

## الفصل الأول

### التعريف على Visual C++

### داخل تقنية .NET

ربما تتساءل الآن عن بيئة عمل **.NET Framework**. وعن الأهداف التي من أجلها تم تطوير هذه التقنية. وهذا ما سنحاول سريعاً التعرف عليه من خلال هذا الفصل. بانتهاء هذا الفصل ستعرف على:

- ◆ ما هي تقنية **.NET** ؟
- ◆ بيئة عمل **.NET Framework**
- ◆ نموذج **.NET Framework** التنفيذي.

## 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**.

يحتوي **.NET Framework**. 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 يحتوي هذا النطاق على عدد من لغات البرمجة القوية والشهيرة مثل **Visual Basic 2005** و **Visual J# 2005** إلى جانب لغة **Visual C# 2005** ولغة **Visual C++ 2005** موضوع هذا الكتاب، بالإضافة إلى تضمين **ASP.NET** و **ADO.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 (**ASP Active Server Pages**) 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 **.NET Framework**. كما يمكنك داخل **.NET Framework** 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 بأثره. والهدف من هذا الكتاب أن نتعرف على الكيفية التي يتعامل بها Visual C++ 2005 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. فمثلاً إذا أردت في هذه الأيام اقتناء أحد برامج تحرير النصوص كبرنامج Word مثلاً، 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 متاحاً 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 المرسله من

أحدهما إلى الآخر وهذا يتم من خلال لغة عالمية موحدة تمكن من الاتصال السهل والسريع بين المرسل والمستقبل وهذا يتحقق حقيقةً من خلال مبدأين غاية في الأهمية هما لغة الترميز الممتدة **Extensible Markup Language (XML)** والبروتوكول **Simple Object Access Protocol (SOAP)**.

## نموذج .NET Framework التنفيذي

حينما ترغب في إنشاء أحد التطبيقات باستخدام بيئة تطوير **Visual Studio 2005**، 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 المصدري للغة المستخدمة إلى تنسيق جديد يسمى تنسيق اللغة الوسيطة **Intermediate Language (IL)** أو لغة مايكروسوفت الوسيطة **Microsoft Intermediate Language (MSIL)** حيث يطلق على الملف الناتج من عملية التحويل والذي يحتوي على تنسيق اللغة الوسيطة "الملف التنفيذي الخمول" **Portable Executable (PE)**.
٣. لتنفيذ التطبيق، يتم تحويل تنسيق اللغة الوسيطة إلى كود أصلي **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 **Intermediate Language (IL)** 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 يلي:

- 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 أصلي **Native Code** وذلك كما يلي:

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 الأصلى 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 مباشرة 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 الذي يمثل المورد.
  ٢. يتم استهلاك المورد، أي 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 الذي يستخدمه 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** الذي يعد أحد العناصر الأساسية داخل **CLR** والذي 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، فأني لمجمع النفايات أن يعرف ذلك؟. حقيقةً يجب أن يوضح المبرمج ذلك من خلال دالة الهدم **Destructor** الخاصة بحالة التصنيف أو من خلال الدالة **Close()** مثلاً بحيث يعرف المجمع فقط رغبة المبرمج في إنهاء استخدام المورد.

