

المبداج الأول

مفاهيم أساسية

- ١ . التعرف على *Visual Basic* داخل تقنية *NET*..
- ٢ . الجدياء فى *Visual Basic 2008*.
- ٣ . مءءءل إلى *Visual Basic 2008*.

الفصل الأول
التعرف على Visual Basic
داخل تقنية .NET

ربما تتساءل عن بيئة عمل .NET Framework وعن
علاقتها بموضوع الكتاب وهو Visual Basic 2008 وعن
الأهداف التي من أجلها تم تطوير هذه التقنية. وهذا ما سنحاول
سريعاً التعرف عليه من خلال هذا الفصل.
بانتهاء هذا الفصل ستتعرف على:

- ◆ نلحة عن تطور Visual basic.
- ◆ ما هي تقنية .NET ؟
- ◆ Visual Basic وبيئة .NET Framework.
- ◆ نموذج .NET Framework التنفيذي.

لمحة عن تطور Visual Basic

كانت **Visual Basic 6** برنامجاً مستقلاً ، أما **Visual Basic 2008** فهي عبارة عن لغة في نظام تطوير أكبر، فإذا عدنا إلى الوراء للتعرف على أصول **Visual Basic** فقد كانت لغة برمجية يتم استخدامها منذ عشرين عاماً كجزء من **MS-Dos**. وفي عام ١٩٨٥ تحولت **Basic** إلى **Visual Basic** كما أصبحت جزءاً من أداة إنشاء التطبيقات الخاصة بـ **Windows**. أما بخصوص برنامج **Visual Basic 6** فهو ليس مجرد لغة – بل كان يمكنه إنشاء النماذج الخاص به – على سبيل المثال – أما بالنسبة لـ **Visual Basic 2008** فيملك أداة إنتاج نماذج جديدة توفر له وسيلة جديدة للتفاعل مع نظام التشغيل **Windows** تلك هي بيئة **.NET Framework** التي تحيط بلغة **Visual Basic 2008**.

عندما تنظر إلى **Visual Basic** فمن المحتمل أن تفكر في تطبيقات **Windows** الخاصة بـ **Microsoft**. فعلى مدى خمسة عشر عاماً استخدم المطورون **Ruby Forms Engine** الخاصة ببرنامج **Visual Basic**. لكتابة تطبيقات العمل الشائعة، وعندما كان يتم تعريف برنامج كبرنامج **Visual Basic**، فهذا معناه أنه تطبيق **Windows** بشكل افتراضي.

لكن زاد مجال **Visual Basic** واستخدامها مع ظهور **Visual Basic.NET** و **Visual Basic.NET** هي عبارة عن لغة مثل **C++** أو **COBOL** أو **Java** والتي يمكنك استخدامها لكتابة أى نوع من التطبيقات مدعم من قبل (Application programming interface) **API**. والآن عندما يتم تعريف برنامج كبرنامج **Visual Basic**، فيجب أن تطرح الأسئلة التالية : ما نوع البرنامج ؟ هل هو تطبيق **Windows** أو موقع ويب؟ هل هو خدمة **Windows** أو خدمة ويب **XML**؟

يمكن القول أنه ما زلت هناك حاجة كبيرة لإنشاء تطبيقات **Windows** – والتي يطلق عليها الآن تطبيقات **Windows Forms**. فبالرغم من أن تطبيقات الويب تزداد الحاجة

إليها يوماً بعد يوم، فإن بيئة العمل الفعالة لأي تطبيق Windows ليست مألوفة فحسب، بل لها أيضاً العديد من المزايا التي من الصعب أن تتوفر بالنسبة لأي نوع تطبيق آخر. في بيئة .NET. أصبحت اللغة البرمجية مجرد وسيلة للتفاعل مع بيئة العمل وبالتالي مع نظام تشغيل Windows فجميع البرامج تحتاج إلى مجموعة من القواعد المحددة (بالنسبة لاتخاذ القرار وعمليات التكرار وما شابه) داخل البرامج للحصول على التطبيق المطلوب توفر لغة Visual Basic مثل هذه المجموعة من القواعد في حين توفر بيئة العمل الكائنات والأحداث التي سيتم التعامل معها.

إصدارات Visual Studio

توجد بيئة تطوير Visual Studio في عدد من الإصدارات كالتالي :

- **Team System** : لقد تم تصميم هذا الإصدار لفرق البرمجة في الشركات الكبرى. وهو يتضمن أدوات تصميم على نطاق كبير مثل التطوير المعتمد على عمليات الاختبار و **Team Foundation Server**.
- **Professional Edition**: يعتبر هذا الإصدار شائعاً بصورة كبيرة بالنسبة للمطور المنفرد أو لتطوير تطبيقات متوسطة الحجم، وهذا هو الإصدار المستخدم في هذا الكتاب.
- **Standard Edition** : تم تصميم هذا الإصدار لإنشاء تطبيقات مستقلة أصغر حجماً وهو يعمل بشكل جيد في 80 بالمائة من التطبيقات المنشأة به. ولكن إذا كنت تخطط لإنشاء نظم أكبر تحتاج أن تكون على نطاق أوسع، وأن تكون للعديد من المستخدمين، فاستخدام إصدار **Professional Edition**

ما هي تقنية .NET ؟

أرادت مايكروسوفت من خلال تقنية .NET. توفير بنية تحتية مشتركة لجميع المطورين والمستخدمين على حد سواء تهدف أساساً إلى تحويل نظرة البرمجة والبرمجيات من مفهوم الحاسبات الشخصية المستقلة أو الشبكات الصغيرة إلى مفهوم إتاحة البيانات

وعرضها في جميع أنحاء العالم من خلال شبكة الإنترنت. وقد أطلقت مايكروسوفت في البداية على هذا النطاق الجديد المصطلح **Next Generation Windows Services(NGWS)** ثم عادت وأسمته **.NET**.

يحتوى **.NET Framework** (نطاق **.NET**) حقيقةً على وفرة هائلة من التصنيفات والبنية التحتية والأدوات التي تمكن المطورين من تطوير تطبيقات عالية الدقة والجودة بسهولة منقطعة النظر مقارنةً بالإصدارات السابقة، حيث يحتوى هذا النطاق على عدد من لغات البرمجة القوية والشهيرة مثل **Visual C++ 2008** ولغة **Visual C# 2008** إلى جانب لغة **Visual Basic 2008** موضوع هذا الكتاب، بالإضافة إلى تضمين **ADO.NET** و **ASP.NET** كعناصر أساسية داخل النطاق.

وقديماً كنت في حاجة إلى استخدام وإتقان لغة واحدة من لغات البرمجة إذا أردت تطوير تطبيق من التطبيقات يعمل على الأجهزة الشخصية أو من خلال خادم إحدى الشبكات. أما الآن فالأمر تغير إلى حد ما، حيث يلزمك التعرف على العديد من التقنيات والمهارات مثل صفحات الخادم النشطة (**Active Server Pages (ASP)**) ومكونات **COM** و **HTML** و **XML** و **VBScript** و **JScript** وغيرها من التقنيات الأخرى إذا أردت تطوير تطبيق متعدد الأطراف **n-tier Application**. ومن حسن الحظ تضمين جميع هذه التقنيات داخل نطاق **.NET** (**.NET Framework**) كما يمكنك داخل **.NET Framework** (نطاق **.NET**) اختيار اللغة التي تشعر تجاهها بالراحة كما يمكنك إنشاء وتطوير تطبيقك باستخدام أكثر من لغة، حيث تتيح **.NET** التداخل بين اللغات.

وإذا كان الموطن الطبيعي للبرامج الآن هو الحاسبات الشخصية، فإن استخدام **.NET** في تطوير التطبيقات والبرامج يعمل على نقل موطن هذه التطبيقات أو تلك البرامج إلى أعلى البحار ليكون داخل الويب وهو ما يعنى إتاحتها للعالم بآثره. والهدف من هذا الكتاب أن تتعرف على الكيفية التي يتعامل بها **Visual Basic 2008** مع بيئة عمل **.NET Framework** الخاصة بنظام **Windows**.

تحويل البرامج إلى خدمات

قديمًا كنا نطلق على البرامج أو السمات المتاحة لشريحة من الناس كلمة "خدمة" **Service** مثل برنامج إظهار رقم الطالب الموجود بشركات الاتصالات فهو عبارة عن خدمة عامة بأجر رمزي معين وكذلك خدمة البريد الإلكتروني المتاحة عبر الويب. وقد أرادت مايكروسوفت من خلال **.NET** معاملة جميع البرامج كخدمات متاحة لجميع المستخدمين من خلال شبكة الإنترنت. فمثلًا إذا أردت في هذه الأيام اقتناء أحد برامج تحرير النصوص كبرنامج **Word** مثلًا، تقوم بشراء البرنامج على قرص مدمج ثم تثبته على حاسبك من خلال هذا القرص المدمج. أما مع استخدام **.NET** فالأمر مختلف، حيث يمكن إتاحة محرر النصوص كخدمة مشتركة من خلال الإنترنت ومن ثمّ يمكنك تثبيته على حاسبك أو يمكنك بطريقة أخرى تشغيل المحرر من خلال الإنترنت، على أن تقوم بدفع مقابل يتناسب مع الإمكانيات التي تستخدمها فقط.

ولعلك الآن تتساءل، أليس هذا متاحًا داخل الشبكات الصغيرة حيث يمكنك استخدام البرامج المثبتة على الخادم كما لو كانت موجودة على حاسبك المتصل بالشبكة؟. وكلامك صحيح مائة في المائة، إلا أن الويب يختلف اختلافاً كبيراً عن الشبكات الصغيرة. فكل شبكة يكون لها نظام تشغيل معين ونموذج مكونات معين وخصائص عديدة مشتركة إلى جانب عدد معين من الحاسبات المتصلة بهذه الشبكة. أما شبكة الإنترنت أو الويب فتتكون من أعداد هائلة من الحاسبات ذات أنظمة التشغيل والخصائص المختلفة. لذا فإن أخذ مبدأ التجانس في الحسبان سيزيد المشكلة تعقيداً. فإذا وجدت الخدمة على أحد الأجهزة البعيدة، فإن الجهاز الذي يقوم باستخدام هذه الخدمة لا يعرف عنها أو عن خصائصها شيئاً، وإنما يتم التأكد من أن كلا الحاسبين يفهم معنى البيانات المرسلّة من أحدهما إلى الآخر وهذا يتم من خلال لغة عالمية موحدة تمكن من الاتصال السهل والسريع بين المرسل والمستقبل وهذا يتحقق حقيقةً من خلال مبدئين غاية في الأهمية هما لغة الترميز الممتدة **Extensible Markup Language (XML)** والبروتوكول **Simple Object Access Protocol (SOAP)**.

Visual Basic وبيئة .NET Framework

لقد قامت شركة Microsoft بإنشاء بيئة عمل .NET Framework لتسهيل تطوير التطبيقات المعتمدة على نظام Windows. ولكن نظراً للفروق الموجودة بين الإصدارين Visual Basic 6.0 و Visual Basic .NET من لغة Visual Basic (حيث تعتبر Visual Basic .NET بمثابة أول إصدار .NET)، فقد وجد معظم مطوري Visual Basic أن التطوير أصبح أكثر صعوبة. فعلى سبيل المثال، قام الإصدار Visual Basic .NET بتحويل جميع المتغيرات إلى كائنات، الأمر الذي قضى على قدرة المبرمج على تعريف نوع متغير بشكل مباشر.

لكن تطوير التطبيقات داخل بيئة عمل .NET Framework ليس أكثر صعوبة مما كان عليه في الإصدار 6.0 من Visual Basic ومن الممكن أن تكون بيئة عمل .NET Framework Development Environment (IDE) الخاصة بـ Visual Studio .NET.

المريد عن بيئة عمل .NET Framework

تعد بيئة عمل .NET Framework كمفهوم بمثابة مجموعة من البرامج المتصلة ببعضها البعض والتي تم تطويرها من قبل شركة Microsoft والتي تعمل على الربط بين الأشخاص والنظم والأجهزة التي يستخدمونها بالمعلومات التي يحتاجون إليها. وتعد بيئة عمل .NET Framework وتعد بيئة عمل .NET Framework هي بيئة التطوير التي تقوم بكل ما سبق من خلال Visual Basic. وتعد Visual Basic جزء من بيئة .NET Framework.

تتضمن التطبيقات المصممة بصورة جيدة الطبقات التالية:

- تتضمن الأجهزة التابعة في بيئة .NET. أجهزة مثل التليفونات المحمولة وأجهزة PDA وأجهزة الكمبيوتر الشخصية التي تعمل بنظام Windows أو برنامج تصفح الويب على أي نظام تشغيل.

- تعتمد وحدات الخدمة Servers في بيئة .NET على Windows Server أو SQL Server. ويعد نظام تشغيل وحدة الخدمة أقل مرونة من نظام تشغيل الجهاز التابع في عالم .NET. وأحياناً يتم استخدام وحدات خدمة مثل Biztalk أو SharePoint Services. بوجه عام، توفر وحدات الخدمة خدمات متعددة.
 - يوجد بين وحدات الخدمة والأجهزة التابعة خدمات ويب XML أو غيرها من الأدوات الخاصة بالاتصال. وتعد خدمات ويب XML أداة يمكنها العمل على نظم مختلفة لنقل معلومات من وحدات خدمة إلى أجهزة تابعة أو من أجهزة تابعة إلى أجهزة أخرى أو حتى بين الخدمات بعضها البعض.
- أما عن أدوات المطور، فتمثل في Visual Basic و Visual Studio. وتعد Visual Basic هي اللغة، في حين تعد Visual Studio هي الأداة. ويملك المطور أدوات أخرى مثل C++ أو C# أو Jscript

تعتبر الكيفية التي تتفاعل بها Visual Basic مع أجزاء Visual Studio أمراً مهماً للغاية أيضاً؛ فهذا يشكل ماهية عمل البرامج. فالبرنامج سيستخدم الخدمات الموفرة من قبل بيئة عمل .NET Framework. عن طريق الأدوات الخاصة باللغة. ويعد أمر التفاعل هذا أساسياً لكل شئ — وهو الأمر الذي تحتاج إلى التركيز عليه في تخطيطك.



- تمثل المصطلحات التي تواجه الكثير من المبرمجين عند الانتقال إلى بيئة عمل .NET Framework بعض الصعوبات والارتباك، لتجنب الوقوع في الارتباك الناشئ عن هذه المصطلحات نورد فيما يلي أهم المصطلحات المستخدمة في التطوير باستخدام بيئة عمل:
- **Visual Basic 2008**: ونقصد بها لغة البرمجة التي هي موضوع هذا الكتاب، حيث لم يعد بإمكانك تشغيل لغة Visual Basic أو تحميلها ككيان منفصل كما كان يحدث في Visual Basic 6 أو الإصدارات التي قبله. وهي تعتبر إحدى لغات البرمجة التي تعمل تحت بيئة .NET Frame Work.

- **Visual Basic .NET Framework** : عبارة عن الطبقة الموجودة بين اللغة (لغة Visual Basic في هذه الحالة) ونظام التشغيل (إصدار Windows 98 أو ME/2000 أو XP أو Server 2003). وتعمل طبقة .NET Framework على توفير الإمكانيات الوظيفية المطلوبة اعتماداً على أسلوب عمل نظام Windows الذي تعمل عليه. وكذلك مجموعات الملفات الخاصة بإمكانيات وظيفية أخرى (مثل الوصول إلى قاعدة البيانات والعمليات الحسابية).
- **Visual Studio 2008** : عبارة عن الأداة التي يمكنك استخدامها لإنشاء أى نوع من التطبيقات باستخدام أى لغة برمجية متوافقة. حلت بيئة تطوير Visual Studio محل Visual Basic 6.0 والذي كان يعتبر فيما سبق جزءاً من مجموعة برامج Visual Studio (والتي كان يحمل كل مكون بها رقم الإصدار 6.0). عندما تريد كتابة برنامج جديد في بيئة .NET ، يجب أن تقوم أولاً بتشغيل بيئة تطوير Visual Studio 2008 ثم تقوم بتحديد نوع البرنامج الذي تريد كتابته باللغة البرمجية التي ترغب في استخدامها. فمثلاً لإنشاء برنامج Visual Forms يجب استخدام لغة Visual Basic ، ولكتابة تطبيق لـ Smart Device يجب استخدام لغة # C.
- **تطبيق Windows Forms** : هو المصطلح الجديد لتطبيق Visual Basic القياسي. ويشير هذا المصطلح إلى تطبيق تمت كتابته باستخدام بيئة عمل .NET Framework وله واجهة استخدام Windows.
- **تطبيق Web Forms** : مصطلح يطلق على تطبيق له واجهة صفحة ويب مكتوبة باستخدام بيئة عمل .NET Framework. ويشبه إنشاء تطبيق Web Forms إنشاء تطبيق Windows Forms.
- **خدمات الويب** : عبارة عن مجموعات ملفات الفئات المكتوبة باستخدام معيار متعارف عليه من قبل الأشخاص أنفسهم الذين قاموا بتعريف المعايير بالنسبة لشبكة الويب . ويتم استخدام خدمات الويب للتفاعل بين النظم المختلفة.

خلاصة القول من السهل على بيئة عمل **.NET Framework** التعامل مع كل الجوانب الخاصة بنظام **Windows**. فتوفر **.NET Framework** على وجه التحديد - اسماً برمجياً لكل كائن وكل حدث يمكن لـ **Windows** التحكم فيه . ويمكن أن يستخدم المبرمج ذلك الاسم للإشارة إلى أي شيء له كود في نظام التشغيل.

نموذج **.NET Framework** التنفيذي

حينما ترغب في إنشاء أحد التطبيقات باستخدام بيئة تطوير **Visual Studio 2008**، يتم إجراء الخطوات الآتية:

١. يتم أولاً كتابة الكود باستخدام اللغات المتعددة التي يدعمها **.NET**.
٢. بعد ذلك يتم ترجمة هذا الكود من خلال مترجم **.NET** الذي يقوم بتحويل الكود المصدرى للغة المستخدمة إلى تنسيق جديد يسمى تنسيق اللغة الوسيطة **Microsoft Intermediate Language (IL)** أو لغة مايكروسوفت الوسيطة **Microsoft Intermediate Language (MSIL)** حيث يطلق على الملف الناتج من عملية التحويل والذي يحتوى على تنسيق اللغة الوسيطة "الملف التنفيذي الخمول" **Portable Executable (PE)**.
٣. لتنفيذ التطبيق، يتم تحويل تنسيق اللغة الوسيطة إلى كود أصلى **Native Code** وذلك داخل الحاسب المستخدم لتشغيل التطبيق باستخدام مترجم يسمى **Just-in-Time (JIT)**.

ولعلك تلاحظ أننا لم نقم باستخدام المفسرات **Interpreters** أثناء الدورة بالكامل وإنما يتم ترجمة جميع الكود داخل بيئة التشغيل **CLR** وهذا يؤدي بالطبع إلى تشغيل التطبيقات بكفاءة أعلى لأن المفسرات تتسبب في بطء التنفيذ.

اللغة الوسيطة IL

اللغة الوسيطة (IL) Intermediate Language عبارة عن لغة آلة قامت مايكروسوفت بإنشائها وتحتوى على مجموعة من التعليمات والأوامر التي لا تعتمد على نوع المعالج المستخدم والتي يتم تحويلها إلى كود أصلي Native Code. وهذه التعليمات تحتوى على ما يلى:

- العمليات الحسابية والمنطقية
- عبارات التحكم
- الوصول المباشر للذاكرة
- احتواء الاستثناءات (الأخطاء)

كما أن اللغة الوسيطة تمتاز بالميزات التالية:

- يمكنك تشغيل اللغة الوسيطة على أى معالج.
- يمكنك تشغيل اللغة الوسيطة على أى نظام تشغيل.
- تحتوى اللغة الوسيطة على المزيد من الأمان فى الوصول إلى الكود.
- تتيح اللغة الوسيطة تشغيل أكثر من تطبيق داخل نفس مساحة الذاكرة.

ترجمة اللغة الوسيطة إلى الكود الأصلى

لا يمكنك تنفيذ كود اللغة الوسيطة إلا بعد تحويله إلى كود أصلى Native Code وذلك كما يلى:

١. يتم إرفاق كود يسمى Stub code مع كل دالة داخل نوع التصنيف الذى يجرى تحميله.
٢. حينما يتم استدعاء إحدى الدوال، يقوم الكود المصاحب للدالة بتوجيه تنفيذ البرنامج إلى المترجم JIT الذى يقوم بالتحويل من اللغة الوسيطة إلى الكود الأصلى
٣. يتم استبدال الكود المصاحب للدالة بعنوان الدالة داخل الكود الأصلى.

٤. إذا تم استدعاء نفس الدالة مرةً أخرى، يتم تنفيذ الكود الأصلي من الذاكرة مباشرةً دون الحاجة إلى استخدام المترجم JIT مرةً أخرى. وكما ترى من اسم المترجم **Just-in-Time (JIT)** فإن عملية الترجمة تتم فقط وقت الحاجة ولا يوجد مفسر **Interpreter** داخل العملية.

تنفيذ التطبيق

بمجرد الانتهاء من تحويل كود اللغة الوسيطة إلى الكود الأصلي، يتم مباشرةً تنفيذ التطبيق وهنا تظهر عملية تجميع نفايات الذاكرة أو ما يسمى **Garbage Collection**، حيث يقوم البرنامج أثناء تنفيذه باستخدام عدد من الموارد مثل سجلات قاعدة البيانات ووصلات الشبكة وهكذا، ويكون التحكم في هذه الموارد داخل البرنامج أمرًا بالغ التعقيد، لذا يتم التحكم في هذه الموارد باتباع الخطوات الآتية:

١. يتم حجز جزء من الذاكرة بقدر النوع الذى يمثل المورد.
٢. يتم استهلاك المورد، أى تشغيل حالته الابتدائية وتهيئته للاستخدام.
٣. يتم الوصول إلى حالة من تصنيف المورد أو بمعنى آخر استخدامه لأداء الوظيفة المنوطة به.
٤. يتم تحديث حالة المورد دلالةً على إمكانية تدميره لعدم الحاجة إليه بعد الانتهاء من استخدامه.
٥. يتم تحرير جزء الذاكرة الذى استخدمه المورد.

إذا لم يتم تحرير الذاكرة المستخدمة من قبل المورد أو تمت محاولة الوصول إلى المورد بعد تحرير ذاكرته، في كلتا الحالتين يصبح البرنامج غير ثابت ويتصرف تصرفات غريبة. وهنا يأتي دور مجمع النفايات **Garbage Collector** الذى يعد أحد العناصر الأساسية داخل **CLR** والذى يقوم نيابةً عن المبرمج بالتحكم في تحرير الذاكرة والتعامل مع الموارد. ولكن لا يستطيع مجمع النفايات التعرف على وقت تنفيذ الخطوة الرابعة في الخطوات السابقة. فعلى سبيل المثال إذا أراد المبرمج إنهاء الاتصال بالشبكة، فأتى لجمع النفايات أن يعرف ذلك؟. حقيقةً يجب أن يوضح المبرمج ذلك من خلال دالة الهدم **Destructor** الخاصة

بجالة التصنيف أو من خلال الدالة Close() مثلاً بحيث يعرف المجمع فقط رغبة المبرمج في إنهاء استخدام المورد.

