، حيث تعيد لك كائن حجم بدقّة عشريّة SizeF، بحيث يمكنك استخدام خاصيتيه العرض والارتفاع لمعرفة مساحة النصّ.

والمثال التالي يوضّح لك كيف النصّ في مركز النموذج تماما:

Dim textSize As SizeF

Dim X As Integer, Y As Integer = 0

textSize = Me.Graphics.MeasureString(string, font)

X = (Me.Width – textSize.Width) / 2

G.DrawString("هذا النصّ في مركز النموذج", font, brush, X, Y)

ويمكنك تجربة ذلك في مشروع TextEffects في مجلّد برامج هذا الفصل.



وهناك صيغة أخرى ترسم النصّ داخل المستطيل الذي تحدّده لها، حتّى لو اقتضى الأمر كتابة النصّ على أكثر من سطر داخل المستطيل (إذا كان النصّ أطول من عرض المستطيل):

Graphics.DrawString(النصّ, الخطّ, الفرشاة, مستطيل)

Graphics.DrawString(النصّ, الخطّ, الفرشاة, مستطيل,تنسيق النصّ)

ولمعرفة عدد السطور التي كتب فيها النصّ، استخدم الوسيلة MeasureString.. طبعا ستتعجب: ألم نستخدم هذه الوسيلة لمعرفة أبعاد النصّ؟

نعم، وهناك صيغة أخرى منها تنتهي بمعاملين مرجعيّين، يرجعان لك عدد سطور النصّ، وعدد الأعمدة التي كتب عليها:

e.Graphics.MeasureString(النصّ, الخطّ, كائن حجم به أبعاد المستطيل الذي يحتوي النصّ, \_

تنسيق النصّ, عدد السطور, عدد الأعمدة)

ويمكنك تجربة ذلك في مشروع TextEffects في مجلّد برامج هذا الفصل.

ولكن ما هو المعامل تنسيق النصّ، ذلك الذي ظهر في أكثر من وسيلة؟

هذا المعامل من النوع StringFormat، وهو يمنحك الخصائص التالية:

**محاذاة النصّ Alignment:**

تأخذ هذه الخاصيّة واحدة من قيم المرقّم StringAlignment، وهي:

|  |  |
| --- | --- |
| Center | يتم توسيط النصّ في مركز المستطيل. |
| Far | تتمّ محاذاة النصّ بعيدا عن ركن المستطيل العلويّ الأيسر. |
| Near | تتمّ محاذاة النصّ قريبا من ركن المستطيل العلويّ الأيسر. |

**قصّ النصّ Trimming:**

تستخدم هذه الخاصيّة لتوضيح كيف سيتمّ قصّ النصّ إذا تجاوز طوله حدود المستطيل.. وتأخذ هذه الخاصيّة القيم التالية:

|  |  |
| --- | --- |
| Character | يتمّ قصّ النصّ عند أقرب حرف لحدود المستطيل. |
| EllipsisCharacter | يتمّ قصّ النصّ عند أقرب حرف لحدود المستطيل، وتوضع بعض النقاط في نهاية النصّ للدلالة على أنّ باقي النصّ غير ظاهر. |
| EllipsisPath | يتمّ حذف جزء من منتصف النصّ، وتوضع مكانه بعض النقاط. |
| Word | يتمّ قصّ النصّ عند أقرب كلمة لحدود المستطيل. |
| EllipsisWord | يتمّ قصّ النصّ عند أقرب كلمة لحدود المستطيل، وتوضع بعض النقاط في نهاية النصّ للدلالة على أنّ باقي النصّ غير ظاهر. |
| None | لا يتمّ قصّ النصّ. |

**مؤشّرات التنسيق FormatFlags:**

يمكن لهذه الخاصيّة أن تأخذ قيمةً أو أكثر من قيم المرقّم StringFormatFlags، ومن أهمّها DirectionRightToLeft لتحديد أنّ اتجاه النصّ من اليمين لليسار، DirectionVertical لكتابة النصّ رأسيّا.

والمثال التالي يريك كيف تكتب رأسيّا على النموذج:

Dim G As Graphics = Me.CreateGraphics

Dim SF As New StringFormat()

SF.FormatFlags = StringFormatFlags.DirectionVertical

G.DrawString("Visual Basic", Me.Font, Brushes.Red, 80, 80, SF)

**التعامل مع الصور**

* [**مربّع الصورة PictureBox:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20التعامل%20مع%20الصور\1-%20مربّع%20الصورة.htm)
* [**كائن الصورة Image Object:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20التعامل%20مع%20الصور\2-%20كائن%20الصورة.htm)
* [**الصور ولوحة القصاصات Clipboard:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20التعامل%20مع%20الصور\3-%20الصور%20ولوحة%20القصاصات.htm)
* [**رسم صورة DrawImage:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20التعامل%20مع%20الصور\4-%20رسم%20صورة.htm)
* [**الصور النقطيّة Bitmaps:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20التعامل%20مع%20الصور\5-%20الصور%20النقطيّة.htm)

**مربّع الصورة PictureBox:**

يمنحك VB أداةً متخصّصةً في عرض الصور والتعامل معها، تلك هي مربّع الصورة PictureBox.. ولكي تعرض صورةً فيه، استخدم خاصيّة "صورة" Image في نافذة الخصائص، حيث ستجد زر انتقال Ellipsis Button في خانة القيمة.. اضغط هذا الزر ليظهر لك مربّع حوار فتح ملفّ.. اختر الصورة التي تريد عرضها واضغط موافق.. هذه الصورة سيتمّ حفظها في الملفّ الذي يحمل نفس اسم النموذج ولكن امتداده ".resx".. لهذا لن تحتاج لتوزيع الصورة الأصليّة مع برنامجك، فهي محتواةٌ فيه بالفعل.

ومن خصائص مربّع الصورة الهامّة ما يلي:

**طراز الحجم SizeMode:**

تمكّنك هذه الخاصيّة من اختيار كيفيّة ظهور الصورة ومحاذاتها في مربّع الصورة.. ولهذه الخاصيّة القيم التالية:



|  |  |
| --- | --- |
| Normal | تظهر الصورة طبيعية كما هي بدون تعديل.. فإذا كانت أكبر من مربّع الصورة، فسيختفي جزء منها، وإذا كانت أصغر، فلن تملأ كل مساحة مربّع الصورة. |
| CenterImage | سيتمّ عرض الصورة بحيث تتوسّط مربّع الصورة. |
| StretchImage | سيتمّ مطّ الصورة أو تقليصها بحيث تشغل مساحة مربّع الصورة بالضبط.. ورغم أنّ هذه الطريقة هي أفضل طريقة لعرض الصورة، إلا إنّ عليك أن تراعي تناسب أبعاد مربّع الصورة مع أبعاد الصورة، حتّى لا تتشوّه الصورة عند مطّها.. افحص مشروع ImageLoad في مجلّد برامج هذا الفصل، لترى كيف نحافظ على تناسب مقاييس الصورة. |
| AutoSize | سيتمّ تغيير أبعاد مربّع الصورة ليطابق أبعاد الصورة.. إنّ هذا الاختيار غير مفضّل، لأنّ وضع صورة كبيرة في مربّع الصورة سيجعله يغطّى باقي الأدوات الموجودة على النموذج. |

**طراز الحافّة BorderStyle:**

وهي تأخذ قيم المرقّم BorderStyle وهي: ثابتة مجسّمة **Fixed3D** ـ ثابتة مفردة **FixedSingle** ـ بدون حافّة **None**.

**الصورة Image:**

تسمح لك بتحديد الصورة التي تعرضها في مربّع الصورة.

**صورة الخلفيّة BackgroundImage:**

تسمح لك بتحديد الصورة التي تعرضها في خلفيّة مربّع الصورة.

**كائن الصورة Image Object**

* [**تعريف كائن الصورة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\2-%20كائن%20الصورة\1-%20تعريفه.htm)
* [**الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\2-%20كائن%20الصورة\2-%20الخصائص.htm)
* [**الوسائل:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\2-%20كائن%20الصورة\3-%20الوسائل.htm)

**تعريفه:**

مرارا نكرّر: إنّ كلّ شيء في VB.Net ما هو إلا كائن، لهذا فمن الطبيعيّ أن يكون هناك كائن يمكنك من خلاله التعامل مع الصورة.

فمثلا، خاصيّة Image الخاصّة بمربّع الصورة تمثّل كائن صورة Image object.. هذا الكائن يمثّل الصورة الموجودة في مربّع الصورة، ويمنحك الخصائص والوسائل اللازمة للتعامل معه.

ويمكن أن تعرّف كائن صورة، وتضع فيه الصورة التي تحملها خاصيّة Image كالتالي:

**Dim img As Image = PictureBox1.Image**

كما يمكنك أن تحمّل صورة من ملفّ وتضعها في كائن صورة عن طريق الوسيلة "من ملفّ" FromFile ثمّ تضعها في مربّع الصورة كالتالي:

**Dim img As Image = Image.FromFile("Butterfly.jpg")**

**PictureBox1.Image = img**

**الوسائل:**

**تدوير وعكس RotateFlip:**

تمكّنك هذه الخاصيّة من تدوير الصورة أو عكسها أو كلا الأمرين معا.

**Image.RotateFlip(النوع)**

حيث النوع هو إحدى قيم المرقّم RotateFlipType التالية:

|  |  |
| --- | --- |
| RotateNoneFlipNone | لا تدوير ولا عكس.. واضح طبعا أنّك بهذا لن تفعل شيئا في الصورة (يفيد هذا في إلغاء عمليات التدوير والعكس السابقة، لإعادة الصورة إلى وضعها الأصليّ). |
| Rotate90FlipNone | تدوير الصورة بزاوية 90 بدون عكسها. |
| Rotate90FlipX | تدوير الصورة بزاوية 90 وعكسها أفقيّا. |
| Rotate90FlipXY | تدوير الصورة بزاوية 90 وعكسها أفقيّا ورأسيّا. |
| Rotate90FlipY | تدوير الصورة بزاوية 90 وعكسها رأسيّا. |

وأعتقد أنّك تستطيع فهم باقي هذه التعبيرات:

**Rotate180FlipNone** ـ **Rotate180FlipX** ـ **Rotate180FlipXY** ـ **Rotate180FlipY** ـ **Rotate270FlipNone** ـ **Rotate270FlipX** ـ **Rotate270FlipXY** ـ **Rotate270FlipY** ـ **RotateNoneFlipX** ـ **RotateNoneFlipXY** ـ **RotateNoneFlipY**.

مثال:

**PictureBox1.Image.RotateFlip(RotateFlipType.RotateNoneFlipY)**

**PictureBox1.Refresh()**

لاحظ استخدام خاصيّة الإنعاش لإعادة رسم الصورة، وإلا فلن يرى المستخدم أيّ تغيير.

ويمكنك التدريب على هذه الوسيلة في مشروع LoadImage.

**كوّن صورة مصغّرة GetThumbnailImage:**

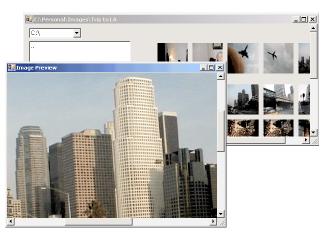
تمكّنك هذه الوسيلة من الحصول على نسخة مصغّرة من الصورة، حيث يمكنك استخدامها في عرض "كتالوج" الصور للمستخدم، لتسمح له باختيار صورة منها لعرضها كاملة.. ولهذه الوسيلة الصيغة التالية:

**Image.GetThumbnailImage(العرض, الارتفاع, إلغاء, بيانات)**

حيث العرض والارتفاع يحدّدان بُعدَيِ الصورة المصغّرة الناتجة، بينما إلغاء وبيانات يستخدمان في حالة إلغاء العمليّة، ونظرا لأنّ تكوين الصورة المصغّرة لا يستغرق وقتا، فسنجعل هذين المعاملين دائما Nothimng.. مثال:

**PictureBox1.Image = img.GetThumbnailImage(32, 32, Nothing, Nothing)**

ولديك في مجلّد برامج هذا الفصل، المشروع Thumbnails، وفيه تسمح للمستخدم باختيار مجلّد (باستخدام الوسائل التي درسناها في [الفصل العاشر](file:///C:\Users\Administrator\02-%20Classes%20الخلايا\10-%20الملفّات\4-%20مربعا%20حوار%20فتح%20ملفّ%20وحفظ%20ملفّ.htm))، حيث يقوم البرنامج بفحص ملفّات هذا المجلّد وعرض صور مصغّرة على النموذج لكلّ الصور الموجودة بهذا المجلّد.. ويتمّ عرض كلّ صورة مصغّرة في مربّع صورة خاصّ بها، حيث يتمّ إنشاؤه في وقت التنفيذ وضبط إحداثياته، باستخدام الوسائل التي درسناها في [الفصل الخامس](file:///C:\Users\Administrator\01-%20أساسيّات%20اللغة\05-%20Forms%20النماذج\4-%20بناء%20نماذج%20مرنة.htm).. وعندما يضغط المستخدم أيّ صورة مصغّرة، يتمّ عرض صورتها الأصليّة كاملة في نموذج آخر.



هذا هو الكود الذي يعرض الصور المصغّرة:

**Dim file As String**

**Dim FI As FileInfo**

**Dim PBox As PictureBox, img As Image**

**Dim Left As Integer = 280**

**Dim Top As Integer = 40**

**Dim ctrl As Integer**

**' احذف كلّ مربّعات الصور الموجودة على النموذج**

**For ctrl = Me.Controls.Count - 1 To 2 Step -1**

**Me.Controls.Remove(Me.Controls(ctrl))**

**Next**

**' استخدم الوسيلة التالية لمحو أيّ رسومات غير دائمة من على النموذج**

**Me.Invalidate()**

**For Each file In Directory.GetFiles(Directory.GetCurrentDirectory)**

**FI = New FileInfo(file)**

**If FI.Extension = ".GIF" Or FI.Extension = ".JPG" Or \_**

**FI.Extension = ".BMP" Then**

**' إنشاء مربّع صورة جديد**

**PBox = New PictureBox()**

**' تحميل الصورة**

**img = Image.FromFile(FI.FullName)**

**' وضع الصورة المصغّرة في مربّع الصورة**

**PBox.Image = img.GetThumbnailImage(64, 64, \_**

**Nothing, Nothing)**

**' إذا تجاوز موضع اليسار لمربّع الصورة الجديد عرض النموذج،**

**' فضعه في بداية صفّ تالي (بزيادة موضع حافته العليا)**

**If Left > 580 Then**

**Left = 280**

**Top = Top + 74**

**End If**

**' ضبط إحداثيات مربّع الصورة**

**PBox.Left = Left**

**PBox.Top = Top**

**PBox.Width = 64**

**PBox.Height = 64**

**PBox.Visible = True**

**' لا بدّ من الاحتفاظ باسم ملفّ الصورة،**

**' حتّى يمكن عرضها عند الضغط على الصورة المصغّرة**

**PBox.Tag = FI.FullName**

**' عرض مربّع الصورة على النموذج**

**Me.Controls.Add(PBox)**

**' إنشاء مستجيب لحدث ضغط مربّع الصورة**

**AddHandler PBox.Click, New \_**

**System.EventHandler(AddressOf OpenImage)**

**' إضافة عرض مربّع النصّ + 10 (مسافة فاصلة)**

**' إلى المتغيّر الدال على موضع مربّع الصورة الجديد**

**Left += 74**

**End If**

**Next**

وهذا هو الكود الذي يعرض الصورة عند الضغط على صورتها المصغّرة:

**Dim imgForm As New previewForm()**

**imgForm.PictureBox1.Image = Image.FromFile(sender.tag)**

**imgForm.Show()**

**حفظ Save:**

استخدم هذه الوسيلة لحفظ الصورة الموجودة في كائن الصورة، كملفّ على الجهاز.

مثال:

**PictureBox1.Image.Save("c:\tmpImage.bmp")**

ويمكنك اختيار النوع الذي تريد حفظ الصورة به، كالتالي:

**PictureBox1.Image.Save("c:\tmpImage.bmp", نوع الصورة)**

حيث نوع الصورة هو إحدى قيم المرقّم System.Drawing.Imaging.ImageFormat.. واضح أنّ الاسم طويل، وطبعا تعرف أنّك تستطيع كتابة جملة الاستيراد التالية في مشروعك:

**Imports System.Drawing.Imaging**

وبذلك تستطيع استخدام اسم المرقّم مختصرا.

ولهذا المرقّم القيم التالية:

|  |  |
| --- | --- |
| Bmp | Bitmap، وهو أبسط أنواع الصور، حيث يتمّ حفظ الصورة في هيئة مصفوفة ثنائيّة البعد (جدول من صفوف وأعمدة).. وبهذا فإنّ كل خانة تحمل الإحداثيّات المناظرة لنقطة في الصورة (رقمها في الصف يناظر الإحداثيّ الأفقيّ، ورقمها في العمود يمثّل الإحداثيّ الرأسّي)، بينما قيمة الخانة تمثّل لون النقطة. |
| Emf | Enhanced Windows metafile. |
| Exif | Exchangeable Image Format. |
| Gif | Graphics Interchange Format. |
| Icon | أيقونة. |
| Jpeg | Joint Photographic Experts Group. |
| MemoryBmp | لحفظ الصورة كمصفوفة ذاكرة. |
| Png | W3C Portable Network Graphics. |
| Tiff | Tagged Image File Format. |
| Wmf | Windows metafile. |

واضح طبعا مدى ثراء أنواع الصور التي يتعامل معها VB.Net، أكّثر ممّا في VB6.

**خصائص كائن الصورة:**

الخصائص التالية للقراءة فقط:

**الدقّة الأفقيّة HorizontalResolution والدقّة الرأسيّة VerticalResolution:**

يمكنك بهاتين الخاصيّتين أن تعرِف دقّة تمثيل الصور (عدد النقط في كل بوصة Pixels -per-inch).

**العرض Width والارتفاع Height:**

أعتقد أنّ هاتين الخاصيتيّن واضحتان.. ولكنّ ما سألفت نظرك إليه هنا، هو التالي:

HorizontalResolution ÷ Width = عرض الصورة بالبوصة.

VerticalResolution ÷ Height = ارتفاع الصورة بالبوصة.

كما أحبّ أن ألفت نظرك أيضا إلى أنّ تغيير عرض وارتفاع مربّع الصورة، لا يؤثّر على عرض وارتفاع كائن الصورة، فهاتان الخاصيتان تشيران دائما إلى مقاييس الصورة الأصليّة، وليس النسخة المعروضة في مربّع الصورة.

**تنسيق النقط PixelFormat:**

هناك العديد من تنسيقات النقط، يجمعها المرقّم PixelFormat.. فهناك مثلا القيمة Format24bppRgb، وهي تدلّ على أنّ كل نقطة في الصورة تمثّل اللون بـ 24 خانة ثنائيّة 24 bits per pixel.

**الصور ولوحة القصاصات Clipboard:**

رأينا من قبل كيف ننسخ النصوص إلى لوحة القصاصات، حتّى نستطيع أن نلصقها في أيّ تطبيق ويندوز آخر أو في أيّ موضع من هذا التطبيق.

كلّ ما فعلناه حينئذ هو استخدام الوسيلتين "اقرأ كائن البيانات" GetDataObject و"احفظ كائن البيانات" SetDataObject.

لا جديد هنا.. سنستخدم نفس الوسيلتين، ولكن مع الصور هذه المرة.

استخدم الجملة التالية لنسخ الصورة المعروضة في مربّع الصور إلى لوحة القصاصات:

**Clipboard.SetDataObject(PictureBox1.Image)**

ولو أحببت أن تبقى الصورة في لوحة القصاصات بعد إغلاق برنامجك فاستخدم الصيغة:

**Clipboard.SetDataObject(PictureBox1.Image, True)**

ولاستعادة الصورة من لوحة القصاصات، يمكنك استخدام الوسيلة GetDataObject.. هذه الوسيلة تُرجع كائنا من النوع "كائن واجهة البيانات" IDataObject، وهو يمنحك الوسيلتين:

**اقرإ البيانات GetData:**

تعيد لك محتويات لوحة القصاصات.. ونظرا لأنّ لوحة القصاصات قد تحتوي على بيانات من أنواع مختلفة معا في نفس الوقت، فإنّك تمدّ هذه الوسيلة بمعاملٍ يمثّل نوع البيانات الذي تريده.

**هل البيانات موجودة GetDataPresent:**

ترجع True إذا كان نوع البيانات الذي أرسلته لها كمعامل هو نوع البيانات الموجودة في لوحة القصاصات.. ويمكنك أن ترسل لهذه الوسيلة معاملا ثانيا، إذا جعلت قيمته True، فإنّ لوحة القصاصات تحاول تحويل البيانات الموجودة بها إلى النوع الذي أرسلته في المعامل الأوّل.

ويمكنك استخدام الجملة التالية لعرض الصورة من لوحة القصاصات في مربّع الصورة:

**PictureBox1.Image = \_**

**Clipboard.GetDataObject.GetData(DataFormats.Bitmap)**

ولمزيد من التدريب، لديك مشروع ImageClipboard في مجلّد برامج هذا الفصل.

**رسم صورة DrawImage:**

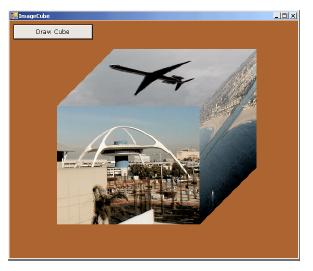
يمنحك كائن الرسوم Graphics الوسيلة DrawImage لتمكّنك من رسم الصور على أيّ سطح تريده.. ولهذه الوسيلة 30 صيغة مختلفة، قد تحتاج لكتاب بمفردها!!.. لهذا سنكتفي بشرح أربع صيغ:

**Graphics.DrawImage(الصورة, نقطة رأسها العلويّ الأيسر)**

**Graphics.DrawImage(الصورة, المستطيل الذي يحتويها)**

**Graphics.DrawImage(الصورة, points())**

وفي الصيغة الثالثة، المعامل الثاني هو مصفوفة نقط، حيث تضع فيها ثلاثة نقاط، تمثّل ثلاثة من رءوس متوازي الأضلاع، الذي تريد أن تحوّر الصورة لترسم بداخله.. هذه الرءوس الثلاثة هي العلويّ الأيسر والعلوي الأيمن، والسفليّ الأيسر.. أمّا الرأس الرابع فيمكن استنتاجه، ولا حاجة بك لإرساله.. ولديك في مجلّد برامج هذا الفصل، المشروع المثير ImageCube، وفيه تعرف كيف يمكنك رسم مكعّب ثلاثيّ الأبعاد، ووضع صورة على كلّ وجه من أوجهه الثلاثة المواجهة للمستخدم.. إنّ فعل هذا بسيط جدّا، فأنت تعرف أنّك ترسم المكعّب بثلاثة متوازيات أضلاع، أحدها في المواجهة، والثاني يعطي الإحساس بالوجه الجانبيّ، والثالث يعطي الإحساس بقمّة المكعّب.. اختبر هذا المشروع الشيّق.

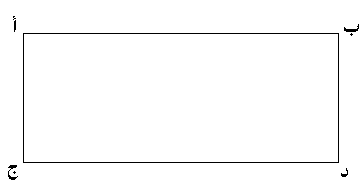


وهناك ملاحظة طريفة أخرى بخصوص هذه الصيغة.. إنّ بإمكانك أن (تلخبط) ترتيب مصفوفة النقط، وبذلك تقلب الصورة أو تعكسها، نتيجة تغيير مواضع رءوسها!.. طبعا لا يشترط دائما أن تعرض الصورة في متوازي أضلاع.. تذكّر فقط أنّ المستطيل هو متوازي أضلاع قائم الزاوية!

ويمكنك أن تجعل رأسين متطابقين لترسم الصورة في مثلث!

أبدع كما يحلو لك، فعلى حسب ترتيب رءوس المضلّع، سيتمّ قلب الصورة أو تدويرها أو عكسها أو مطّها أو تقليصها.. المهمّ أن تعرف ما تفعله!!

افترض أنّ مستطيل الصورة هو أ ب ج د، حيث أ هو الرأس العلويّ الأيسر، و ب هو الرأس العلويّ الأيمن، و ج هو الرأس السفليّ الأيسر، و د هو الرأس السفليّ الأيمن.



الجدول التالي يوضّح لك بعض طرق التلاعب بهذه الصورة، على حسب ترتيب إضافتك للنقاط للمصفوفة:

|  |  |
| --- | --- |
| ترتيب مصفوفة النقط | التأثير |
| أ ب ج | الصورة كما هي. |
| ج د أ | عكس الصورة بالنسبة للمحور الأفقيّ. |
| ب أ د | عكس الصورة بالنسبة للمحور الرأسيّ. |
| د ب ج | عكس الصورة بالنسبة للقطر ب ج. |
| أ ج د | عكس الصورة بالنسبة للقطر أ د. |
| ج أ د | تدوير الصورة 90 درجة في اتجاه عقارب الساعة. |
| ب د أ | تدوير الصورة 90 درجة عكس اتجاه عقارب الساعة. |
| أ ج ج | رسم الصورة في مثلث! |

كما أنّك تستطيع استخدام أيّ نقط أخرى تقع داخل الصورة أو خارجها، لرسم شرائح ممطوطة أو مقلّصة من الصورة!

وهكذا.. يمكنك أن تفعل بالصورة ما تشاء.

أمّا الصيغة الرابعة فهي تسمح لك بتحديد سمات الصورة:

**Graphics.DrawImage(الصورة, points(), مستطيل, وحدة القياس, السمات)**

ويمثّل المستطيل الجزء الذي تريد رسمه من الصورة.. وتمثّل وحدة القياس الوحدة التي ستقاس بها أبعاد المستطيل.

أمّا المعامل الأخير فهو كائن من النوع ImageAttributes، حيث يمنحك بعض الوسائل لتغيير سمات الصورة، مثل جعلها تظهر على النموذج بالتدريج، ومثل تصحيح ألوانها.

ابدأ بتعريف كائن من هذا النوع، مع ملاحظة أنّه ينتمي لفضاء الاسم System.Drawing.Imaging، فاستورده لو أردت.

**Dim attr As New System.Drawing.Imaging.ImageAttributes**

**ومن وسائل هذا الكائن:**

**تغيير إضاءة الصورة SetGamma:**

إذا تركت هذه الخاصيّة بالقيمة 1، فلن يتغيّر شيءٌ في لون الصورة، ولكن لو جعلتها أكبر من 1 فستصير ألوان الصورة فاتحةً أكثر، وإذا جعلتها أصغر من 1فستصير ألوان الصورة أغمق.

والكود التالي يريك كيف ترسم صورة في مربّع الصورة، مع تفتيح لونها بنسبة 25%:

**Dim attrs As New System.Drawing.Imaging.ImageAttributes()**

**attrs.SetGamma(1.25)**

**Dim dest As New Rectangle(0, 0, \_**

**PictureBox1.Width, PictureBox1.Height)**

**G.DrawImage(img, dest, 0, 0, img.Width, img.Height, \_**

**GraphicsUnit.Pixel, attrs)**

هذا بافتراض أنّ G هو كائن رسوم يرسم في مربّع الصورة، و img هو كائن صورة تمّ تحميل الصورة به.

**قنوات فصل اللون SetOutputChannel:**

تقنية فصل الألوان هي تقنية معروفة لزيادة جودة طباعة الصور الملوّنة، حيث تتمّ طباعة الصورة على نفس الورقة أربع مرّات.. في كلّ مرّة يتمّ طباعة لون معيّن تمّ فصله من الصورة بمفرده.. وفي الغالب تُفصل الصورة إلى الألوان: cyan و البنفسجيّ المحمرّ magenta والأصفر yellow والأسود black.

وللحصول على هذه النسخ من الصورة، استدع هذه الوسيلة أربع مرّات بالمعاملات التالية:

**attrs.SetOutputChannel(ColorChannelFlag.ColorChannelC) ' cyan**

**attrs.SetOutputChannel (ColorChannelFlag.ColorChannelM) ' magenta**

**attrs.SetOutputChannel(ColorChannelFlag.ColorChannelY) ' yellow**

**attrs.SetOutputChannel(ColorChannelFlag.ColorChannelK) ' black**

وهناك صيغة أخرى كالتالي:

**attrs.SetOutputChannel(ColorChannelFlag.ColorChannelLast)**

وذلك لاستدعاء الوسيلة مرّة أخرى بآخر لون تمّ استدعاؤها به.

وبخصوص باقي صيغ وسيلة رسم الصورة DrawImage، فأرجو أن تتعرّف عليها، إمّا عن طريق ملفات المساعدة، أو عن طريق تلميح الشاشة الذي يظهر عند كتابة اسم الوسيلة في محرّر الكود.. مع ملاحظة أنّ معظم هذه الوسائل تتلقّي معاملا، عبارة عن مستطيل يمثّل المساحة التي سترسم فيها الصورة.. فإذا كانت أبعاد الصورة مختلفة عن أبعاد المستطيل، يتمّ مطّها أو تقليصها لتتناسب معه.

كما أنّ بعض الصيغ تتلقّي مستطيلين: الأوّل يحدّد الجزء الذي تريد رسمه من الصورة، والآخر يوضّح المساحة التي سترسم هذا الجزء فيها.

**الصور النقطيّة Bitmaps**

* [**تركيب الصورة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20الصور%20النقطيّة\1-%20تركيب%20الصورة.htm)
* [**إبقاء الرسوم على النموذج والأدوات:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20الصور%20النقطيّة\2-%20إبقاء%20الرسوم%20على%20النموذج%20والأدوات.htm)
* [**تغيير نقط الصورة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20الصور%20النقطيّة\22-%20تغيير%20نقط%20الصورة.htm)
* [**التعامل مع نقط الصورة في الذاكرة مباشرة:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20الصور%20النقطيّة\3-%20التعامل%20مع%20نقط%20الصورة%20في%20الذاكرة%20مباشرة.htm)
* [**تغير لون معيّن في الصورة إلى لون آخر:**](file:///C:\Users\Administrator\Desktop\vb.net\d-%20%20مواضيع%20متقدّمة\12-%20الرسم%20والتلوين\5-%20الصور%20النقطيّة\4-%20تغير%20لون%20معيّن%20في%20الصورة%20إلى%20لون%20آخر.htm)

**تركيب الصورة:**

الصورة هي مصفوفة من الألوان مرتّبة في صفوف وأعمدة.. ونتيجة للألوان وترتيبها، تبدو للعين كأنّها صورة متصلة مفعمة بالتفاصيل.

وهنا يجب توضيح الفارق بين كائن الصورة Image، وكائن الصورة النقطيّة Bitmap.. فكائن الصورة Image ثابت Static، بمعنى أنّك لا تستطيع تغيير نقاطه كلّ على حدة.. أمّا كائن الصورة النقطيّة Bitmap، فهو يسمح لك بالتعامل مع الصورة نقطة نقطة.. وهذه هي الطريقة التي يمكنك بها تعديل جزء من الصورة.. هذه باختصار هي عمليّة التعامل مع الصور Image Processing.. وإن كان من واجبي هنا أن أحذّرك، أنّ التعامل مع الصورة نقطة نقطة مضيعة للوقت، خاصّة أنّ العمليات التي تريد إجراءها على الصورة غالبا ما تكون معقّدة، مثل التعرّف على الأشكال والملامح أو البصمات.. فمثلا في هذه الحالات يمكن اللجوء إلى وسائل رياضيّة معقّدة، يتمّ بها تحويل الصورة إلي متّجه Vector يتكوّن من 5 أو 6 أرقام تصف الصورة وصفا كافيا، حيث يتمّ أخذ كميّة ضخمة من العينات من الصور، يتمّ التعامل مع المتجهات الناتجة عنها لاستخلاص الاستنتاجات وتحديد كيفية التعرّف على الشكل المطلوب.

هذه طريقة.. الطريقة الأخرى هي استخدام الخلايا العصبيّة Neural Networks للتعرّف على الأشكال.

عامّة هذا فرع شيّق من البرمجة وإن كان معقّدا، وله عشرات المراجع المستقلّة، وما زالت به مناطق شاسعة قيد البحث والتطوير، فشمّر ساعدك لو أردت.

لهذا فلن نخوض في هذا الأمر ها هنا.. كل ما نريد إيضاحه هو كيف يمكنك التعامل مع نقاط الصورة.. ولن يفيدنا في شيء التوسع في الأمثلة حول معالجة الصور.

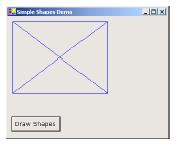
ولكن تعال أولا نرى استخداما هاما جدّا لكائن الصورة النقطيّة، وذلك باستخدامه في تثبيت الرسوم.

**إبقاء الرسوم على النموذج والأدوات:**

إنّ كلّ ما ترسمه على النموذج والأدوات يختفي لو صغّرت النموذج ثمّ كبّرته، أو جعلت نموذجا آخر يخفي جزءا من نموذجك ثم أبعدته.. الصور فقط هي التي لا تُمحى إلا عندما تزيلها أنت.. بطريقة أخرى: إنّ الرسوم ـ على خلاف الصور ـ مؤقّتة، ولا تُضاف لكائن الرسوم.

ولحلّ هذه المشكلة، يمكنك أن تكتب كود الرسم في الحدث Paint، أو الدالة التي تستدعيه OnPaint، بحيث يتمّ تنفيذه كلّما احتاج النموذج لإعادة رسمه.

إنّ للنموذج إجراءً خاصّا Private يدعى OnPaint، يمكنك أن تستبدله Override بإجراء آخر بنفس الاسم، لكي يُنفّذ كلّما احتاج النموذج لإعادة رسمه.. لقد كتبنا فيه الكود الذي يرسم الشكل التالي على النموذج:



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

**Dim G As Graphics**

**G = e.Graphics**

**Dim P As New Pen(Color.Blue)**

**G.DrawRectangle(P, 10, 10, 200, 150)**

**G.DrawLine(P, 10, 10, 210, 160)**

**G.DrawLine(P, 210, 10, 10, 160)**

**End Sub**

ولكنّ المشكلة لم تحلّ بعد!!.. فلو كانت الرسوم معقّدة وتستهلك وقتا، فسيبدو أداء برنامجك عقيما كلّما انتقلت من النموذج ثمّ عدت إليه.. ثمّ من قال إنّك ترسم دائما رسوما ثابتة؟.. إنّها قد تتغيّر مع ضغط المستخدم للقوائم والأزرار لاختيار وظائف معيّنة.. في هذه الحالة ستحتاج لكتابة كود معقّد في حدث رسم النموذج حتّى تستعيد الرسم كما كان.

إذن ما الحل؟

الحلّ هو أن نضيف الرسومات إلى كائن صورة نقطية، بحيث تصبح كالصور لا نحتاج لإعادة رسمها.

يا لها من فكرة!.. ولكن كيف ننفّذها؟

في البداية سنعرّف كائن صورة نقطيّة Bitmap Object، وهو مماثل لكائن الصورة Image Object، لدرجة أنّك تستطيع تحميل الصورة من أحدهما للآخر.

والآن سنعرّف كائن صورة نقطيّة له نفس مساحة مربّع الصورة:

**Dim bmp As Bitmap**

**bmp = New Bitmap(PictureBox1.Width, PictureBox1.Height)**

هذا الكائن يمثّل صورة فارغة الآن.

الخطوة الثانية هي أن نربط كائن الصورة الخاصّ بمربّع الصورة، بكائن الصورة النقطيّة.

**PictureBox1.Image = bmp**

وأنت تعرف طبعا أنّ هذا التساوي مرجعيّ ByRef.. الآن صار أيّ تغيير يحدث للكائن bmp، يحدث كذلك لكائن الصورة.

الخطوة الثالثة هي أن ننشئ كائن رسوم يرسم على الصورة النقطيّة، وذلك كالتالي:

**Dim G As Graphics**

**G = Graphics.FromImage(bmp)**

والآن، كلّ ما نرسمه بكائن الرسوم يتمّ رسمه على كائن الصورة النقطيّة bmp، وبالتالي يتأثّر به كائن الصورة Image الخاصّ بمربّع الصورة، وبالتالي يصبح رسما دائما في مربّع الصورة.

هذا هو الكود وقد جمعناه معا في دالّة واحدة، ليسهل عليك استخدامها بعد ذلك:

**Function GetGraphicsObject(ByVal PBox As PictureBox) As Graphics**

**Dim bmp As Bitmap**

**bmp = New Bitmap(PBox.Width, PBox.Height)**

**PBox.Image = bmp**

**Dim G As Graphics = Graphics.FromImage(bmp)**

**Return G**

**End Function**

والآن كلّ ما عليك هو إرسال اسم مربّع الصورة إلى هذه الدالة، لتعيد إليك كائن رسوم يرسم رسوما ثابتة على سطحه.

لاحظ أنّك لو أردت أن تفعل المثل مع النموذج، فستجد أنّه لا يمتلك خاصيّة Image.. لا عليك: استخدم خاصيّة "صورة الخلفيّة" BackgroundImage بدلا منها.. هذه هي الدالة التي تتعامل مع النموذج:

**Function GetGraphicsObject() As Graphics**

**Dim bmp As Bitmap**

**bmp = New Bitmap(Me.Width, Me.Height)**

**Dim G As Graphics**

**Me.BackgroundImage = bmp**

**G = Graphics.FromImage(bmp)**

**Return G**

**End Function**

انظر كيف نستخدم هاتين الدالتين في مشروع SimpleGraphics في مجلّد برامج هذا الفصل.

**ملحوظة:**

يمتلك النموذج والأدوات الوسيلة Invalidate، لمسح الأشكال غير الثابتة من عليها.. ولو ناديت هذه الوسيلة بدون معاملات فستؤثّر على كلّ مساحة النموذج أو الأداة.. ولو ناديتها وأرسلت لها مستطيلا كمعامل، فستؤثّر على مساحة النموذج أو الأداة، التي تقع داخل هذا المستطيل.. طبعا هذه الوسيلة لن تؤثّر على الرسوم التي أضفناها لصرة نقطيّة.

**ملاحظات على تغيير شكل النموذج:**

رأينا في الفصل الخامس "النماذج" كيف نغيّر شكل النموذج.. ولقد استخدمنا الكود التالي لجعل النموذج بيضاويّا:

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

**PaintEventArgs) Handles MyBase.Paint**

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

**Me.TransparentKey = Color.Red**

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

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

**End Sub**

جميل.. ما رأيك أن نستخدم طريقة الرسم على صورة نقطيّة، بدلا من استخدام الحدث Paint؟

ضع الكود التالي في حدث تحميل النموذج:

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

**System.EventArgs) Handles MyBase.Load**

**Dim Bt As New Bitmap(Me.Width, Me.Height)**

**Dim G As Drawing.Graphics = Drawing.Graphics.FromImage(Bt)**

**Me.TransparentKey = Color.Red**

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

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

**Me.BackgroundImage = Bt**

**End Sub**

جميل.. لكن لو جرّبت هذا الكود فلن يعمل!!

نتيجة مدهشة وغير متوقّعة.. صحّ؟

يا ترى ما هو السبب؟

السبب هو أنّ مفتاح الشفافية يتعامل فقط مع سطح النموذج، لهذا فهو يتجاهل الصورة في هذه الحالة تماما!

إذن فقد فشلت هذه الوسيلة؟

لا تتعجّل النتائج.. حينما حدثت معي هذه المشكلة لم أستسلم بسهولة.. جربت الكثير من الحلول، حتّى هداني الله سبحانه إلى الحلّ التالي:

1. لوّن خلفيّة النموذج بلون مفتاح الشفافية (الأحمر في مثالنا هذا) في وقت التصميم.. وانتبه لتغيّر لون الزرّ بدوره.. أعده للونه الأصليّ.
2. في بداية حدث التحميل، أجبر النموذج على الظهور:

**Me.TransparentKey = Color.Red**

**Me.Show**

بهذا سيتمّ حذف المساحة الشفافة (كلّ سطح النموذج في هذه الحالة).

1. بعد هذا يمكن وضع الصورة البيضاويّة على النموذج، بحيث تشكل مساحة سطحه.. هذا مع إلغاء جملة:

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

لأنّها تعمل على تغطية المساحة الشفافة بصورة حمراء، ممّا يلغي الشفافية!!

هذا هو الكود:

**Private Sub Form1\_Load(ByVal sender As System.Object, ByVal e As \_**

**System.EventArgs) Handles MyBase.Load**

**Me.TransparentKey = Color.Red**

**Me.Show()**

**Dim Bt As New Bitmap(Me.Width, Me.Height)**

**Dim G As Drawing.Graphics = Drawing.Graphics.FromImage(Bt)**

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

**Me.BackgroundImage = Bt**

**End Sub**

ولو جرّبته لوجدته يعمل بإذن الله.. مبروك.

**تغيير نقط الصورة:**

لكي تتمكّن من التعامل مع كلّ نقطة في الصورة على حدة، يمنحك كائن الصورة النقطيّة الوسيلتين التاليتين:

**اقرأ النقطة GetPixel:**

تأخذ هذه الوسيلة إحداثيي النقطة المطلوبة، وتعيد لك لونها:

**Dim Clr As Color = Bitmap.GetPixel(X, Y)**

وتستهلك هذه الوسيلة وقتا ملموسا، خاصّة مع تكرار استدعائها لكلّ نقاط الصورة.

**تغيير النقطة SetPixel:**

تأخذ هذه الوسيلة إحداثيي النقطة المطلوبة واللون الذي تريد تلوينها به:

**Bitmap.SetPixel(X, Y, Clr)**

وفي المثال التالي نرى كيف يمكننا عكس ألوان صورة ما للحصول على النيجاتيف منها.

لقد كتبنا في هذا الفصل دالة عكس الألوان [RevColor](file:///C:\Users\Administrator\Desktop\vb.net\2-%20التعامل%20مع%20الألوان\1-%20كائن%20اللون.htm).. والآن كلّ ما سنفعله، هو أن نطبّقها على كلّ نقطة في الصورة لعكس لونها.

ولقراءة نقاط الصورة وتغييرها، لا بد من كتابة جملتين تكراريّتين متداخلتين، إحداهما لقراءة الصفوف والأخرى لقراءة الأعمدة.

هذا هو الكود الذي يعكس الألوان:

**Dim R, C As Integer**

**' تعريف متغير صورة نقطيّة، وتحميل صورة مربّع الصورة فيه**

**Dim Btmap As New Bitmap(PictureBox1.Image)**

**' ربط مربّع الصورة بكائن الصورة النقطيّة، حتّى يتأثّر بالتغيرات التي تحدث له**

**PictureBox1.Image = Btmap**

**With Btmap**

**' جملة تكراريّة للمرور عبر صفوف الصورة**

**For R = 3 To .Width - 3**

**' جملة تكراريّة للمرور عبر نقاط كلّ صفّ**

**For C = 0 To .Height - 3**

**.SetPixel(R, C, RevColor(.GetPixel(R, C)))**

**Next**

**Next**

**PictureBox1.Refresh()**

**End With**

ويقدّم لك مشروع ImageProcessing أربع عمليّات أخرى على الصور.. حاول تجربته.

**التعامل مع نقط الصورة في الذاكرة مباشرةً:**

إنّ الوسيلتين GetPixel و SetPixel تتسمان بنوع ملحوظ من البطء، حتّى إنّك ستلاحظه في عمليّة تافهة كعكس الألوان.. المشكلة أنّ التعامل مع الصور لا يكون بنفس بساطة برنامج عكس الألوان، فلو فحصت العمليّات الأخرى الموجودة في مشروع ImageProcessing، فستجد أن تغيير قيمة أيّ نقطة يستلزم إجراء بعض العمليّات الحسابيّة على النقاط الثمانية المحيطة بها من الجهات الأربع (وأحيانا تمتدّ العمليات إلى غير ذلك من النقاط).. في هذه الحالة سيكون من العبث استدعاء الوسيلة GetPixel تسع مرّات لتغيير كل نقطة!

وأبسط تفكير يرِدُ على الخاطر في هذه الحالة، هو قراءة كل نقاط الصورة مرّة واحدة وحفظها في مصفوفة، والتعامل بعد ذلك مع المصفوفة عند الحاجة لقراءة أيّ نقطة.

ولكنّ هذا لن يحلّ مشكلة البطء نهائيّا، كما أنّه يستهلك الذاكرة!

لهذا فإنّ لدينا حلاّ آخر.. ما رأيك لو تعاملنا مع النقاط في أماكن تخزينها في الذاكرة مباشرة؟

إنّ ذلك سيكون صعبا نوعا ما، وفيه بعض المخاطرة، فمن الممكن أن تؤدّى إلى إغلاق البرنامج أو الويندوز كلّه، لو كتبت في منطقة خاطئة من الذاكرة، فأفسدت بعض بيانات البرامج الأخرى أو النظام.

توكّل على الله وهيّا اتبع معي هذه الخطوات:

في البداية تعال نختار الجزء الذي نريد التعامل معه من الصورة، ونضعه في كائن بيانات الصورة النقطيّة BitmapData.. ولاستخدام هذا الكائن، لا تنسَ أن تكتب جملة الاستيراد التالية في مشروعك:

**Imports System.Drawing.Imaging**

عرّف كائن صورة نقطيّة وضع فيه صورة مربّع النصّ:

**Dim Bitmap As New Bitmap(PictureBox1.Image)**

بعد ذلك عرّف مستطيلا له نفس مساحة الصورة:

**Dim Rect As New Rectangle(0, 0, Bitmap.Width, Bitmap.Height)**

ثمّ عرّف كائن بيانات الصورة النقطيّة كالتالي:

**Dim BitmapData As New BitmapData()**

ثمّ ضع الصورة في كائن بيانات الصورة النقطيّة باستخدام الوسيلة "إغلاق خانات الصورة" LockBits:

**BitmapData = Bitmap.LockBits(Rect, ImageLockMode.WriteOnly, \_**

**Bitmap.PixelFormat)**

هذه الوسيلة تمنع تغيير موضع الصورة في الذاكرة (بفعل جامع القمامة GC أو الويندوز) إلى حين الانتهاء من التعامل معها.. وهي تأخذ ثلاثة معاملات: أوّلها مستطيل يحدّد المساحة التي ستمنع التعامل معها من الصورة، والثاني نوع الإغلاق، وقد اخترنا منع الكتابة على الصورة، والثالث هو نوع تنسيق الصورة.

والآن صار كائن بيانات الصورة يشير إلى بيانات الصورة في الذاكرة، ولكي نحصل على عنوان أوّل نقطة في الصورة في الذاكرة، سنستخدم الوسيلة Scan0.. حيث سنقوم بحفظ العنوان في متغيّر مخصّص لهذا النوع من البيانات، هو المؤشّر Pointer من النوع مؤشّر صحيح IntPtr:

**Dim pixels As IntPtr = BitmapData.Scan0()**

طبعا هذه أوّل مرّة يتعامل فيها مبرمج VB مع المؤشّرات مباشرة (بدون دوال API).. ولا بدّ أنّ التساؤل الذي يحاصره هو: وماذا سنفعل بالمؤشّر و VB لا يتعامل مع الذاكرة مباشرة؟

لقد كان ذلك في الماضي، أمّا الآن، فإنّ لديك خليّة تسمّى Marshal، تمّكنك من القراءة من الذاكرة والكتابة فيها، عن طريق المؤشّرات.. ابدأ بتعريف متغيّر من هذه الخليّة كالتالي:

**Dim M As System.Runtime.InteropServices.Marshal**

وهي تمنحك عشرات الوسائل للتعامل مع الذاكرة، ولكنّ ما يهمّنا هنا هو القراءة والكتابة في الذاكرة.. وفي هذا الصدد لدينا العديد من الوسائل المتماثلة، التي لا تختلف إلا في عدد الوحدات التي تقرأها أوتكتبها في الذاكرة.. فللقراءة يمكنك استخدام:

اقرأ وحدة **ReadByte** ـ اقرأ عدد صحيح قصير **ReadInt16** ـ اقرأ عدد صحيح **ReadInt32** ـ اقرأ عدد صحيح طويل **ReadInt64**.

وكلّ هذه الوسائل تأخذ معاملين: المؤشّر الذي يشير لموضع الذاكرة، وعدد الوحدات التي تريد إضافتها لهذا المؤشّر Offset، حتّى تتمكّن من قراءة المواضع التالية له.. وطبعا تعيد كلّ وسيلة نفس نوع البيانات الذي تقرأه.

وللكتابة في الذاكرة استخدم الوسائل:

اكتب وحدة **WriteByte** ـ اكتب عدد صحيح قصير **WriteInt16** ـ اكتب عدد صحيح **WriteInt32** ـ اكتب عدد صحيح طويل **WriteInt64**.

وتأخذ هذه الوسائل نفس معاملي وسائل القراءة، بالإضافة لمعامل ثالث، هو القيمة التي تريد أن تكتبها في الذاكرة، مع مراعاة أن تكون من نفس النوع الذي تكتبه الوسيلة.

بقي شيء هامّ، هو معرفة كيف تحفظ النقاط في الذاكرة:

إنّ كل نقطة تأخذ عددا من وحدات الذاكرة Bytes يختلف على حسب نوع تنسيق الصورة.. لقد افترضنا هنا أنّ الصورة ملوّنة، وأنّ كلّ نقطة من نقاطها تخزّن في أربع وحدات ذاكرة.

ويجب أن تعرف، أنّ التعامل مع الذاكرة سيختلف عن التعامل مع الصورة، فقد كنا نحصل على النقطة بمعلوميّة الصف والعمود.. ليس في الذاكرة هذا التقسيم، وإنّما يتمّ حفظ نقاط الصورة على التوالي: توضع نقاط الصفّ الأوّل، ثمّ يليها نقاط الصف الثاني، فالثالث وهكذا...

افترض أنّ هذه هي خانات الصورة (سنكتب في كلّ خانة رقمها لللإيضاح):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 |

هكذا تخزّن في الذاكرة:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

فإذا علمت أنّ كل نقطة تخزّن في 4 وحدات ذاكرة، فإنّ هذا يعني أنّ أوّل نقطة في الصف الثاني من الصورة (لاحظ أنّ رقمه 1، لأنّ الصفّ الأوّل رقمه صفر) ستوضع في خانة الذاكرة التي تساوي: عرض الصورة (5 خانات في مثالنا هذا) × 4، والتي تليها (لاحظ أنّ رقم عمودها 1، لأنّ العمود الأوّل رقمه صفر) ستكون في الخانة رقم: عرض الصورة × 4 + 1 × 4، وهكذا .. وبصورة عامّة:

**الخانة رقم ع في الصفّ الثاني ستوضع في خانة الذاكرة رقم: 4 × (عرض الصورة + ع).**

كما أنّ أوّل خانة في الصفّ الثالث ستوضع في الخانة رقم: 4 × (2 × عرض الصورة)، والتي تليها ستكون في الخانة: 4 × ( 2 × عرض الصورة + 1).

وبصورة عامّة: لمعرفة موضع نقطة في الذاكرة، تقع في الصورة في الصفّ ص والعمود ع، طبّق المعادلة:

**رقم الخانة في الذاكرة = (ص × عرض الصورة + ع) × عدد الوحدات التي تمثّل النقطة (4).**

والآن تعال نكتب دالّة عكس الألوان بهذه الطريقة:

**Dim X, Y, C, O As Integer**

**Dim W As Integer = Rect.Width**

**Dim Clr As Color**

**For Y = 0 To Rect.Height - 1**

**For X = 0 To W – 1**

**' حساب موضع النقطة في الذاكرة**

**O = CInt(Y \* W \* 4 + X \* 4)**

**' قراءة عدد صحيح من الذاكرة يمثّل اللون**

**C = M.ReadInt32(pixels, O)**

**' تحويل اللون من عدد صحيح إلى كائن لون**

**Clr = Color.FromArgb(C)**

**' عكس مكوّنات اللون وتكوين اللون المعكوس**

**Clr = Color.FromArgb(Clr.A, 255 - Clr.R, \_**

**255 - Clr.G, 255 - Clr.B)**

**' تحويل كائن اللون إلى عدد صحيح**

**C = Clr.ToArgb**

**' تخزين اللون المعكوس في الذاكرة**

**M.WriteInt32(pixels, O, C)**

**Next**

**Next**

ولا تنس في النهاية تحرير المساحة التي أغلقناها في الذاكرة، مع وضع كائن الصورة النقطيّة في مربّع الصورة وإنعاشه، حتّى ترى التغيير:

**Bitmap.UnlockBits(BitmapData)**

**PictureBox1.Image = Bitmap**

**PictureBox1.Refresh()**

ستجد هذا الكود كاملا في الأمر Inverse1 تحت القائمةProcess في مشروع ImageProcessing.. حاول أن تقارن بين هذه الطريقة وبين الطريقة التقليديّة التي كتبنا بها الأمر Inverse في نفس القائمة.. ستجد الفارق في السرعة واضحا (وصل معي إلى خمسة أضعاف)، وسيتّضح أكثر، لو كتبت العمليّات الأربع الأخرى الموجودة في نفس القائمة بالطريقة الجديدة، فهي بالفعل تأخذ وقتا شنيعا!

ولو شئت تعديل كود عكس الألوان ليتعامل مع صور بتنسيقات مختلفة، تُخزَّن فيها النقطة في وحدة Byte أو اثنتين أو ثلاثة أو أربعة، فاستخدم الوسيلة "قراءة حجم تنسيق النقطة" GetPixelFormatSize، حيث يمكنك أن تعرف بها عدد الوحدات التي تخزّن فيها كلّ نقطة كالتالي:

**N = Bitmap.GetPixelFormatSize(Bitmap.PixelFormat)**

وبناء على هذا الرقم، عدّل الكود ليقرأ N من وحدات الذاكرة في كلّ مرّة بدلا من 4.. لن يكون ذلك بسيطا، ولكنّه أيضا لن يكون معقّدا.. اعتبره تدريبا لك.. هيّا يا بطل!

**تغير لون معيّن في الصورة إلى لون آخر:**

لحسن الحظّ، لن تحتاج إلى المرور على كل نقاط الصورة لونا لونا، بحثا عن اللون الذي تريد تغييره.. إنّ لدينا طريقةً جاهزةً لفعل ذلك.. اتّبع الخطوات التالية:

ابدأ بتعريف كائن الصورة النقطيّة، وضع به الصورة التي تريد تغييرها (ولتكن تلك الموجودة في مربّع الصورة):

**Dim image As New Bitmap(PictureBox1.Image)**

الآن سنعرّف كائنا يسمّى خريطة اللون ColorMap.. لهذا الكائن خاصيتان هامّتان:

**اللون القديم OldColor:**

وهو الذي تريد إزالته من الصورة.

**اللون الجديد NewColor:**

وهو الذي تريد استبداله باللون القديم.

وبإمكانك أن تعرّف مصفوفة من هذا الكائن، بحيث تستبدل مجموعة من الألوان بغيرها مرّة واحدة.. وفي مثالنا هذا سنعرّف مصفوفة من خانة واحدة، نستبدل بها اللون الأزرق (اللون الجديد) باللون الأحمر (اللون القديم):

**Dim colorMap() As ColorMap = {New ColorMap ()}**

**colorMap(0).OldColor = Color.Red**

**colorMap(0).NewColor = Color.Blue**

ثمّ عرّف كائنا من النوع سمات الصورة ImageAttributes:

**Dim imageAttributes As New ImageAttributes()**

الآن سنستخدم الخاصيّة "تغيير خريطة الألوان" SetRemapTableالخاصّة بكائن سمات الصورة لجعل مصفوفة خريطة التغيير جزءا من سمات الصورة:

**imageAttributes.SetRemapTable(colorMap)**

والآن لم يبقَ إلا أن تعيد رسم الصورة، حتّى تبدو التعديلات بها.. وسنستخدم في هذا وسيلة رسم الصور DrawImage الخاصّة بكائن الرسوم.. ومن الطبيعيّ أن نرسم بها على الصورة النقطيّة، حتّى تظلّ الصورة المرسومة ثابتة كما شرحنا من قبل:

**Dim W As Integer = PictureBox1.Width**

**Dim H As Integer = PictureBox1.Height**

**Dim G As Graphics**

**' إنشاء كائن رسوم من الصورة النقطيّة**

**G = G.FromImage(image)**

**' الرسم على الصورة النقطية بالسمات التي ضبطناها، وبذلك نغيّر اللون الأحمر بالأزرق**

**G.DrawImage(image, New Rectangle(0, 0, W, H), \_**

**0, 0, W, H, GraphicsUnit.Pixel, imageAttributes)**

**' وضع الصورة في مربّع الصورة**

**PictureBox1.Image = image**

**' إنعاش مربّع الصورة**

**PictureBox1.Refresh()**

هل تبدو صيغة وسيلة رسم الصورة DrawImage جديدة عليك؟.. لا عليك، فإنّ لها 30 صيغة مختلفة!

وهذه هي المعاملات بالترتيب:

* الصورة التي سنرسمها.
* المستطيل الذي يحدّد المساحة التي سنرسم فيها.
* المعاملات الأربعة التالية تحدّد المساحة التي سنرسمها من الصورة، وهي بالترتيب: الإحداثيين الأفقي والرأسي لزاوية المستطيل العليا اليسرى، وعرض المستطيل، وارتفاعه.
* وحدة قياس الأطوال، وفي حالتنا هذه هي النقطة Pixel.
* وأخيرا المعامل الذي يهمّنا والذي فعلنا كلّ ذلك من أجله: سمات الصورة، حيث أرسلنا كائن سمات الصورة الذي يحمل خريطة تغيير الألوان.

ويمكنك أن تجرّب هذا المثال في مشروع ImageProcessing، في الأمر ChangeColor في قائمة Processing، بشرط أن تضع في مربّع الصورة صورة بها مساحة حمراء، حتّى تشعر بحدوث اختلاف.

لا بدّ أنّه قد ساءك أنّ المستخدم مجبر على تغيير اللون الأحمر إلى الأزرق.. فماذا لو أراد أن يغيّر أيّ لون في الصورة إلى أيّ لون آخر؟

كتدريب خاصّ، حاول تعديل الكود الخاص باستبدال الألوان، ليسمح للمستخدم باختيار اللون الذي يريد تغييره، واختيار اللون البديل، وذلك باستخدام الأداة ColorDialog.

وعلى فكرة: يمكنك تغيير أيّ لون ليصير شفافا Color.Transrarent، أو ليصير أي لون به نسبة شفافية (باستخدام معامل الشفافية في دال تكوين اللون Color.FromARGB).. هذه نقطة مفيدة جدّا.

**رسم الدوالّ Plotting Functions:**

سنختتم هذا الفصل بهذا الموضوع، الذي ستجده شيّقا بإذن الله.. ولمن لا يحبّون الرياضيّات، يمكنهم ألا يكتئبوا، فهذا الجزء بمثابة مشروع تدريبيّ ليس أكثر، وإن كنت أنصحهم بقراءته على كلّ حال، فسنتعرّف فيه على أشياء جديدة، كما سنتعامل فيه مع أداة هامّة، قد تفيدهم فيما بعد.

فلنبدأ:

افترض أنّ لديك المعادلة:

ص = 2س.

والآن لرسم هذه الدالة، لا بدّ أن نوجد قيم ص المناظرة لكلّ قيم س (0، 1، 2، 3، ... إلخ).

ثم نقوم برسم النقاط الناتجة، حيث إنّ كلّ نقطة هي عبارة عن الإحداثيين (س، ص).. فمثلا في دالتنا هذه ستكون النقطة على الصيغة (س، 2س).. مثل: (0، 0) ، (1، 2) ، (2، 4) ... إلخ.

ولا بدّ أن نصل بين هذه النقاط بخطوط مستقيمة، حتّى يبدو المنحنى متصلا.

وهنا ستبرز هذه المشاكل:

1. إذا كان معدّل تغيّر الدالة كبيرا، فستجد أنّ النقاط متباعدة، بحيث ستكون الخطوط الواصلة بينها واضحة للعيان، وسيبدو المنحنى متكسرا غير ناعم.. لحلّ هذه المشكلة نحتاج لتغيير مقياس رسم المحور السيني (الأفقيّ).. وللقيام بذلك نطبّق المعادلة التالية:

**مقياس الرسم الأفقي = عرض مربّع الصورة ÷ (أكبر س – أصغر س).**

فمثلا: لو كنا سنرسم الدالة لقيم س بين -1 و 5، فسيكون:

مقياس الرسم = عرض مربّع الصورة ÷ (5 – [-1])

= عرض مربّع الصورة / 6.

1. إذا كانت قيم ص الناتجة من الدالة كبيرة، فإنّ كثيرا منها لن يظهر في منطقة الرسم.. ولحلّ هذه المشكلة، نحتاج لتغيير مقياس رسم المحور الصادي (الرأسيّ).

**مقياس الرسم الرأسيّ = ارتفاع مربّع الصورة ÷ (أكبر ص – أصغر ص).**

1. إذا كانت هناك قيم سالبة على المحور السينيّ أو الصاديّ، فإنّها لن تظهر في مساحة الرسم (لأنّ إحداثيات النموذج والأدوات تبدأ من (0، 0).. ولحلّ هذه المشكلة، يجب نقل الإحداثيّات، حتّى تصبح أكبر قيمة سالبة منطبقةً على الصفر.

وفي حالتنا هذه، سنقوم برسم النقط في كائن مسار الرسوم GraphicsPath، حيث سنصل بخطّ مستقيم، بين كلّ نقطة جديدة نضيفها إليه والنقطة السابقة لها، ثمّ سنرسم كائن المسار مرّة واحدة.. إنّ هذه الطريقة أسرع من رسم كلّ نقطة على حدة، كما أنّها تمكّنك من استخدام المتغيّر الذي يشير لكائن مسار الرسوم لإعادة رسم الدالة أكثر من مرّة بمقاييس رسم مختلفة.

ولكن كيف نقوم بهذه التحويلات؟

قلنا إنّ عمليات التحويل يتّم حفظها في مصفوفة التحويل.. لهذا سنستخدم هذه المصفوفة في حالتنا هذه، وسنجري عليها التحويلات مباشرة، وأوّل ما سنفعله هو تعريف متغيّر من مصفوفة التحويل:

**World = New System.Drawing.Drawing2D.Matrix()**

**ملحوظة:**

هذه المصفوفة تتكوّن من 3 صفوف و 3 أعمدة، وهي تمتلك وسائل جاهزة لتدوير المصفوفة Rotate وعكسها Invert وضربها في مصفوفة أخرى Multiply.. ولقراءة محتويات مصفوفة التحويلات والحصول على مصفوفة عاديّة Array 3×3 استخدم الخاصيّة Elements.

بعد ذلك سنغيّر مقياس الرسم أفقيّا ورأسيّا:

**World.Scale(((PictureBox1.Width - 4) / (Xmax - Xmin)), \_**

**-(PictureBox1.Height – 4) / (Ymax - Ymin))**

لاحظ أنّنا طرحنا 4 من عرض وارتفاع مربّع الصورة، حتّى نزيل عرض الحوافّ ونترك هامشا بينه وبينها.. لاحظ أيضا أن أصغر وأكبر قيمتين على المحور الأفقي Xmin و Xmax هما اختياريتان، على حسب المدى الذي تريد رسم الدالّة فيه.. أمّا أصغر وأكبر قيمة على المحور الرأسي فلا بدّ من حسابهما أولا، سواء بالوسائل الرياضيّة التي تحسب القمّة العظمى والصغرى (سيحتاج هذا لإجراء عمليات تفاضل، وهذا خارج نطاقنا الآن)، أو بجملة تكراريّة تحسب كل قيم ص المناظرة لقيم س، بحيث تلتقط من بينها أصغر قيمة وأكبر قيمة.

بعد ذلك سنحرّك الإحداثيّات لتظهر كلّ النقط في منطقة الرسم:

**World.Translate(-Xmin, -Ymax)**

لاحظ أنّ الإحداثيّات في الكتب العلميّة والرياضيّة تفترض أنّ القيم على المحور الرأسي تزداد لأعلى وتقلّ لأسفل، لهذا فقد نقلنا الإحداثيّ الرأسيّ لأسفل بمقار أكبر قيمة له، بحيث تظهر هذه القيمة في أعلى الشاشة (ارتفاعها = 0 ).

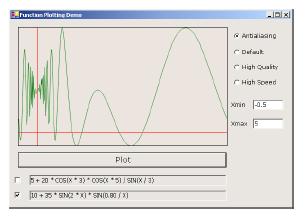
الآن أصبحت مصفوفة التحويلات تحتوي على كل التحويلات المطلوبة، ولكي تنطبق هذه التحويلات على كائن مسار الرسوم، فإنّنا نستدعي الوسيلة Transform الخاصّة به، ونرسل هذه المصفوفة كمعامل لها، ثمّ بعد ذلك نرسمه:

**GraphicPath1.Transform(World)**

**G.DrawPath(plotPen, plot1)**

أعتقد الآن أن التساؤل الذي دار في ذهنك قد أجيب: إنّ الفارق بين استخدام مصفوفة التحويلات ووسائل التحويل الخاصّة بكائن الرسوم Graphics، هي أنّ مصفوفة التحويلات خاصّة فقط بالكائن الذي يستخدمها (كائن مسار الرسوم هنا)، أمّا وسائل التحويل الخاصّة بكائن الرسوم فهي عامّة لكل منطقة الرسم.

ولديك في مجلد برامج هذا الفصل، المشروع Plotting، وفيه سترى كيف يمكن رسم دالتين مختلفتين بواسطة المفاهيم التي شرحناها هنا.



أمّا لو أردت أن تسمح للمستخدم بتعريف الدالة التي يريد رسمها (بالصيغة الإنجليزيّة)، يمكنك استخدام الأداة Script ActiveX control.. ولإضافة هذه الأداة لبرنامجك، اضغط بزر الفأرة الأيمن على كلمة المراجع References في متصفّح المشاريع، ومن القائمة الموضعيّة اختر Add Reference، وفي مربّع الحوار الذي سيظهر اضغط الشريط COM.. ومن بين العناصر التي تملأ القائمة انقر مرتين على العنصر Microsoft Script Control 1.0.. بعد ذلك أغلق مربّع الحوار.

الآن يمكنك كتابة دالة تحسب لك قيمة الصيغة التي عرّفها المستخدم عند قيمة معيّنة لـ X، باستخدام إحدى نسخ VB المخصّصة لصفحات الإنترنت، وهي لغة VBScript، كالتالي:

**Function FVal(X As Double, Formula As String) As Double**

**Dim S As New MSScriptControl.ScriptControl**

**S.Language = "VBScript"**

**Try**

**S.ExecuteStatement("X=" & X)**

**Return CDbl(S.Eval(Formula))**

**Catch exc As Exception**

**Throw New Exception("لا يمكن حساب الدالة عند X=" & X)**

**End Try**

**End Function**

والآن جرّب استدعاء هذه الدالة كالتالي:

**Dim F As String**

**F = "5 + 20 \* Cos(X \* 3) \* Cos(X \* 5) / Log(Abs(Sin(X)) / 10)"**

**MsgBox(FVal(3, F))**

عند تنفيذ هذا الكود، ستظهر لك رسالة تجمل الرقم: 1.75091032730126.

جرّب وضع مربّع نصّ على نموذج، واكتب فيه صيغة الدالة (بالإنجليزيّة) ومربّع نصّ آخر تكتب فيه قيمة X التي تريد أن تعرف قيمة الدالة عندها، وزرا اكتب فيه ما يلي:

**MsgBox(FVal(CDbl(TextBox2.Text), TextBox1.Text))**

وفي ضوء هذا، حاول أن تطوّر مشروع رسم الدوال Plotting، ليسمح للمستخدم بتعريف صيغة الدالة التي يريدها.

عامّة ستجد هذا متحقّقا بالفعل في المشروع FunctionPlotting.

ولو أرد تدريبا جيّدا، فعليك أن تكتب دالة تحوّل الصيغ العربيّة إلى الصيغ الإنجليزيّة، مثل:

|  |  |
| --- | --- |
| ÷ | / |
| × | \* |
| جا | Sin |
| جتا | Cos |
| |......| | Abs(……) |
| لوج | Log |

ويفضّل أن تستخدم في هذا محرّك "التعبيرات النمطيّة" Regular Expression الذي يمنحه لك إطار العمل.. (شرح هذا خارج عن نطاق هذا الكتاب.. أنا نفسي ما زلت أبحث عن مرجع بالإنجليزيّة (الكتروني وليس ورقيّا) في هذا الموضوع.. من لديه معلومات عن مثل هذا المرجع، رجاءً أن يرسل لي على البريد الإلكترونيّ).