الفصل الأول

التعرف على Visual C++

داخل تقنية .NET.

ربما تتساءل الآن عن بيئة عمل .NET Framework. وعن الأهداف التى من أجلها تم تطوير هذه التقنية. وهذا ما سنحاول سريعاً التعرف عليه من خلال هذا الفصل.

بانتهاء هذا الفصل ستتعرف على:

- ما هى تقنية .NET. ؟
- بيئة عمل .NET Framework.
- نموذج .NET Framework التنفيذى.

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

يحتوى **.NET Framework** حقيقةً على وفرة هائلة من التصنيفات والبنية التحتية والأدوات التى تمكن المطورين من تطوير تطبيقات عالية الدقة والجودة بسهولة منقطعة النظير مقارنةً بالإصدارات السابقة، حيث يحتوى 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 فى 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 **.NET Framework**. كما يمكنك داخل **.NET Framework** اختيار اللغة التى تشعر تجاهها بالراحة كما يمكنك إنشاء وتطوير تطبيقك باستخدام أكثر من لغة، حيث تتيح **.NET** التداخل بين اللغات.

وإذا كان الموطن الطبيعى للبرامج الآن هو الحاسبات الشخصية، فإن استخدام **.NET** فى

تطوير التطبيقات والبرامج يعمل على نقل موطن هذه التطبيقات أو تلك البرامج إلى أعالى البحار ليكون داخل الويب وهو ما يعنى إتاحتها للعالم بأثره. والهدف من هذا الكتاب أن تتعرف على الكيفية التى يتعامل بها Visual C++ 2005 مع بيئة عمل .NET Framework الخاصة بنظام Windows.

تحويل البرامج إلى خدمات

قديماً 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 رمزى معين وكذلك خدمة البريد الإلكترونى المتاحة عبر الويب. وقد أرادت مايكروسوفت من خلال .NET معاملة جميع البرامج كخدمات متاحة لجميع المستخدمين من خلال شبكة الإنترنت. فمثلاً إذا أردت فى 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 ثمَّ يمكنك تثبيته على حاسبك أو يمكنك بطريقة أخرى تشغيل المحرر من خلال الإنترنت على أن تقوم بدفع مقابل يتناسب مع الإمكانات التى 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 فى 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 فى الحسبان سيزيد المشكلة تعقيداً. فإذا وُجدت الخدمة على أحد الأجهزة البعيدة، فإن الجهاز الذى يقوم باستخدام هذه الخدمة لا يعرف عنها أو عن خصائصها شيئاً، وإنما يتم التأكد من أن كلا الحاسبين يفهم معنى البيانات المرسلة 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 فى الأهمية هما لغة الترميز الممتدة Extensible Markup Language (XML) والبروتوكول Simple Object Access Protocol (SOAP).

نموذج NET Framework. التنفيذى

حينما ترغب فى إنشاء أحد التطبيقات باستخدام بيئة تطوير Visual Studio 2005، يتم إجراء الخطوات الآتية:

١. يتم أولاً كتابة الكود باستخدام اللغات المتعددة التى يدعمها NET.

٢. بعد ذلك يتم ترجمة هذا الكود من خلال مترجم NET. الذى يقوم بتحويل الكود المصدرى للُّغة 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 والذى يحتوى على تنسيق اللغة الوسيطة "الملف التنفيذى المحمول" 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 يؤدى بالطبع إلى تشغيل التطبيقات بكفاءة أعلى لأن المفسرات تتسبب فى بطء التنفيذ.

اللغة الوسيطة IL

اللغة الوسيطة Intermediate Language (IL) عبارة عن لغة آلة قامت مايكروسوفت بإنشائها وتحتوى على مجموعة من التعليمات والأوامر التى لا تعتمد على

نوع المعالج المستخدم والتى يتم تحويلها إلى كود أصلى **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 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 الأصلى، 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 فى هذه الموارد داخل البرنامج أمرٌ بالغ التعقيد، لذا يتم التحكم فى هذه الموارد باتباع الخطوات الآتية:

١. يتم حجز جزء من الذاكرة بقدر النوع الذى يمثل المورد.
٢. يتم استهلال 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 يستخدمه المورد.

إذا لم يتم تحرير الذاكرة المستخدمة من قِبَل المورد أو تمت محاولة الوصول إلى المورد بعد تحرير ذاكرته، فى كلتا الحالتين يصبح البرنامج غير ثابت ويتصرف تصرفات غريبة. وهنا يأتى دور مجمع النفايات **Garbage Collector** الذى يعد أحد العناصر الأساسية داخل **CLR** والذى يقوم نيابةً عن المبرمج بالتحكم فى تحرير الذاكرة والتعامل مع الموارد. ولكن لا يستطيع مجمع النفايات التعرف على وقت تنفيذ الخطوة الرابعة فى الخطوات السابقة. فعلى سبيل المثال إذا أراد المبرمج إنهاء الاتصال بالشبكة، فأنَّى 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 فى إنهاء استخدام المورد.