البرمجة في بيئة

.NET

تحت المجهر

.NET programming under the hood

م. محمد العليان

@mhdalyan

”اذا لم تزد على الحياة شيئاً تكن أنت زائداً عليها”

مصطفى صادق الرافعي

إهداء

إلى عائلتي وأصدقائي في كل بقاع الأرض

إلى من شغلت قلبي وفكري وروحي ولم أجدها بعد!، ليلى

شكر وعرفان

لا يشكر الله من لا يشكر الناس، كل الشكر والتقدير إلى كل من ساعدني في مراجعة هذا الكتاب، سواءً من الناحية التقنية أو من الناحية اللغوية والصياغة.

الشكر الجزيل للمهندسة والمصممة مي مصيجة لتحملها عناء تصميم غلاف الكتاب والفصول.

الشكر لأهلي ولأصدقائي بلا استثناء، وقبل ذلك الحمد لله أولاً وأخيراً على توفيقه وإعانته لي لإتمام هذا الكتاب.

عن الكاتب

م. محمد العليان، إجازة في الهندسة المعلوماتية اختصاص هندسة شبكات ونظم تشغيل، جامعة دمشق 2013.

مُهتم بتطوير وتشريح التطبيقات والأنظمة البرمجية المُعقدة.

للتواصل مع الكاتب

 about.me

 LinkedIn

 Twitter

 mhdalyan@yahoo.com

كُتيبات سابقة

مقدمة

يقدم هذا الكتاب كم هائل من المعلومات التقنية والمفاهيمية والخبرات الأكاديمية، والتي لا تتبع لحقبة زمنية معينة ولا يزول تأثير هذه المعلومات بالتقادم، تم التركيز على مواضيع مهمة وعميقة في البرمجة للمحترفين وذلك ضمن إطار عمل NET..

المفاهيم والتقنيات المطروحة في الكتاب ليست حكراً على إطار عمل NET. ولغة C#، و إنما هي مستخدمة بشكل أساسي في لغات برمجة أخرى مثل java و C#، وهذا ما يميز محتوى الكتاب ويجعله قابل للفهم حتى للأشخاص الذين لم يبرمجوا بلغة C# من قبل.

إنَّ الفكرة العامة موجودة في جميع لغات البرمجة، وتبقى التفاصيل خاصة بكل لغة برمجة بذاتها.

ما يُهمنا في هذا الكتاب هو إيصال الفكرة بتجرد تام و بصرف النظر عن لغة البرمجة أو إطار العمل المستخدم، وما لغة C# إلا أداة لتمثيل هذه الأفكار.

سأكون مسروراً حقاً بملاحظاتكم على هذا الكتاب، وأرجو ألا تبخلو بها.

آمل من الله تعالى أن يكون هذا الكتاب مفيداً لكم وأن يقدم العون إلى كل من يريد أن يتعلم المفاهيم و الأدوات التي تصل به من مبرمج متوسط إلى مبرمج خبير ومحترف، و أن يكون عملي هذا في صحيفة أعمالي والله من وراء القصد.

دمشق في 12-9-2014

م. محمد العليان

لمن هذا الكتاب

هذا الكتاب موجه للأشخاص ذوي المعرفة المسبقة بلغة C++،Java،C# بشكل متوسط أو متقدم، حيث أنه يغطي مواضيع في المستوى المتقدم للمبرمجين ويركزعلى قضايا عميقة مثل تعدد النياسب والتعامل مع XML وقواعد البيانات مروراً بإدارة الذاكرة ونشر التطبيقات وغيرها من المواضيع المهمة.

محتوى الكتاب

يقدم هذا الكتاب مواضيع عامة ومتفرقة لذلك يمكن البدء بأي فصل، ولكن نوصي بقراءة الفصول الثلاثة الأولى لأهميتها في فهم كيفية عمل إطار عمل NET. واللغات التي تعمل تحت هذا الإطار.

تبدأ الفصول من 1 حتى 3 بوصف عالم NET.، حيث يقدم الفصل الأول "المجمعات" منصة NET. ما هي؟ وما هي العناصر الأساسية في بناء مكتبات إطار عمل NET.. كما يقدم تفاصيل عن كيفية بناء المجمعة وتشريح داخلي لعناصرها، ويشرح كيفية إنشاء وتوقيع المجمعات المشتركة وتسجيلها في ذاكرة المجمعات العامة.

يشرح الفصل الثاني موضوع "الصفات" و هو متمم للفصل الأول، حيث يدور حول البيانات الوصفية وكيفية تخزينها ضمن المجمعة، وكيفية البحث عن الصفات ضمن المجمعة واستخدام الصفات التي يوفرها إطار عمل NET. لتحقيق بعض الأهداف. وفي النهاية يتم الحديث عن كيفية كتابة الصفات المخصصة.

يغطي الفصل الثالث موضوع "الإنعكاس" وهو تقنية مستخدمة للتعرف على الأنماط في زمن التنفيذ وهذه الفكرة مُحققة في معظم لغات البرمجة مثل java،C++. يدور الفصل حول البيانات الوصفية (metadata) وطرق استرجاعها. كما يتم الحديث عن كيفية تشريح المجمعة ومعرفة كافة الأنماط المُعرفة ضمنها.

يُكمِل الفصل الرابع "COM and win api" الحديث عن التوافقية بين التطبيقات المكتوبة باستخدام لغة تندرج تحت إطار عمل NET. والتطبيقات والمكونات القديمة المكتوبة باستخدام المعيار COM. كما يتناول كيفية استدعاء تابع من مكون مكتوب بشيفرة غير مُدارة (مثل c أو c++) من خلال شيفرة مُدارة ضمن إطار عمل NET..

يقدم الفصل الخامس "التعامل مع xml" مقدمة شاملة تقريباً لكيفية التعامل مع xml، باستخدام الأدوات والمكتبات المعيارية. كما يقدم بعض الأمثلة على إختبار بنية ملف xml والتحقق من صحته، وتحويل بيانات بصيغة xml إلى صيغ أخرى مثل html.

يُغطي الفصل السادس "ADO.NET" كافة المفاهيم والأمور الأساسية والإضافية في تقنية ADO.NET وكيفية عملها، مع سرد تاريخي للتقنيات التي كانت مستخدمة سابقاً، كما يتناول مواضيع متقدمة في قواعد البيانات مثل التعامل مع المناقلات، والإجراءات المُخرنة وإنتهاءً بكتابة واستعمال القوادح.

يتناول الفصل السابع "نشرالتطبيقات" أحد المزايا التي يقدمها بيئة التطوير المتكاملة visual studio وإطار عمل NET. وهي تحزيم ملفات التطبيق وتحويلها إلى ملف تنصيب ليتم حمله إلى عدة حواسب وتنصيبه لنشر وتشغيل التطبيق. تُعرف هذه العملية بنشر التطبيقات (deployment).

يبحث الفصل الثامن "تعدد النياسب" في موضوع هام وعميق ومعقد، يستند إلى مفاهيم أكاديمية في أنظمة التشغيل والبرمجة التفرعية. يشرح الفصل الحاجة إلى تعدد النياسب وكيفية تحقيق ذلك، كما يستعرض كافة المشكلات التي واجهت مصممي أنظمة التشغيل وكيف تم حل هذه المشاكل على مستوى المهام processes، ليُترك حلها على مستوى النياسب على عاتق المبرمج. يستعرض الفصل أيضاً الأليات المستخدمة في أنظمة التشغيل لحل مشاكل التزامن والتشارك على الموارد وإداراة التضاربات والمطبقة في نظام التشغيل على مستوى المهام، ليقوم المبرمج بتطبيقها على مستوى النياسب أيضاً.

يُبين الفصل التاسع والأخير "إدارة الذاكرة والتعامل مع المؤشرات" تفاصيل دقيقة وعميقية في كيفية الحجز الذاكري للمتحولات والأغراض وأين يتم تخزين كل منهما، كما يتم الحديث عن جامع النفايات ودوره الهام في تنظيف منطقة الكومة الخاصة بالتطبيق الذي يعمل (المهمة). كما يتم الحديث عن كيفية تحرير الموارد الغير مُدارة من قبل محرك زمن التنفيذ المشترك (CLR) مثل الملفات المفتوحة والإتصالات الشبكية واتصالات قواعد البيانات. ويختم الفصل بالحديث عن أهمية المؤشرات وكيفية استخدامها ضمن شيفرة مُدارة ضمن إطار عمل NET..

تحميل الأكواد البرمجية

الشيفرة المصدرية للأمثلة الموجودة ضمن الكتاب متوفرة على منصة sourceforge، يمكن تحميلها من خلال الرابط التالي:

https://sourceforge.net/projects/netprogrammingunderthehood/

جدول المحتويات

الفصل الأول

مقدمة إلى المُجَمِعات

Assemblies

مقدمة

يُعتبر موضوع المجمعات (Assemblies) من المواضيع المتقدمة في .NET Framework. والتي على أساسها تم بناء مكتبات .NET Framework. كافةً، لذلك من الضروري فهم المجمعات بدايةً لفهم آلية تنفيذ البرنامج في .NET.، وكيف يعمل مترجم C#, VB.NET,F#.

سنرى أن جميع مترجمات لغات .NET. تولد نفس الشيفرة الوسيطة MSIL[1]، و ما هي بنية الملف التنفيذي EXE الذي يولده لنا مترجم .NET.، ثم سنتكلم عن بنية ملف DLL وما هي أوجه التشابه بينه وبين الملف الـ EXE، ثم سنتعرف على ماهية المجمعة وبنيتها الداخلية بشيء من التفصيل.

مقدمة إلى المجمعات

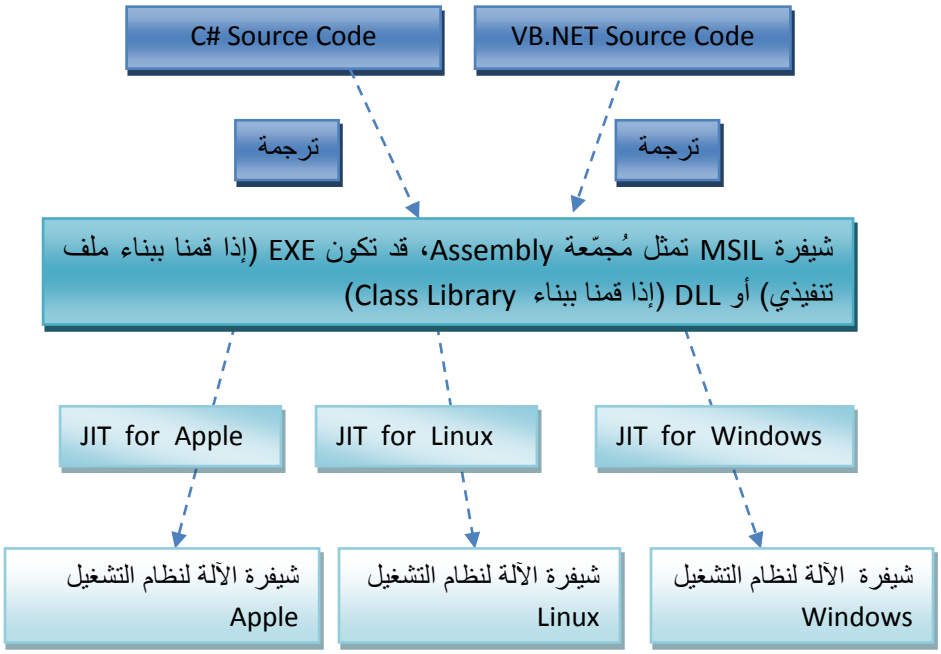
تُعرّف المجمعة على أنها برنامج .NET. تنفيذي أو جزء من برنامج تنفيذي على هيئة وحدة واحدة، تستخدم المجمعات لتحزيم البرامج المكتوبة بلغة C# أو VB.NET أو أي لغة .NET. جديدة لتصبح قابلة للتنفيذ و النقل من لغة إلى أخرى تحت منصة ال .NET.، مثلاً: يمكن لمبرمج أن يقوم بعمله باستخدام C# ثم يحوله إلى مجمعة DLL، ليستخدمها مبرمج أخر يعمل على VB.NET أو C++.NET وبذلك لن نحتاج إلى إعادة تنفيذ الكود في كل مرة نحتاجه.

عند بناء برنامج بلغة .NET. سيتولد ملف EXE يمثل مجمعة، لا نقصد أنه موجه لنظام التشغيل Windows وحسب بل هو برنامج محمول Portable ليعمل على أنظمة مختلفة أي أنه لا يحوي شيفرة محلية خاصة ب windows، بل شيفرة MSIL ومعلومات أخرى، لذلك عند تشغيل برنامج .NET.على نظام Windows لا يحوي مكتبات ال .NET. فأنه سيفشل في تنفيذ التطبيق مع أن نظام Windows يُشغل ملفات تنفيذية!. وعند بناء مكتبة أصناف Class Library سيتولد ملف DLL يمثل مجمعة أيضاً، وبالتالي المجمعة هي ملف EXE أو DLL.

الصورة التالية توضح كيفية تنفيذ تطبيق مكتوب بلغة .NET[2].

[1] للمزيد يمكن الإطلاع على الرابط التالي : http://msdn.microsoft.com/en-us/library/c5tkafs1(vs.71).aspx

[2] تطبيق مكتوب بلغة .NET. أو تطبيق مكتوب بلغة تحت منصة .NET. لهما نفس المعنى وقد يتم استعمال إحدى العبارتين لاحقاً.



لقد صُممت المجمعات بطريقة تمكننا من تقديم شكل آخر للبرامج تسمى بالمكونات، إنَ فهم المكونات أساسي لفهم المجمعات بسبب التشابه الكبير بينهما.

المكونات Components

المكون هو عبارة عن برنامج فرعي أو جزء من برنامج، يحوي شيفرة تنفيذية وليس شيفرة مصدرية، مما يعني أن بإمكان البرامج الأخرى أن تستخدمه دون الحاجة إلى لإعادة ترجمة الشيفرة المصدرية ودون الحاجة لمعرفة الشيفرة المصدرية الخاصة بالمجمعة، مما يوفر نوعاً من الأمن.

مزايا المكونات

من أهم مزايا المكونات:

1- إعادة استخدام البرامج الفرعية في برامج عديدة ، مثلاً: في حال قمت ببناء مكتبة خاصة بك وأردت أن تعطيها لشخص آخر دون أن يعرف ما هي الخوارزميات المتبعة في كتابة الكود الخاص بك، يمكنك أن تعطيه ملف DLL بدلا من الشيفرة المصدرية للمكتبة.

2- المجمعة التي قمت ببنائها يمكنك أن تقوم ببيعها، ويمكنك أن توقعها بأسم فريد يُسمى Strong Name لتكون وحيدة على مستوى العالم ولحفظ حقوقك من السرقة أو الاستخدام غيرالمشروع.

نبذة تاريخية عن المُجمّعات

في هذه الفقرة ما يهمنا ليس هو التاريخ بحد ذاته، ما يهمنا هو المشاكل التي كانت تعاني منها الشركات في السابق بسبب صعوبة التعامل مع المكونات وسنقدم مثالاً عن COM، إن **C**omponent **O**bject **M**odel هو أول مكون حقيقي قدمته شركة Microsoft، مشاكل COM تتلخص في ما يلي:

1- صعوبة تعلم وبرمجة COM، علماً أن المصدر الرئيسي لمكونات COM هو لغة C++ بااستخدام مكتبة ATL.

2- تثبيت مكونات COM صعب جداً ويحتاج إلى معلومات موجودة في مسجل النظام، إزالة تثبيت المكون تحتاج إلى جهد كبير أيضاً.

3- مشكلة تحديث المكونات حيث لا تستطيع التطبيقات التمييز بين ملفات DLL لها نفس الإصدار وهذا يؤدي إلى إخفاق أحد البرامج عن العمل وهذه المشكلة كانت تعرف بجحيم DLL[1]، مع NET. تم تجاوز هذه المشاكل بتعريف مقياس جديد للمكونات في NET.

ذاتية الوصف Self Description

إنَ ما يميز مجمعات NET. عن غيرها هو ذاتية الوصف ونقصد بهذا أن جميع المعلومات المتعلقة بالمجمعة موجودة ضمنها، وهذا يعني أن البرنامج أو النظام الذي سيستخدم المجمعة ليس بحاجة للحصول على أي معلومات من خارج المجمعة من مسجل النظام أو من أي مكان آخر، لأن هذه المعلومات موجودة ضمن المجمعة.

سابقاً، عند بناء DLL في C++ WIN32، كل ملف DLL يأتي معه ملف LIB وهو فهرس للتوابع و الإجراءات الموجودة بالمكتبة ولكن مع NET. تم تجاوز هذه المشكلة.

تتعدى مسألة ذاتية الوصف أسماء المناهج (Methods) والكائنات وأنواع البارمترات، فالمجمعات تتضمن معلومات عن إصدارات الكائنات كأن نقول مثلاً:Shapes1.0 يليها Shapes 1.1 ثم Shapes 2.0، وهذا يجعل عملية تثبيت مكونات NET. أسهل بكثير من السابق، حتى أن عملية تحديث المكونات ليست إلا مجرد نسخ ملفات DLL إلى القرص الصلب فقط!.

[1] للمزيد يمكن الإطلاع على المقال التالي http://www.c-sharpcorner.com/Blogs/7432/dll-hell-problem-and-solution.aspx

مجمعات .NET. ومكتبة أصناف .NET Framework.

إنَ أي برنامج مكتوب ب VB،C# مثلاً لا بد أن يستخدم مكتبة من مكتبات ال .NET.، فكلما استدعينا Method من فضاء الأسماء system فأن محرك زمن التنفيذ المشترك (CLR) سيضمن تحميل المجمعة ومعرفة التابع الذي قمت باستدعائه من المكتبة التي تم تحميلها، في أي namespace وفي أي Class هذا ال method، وذلك من خلال البيانات الوصفية MetaData، ومن خلال البيانات الوصفية يمكنه معرفة المجمعات التي قمت باستخدامها، مثلاً: لنفرض أنك قمت ببناء مجمعة واستخدمت المجمعة System.Drawing.dll عندئذٍ سيقوم ال CLR بقراءة البيانات الوصفية الموجودة في برنامجك، والذي يمثل أيضاً مجمعة، لمعرفة فيما إذا كان التطبيق يحتاج إلى مجمعات أخرى وبالتالي يمكن لتطبيق بسيط جداً يستحدم تعليمة using واحدة أن يستخدم عدة مجمعات مختلفة.

ملاحظة:

إنَ تعليمة using لا تقوم بإضافة مرجع للمجمعة وإنما تؤشر إلى فضاء أسماء (namespace) موجود ضمن المجمعة.

لإضافة مرجع إلى مجمعة من قائمة Project نختار Add reference ثم نحدد المجمعة التي نريدها

التداخل بين لغات البرمجة

قد يخطر لبعضنا السؤال التالي :

لماذا تقوم Microsoft بإنشاء لغات برمجة جديدة مع أنها جميعاً تترجم إلى Managed Code[1]؟

الجواب هو كالتالي :

إنَّ العديد من المبرمجين حساسين جداً للغة التي يبرمجون بها فكل منهم يُحب syntax خاص للغة معتاد عليه، ولا ننسى الدور التسويقي لجذب جميع المبرمجين للعمل على .NET. حيث يمكن لمبرمجي C++، C# ، VB العمل سوياً، كما يوجد محاولات لتحقيق التوافقية بين ال .NET. وال Java وبالتالي يمكن إنتاج منتج بواسطة فريق عمل من المبرمجين يستخدمون لغات برمجة مختلفة، وبالتالي نكون قد أنتجنا المنتج بواسطة عدة لغات برمجة.

[1] الشيفرة المُدارة هي أي شيفرة(كود) مكتوبة بإحدى لغات البرمجة التي تعمل تحت منصة .NET Framework.
للمزيد يمكن الإطلاع على الرابط التالي: http://msdn.microsoft.com/en-us/library/windows/desktop/bb318664(v=vs.85).aspx

المزايا الأساسية التي يوفرها .NET Framework.

سنستعرض بعض المزايا الأساسية التي يوفرها NET Framework. التي تمكننا من التداخل بين لغات البرمجة:

1- محرك زمن التنفيذ المشترك (**C**ommon **L**anguage **R**untime) والذي يدير تنفيذ جميع مجمعات NET. بغض النظر عن اللغة التي كتبت بها (CLR مشابهة ل JVM في Java).

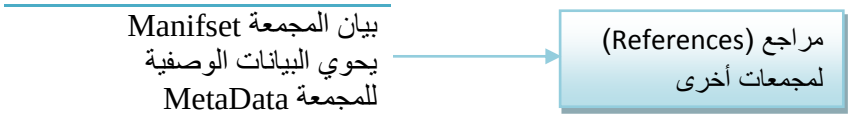
2- لغة MSIL (لغة مايكروسوفت المتوسطة) والتي تنتجها جميع مترجمات لغات NET.

3- مواضفات اللغة المشتركة (**C**ommon **L**anguage **S**pecification)، حيث يمكن للمكونات المكتوبة بلغة C# مثلاً أن تُستخدم في VB.NET مع إمكانية الوراثة الكاملة بين الأصناف.

بنية المجمعات

تحوي المجمعة على شيفرة تنفيذية للبرنامج أو مكتبة الأصناف التي أُنشئت منها بالإضافة إلى بيانات وصفية "MetaData"، إنَ البيانات الوصفية هي بيانات عن المجمعة بحد ذاتها وليست جزءاً من المجمعة.

يكون لمجمعة NET. الشكل التالي:



تحوي كل مجمعة على بيان (manifest) يصف محتوى المجمعة يسمى أيضا (Assembly Metadata) باعتبار أن هذا البيان يصف المجمعة ومحتواها، وما هي الوحدات التي تحويها وما هي المجمعات التي تستخدمها ومعلومات إضافية، سنتحدث عن بيان المجمعة لاحقاً والجدير بالذكر أن مكونات COM ليس لها بيان كهذا مبيت فيها.

يستخدم CLR هذ البيان في مجمعة البرنامج عند تنفيذه، وذلك لاستعادة المراجع للمجمعات الأخرى كما هو مكتوب في بيان المجمعة كاستعادة المكتبة System.Console التي تحتوي المنهج ()writeline في التطبيقات ذات الطبيعة Console مثلاً.

يتبع بيان المجمعة ما يسمى بالبيانات الوصفية للأنواع، تمثل هذه البيانات وصفاً للأصناف والمناهج والخصائص وكافة الأعضاء الموجودة داخل المجمعة بالإضافة إلى بارمترات المناهج وأنواعها و بعض المعلومات الإضافية، يتبع ذلك الشيفرة التنفيذية، وأخيراً هناك قسم المصادر حيث تمثل المصادر الأجزاء غير التنفيذية مثل الصور والأيقونات وملفات الرسائل.

يمكن فصل المجمعة إلى عدة ملفات ولكن عادة تكون ملف وحيد.

إنشاء المجمعات

1- ننشئ مشروع جديد من النوع Class Library، ثم نقوم بإضافة ملف يمثل صنف (class) فارغ ضمن namespace.

2- يمكن أن تحوي المجمعة أكثر من namespace، مثلاً: System.Data تحوي أكثر من namespace مثل System.Data.OLeDB و System.Data.SqlClient وفي كل namespace يمكن أن نضع Classes و delegates و Enum ، وأية بنى معطيات أخرى.

نقوم ببناء المشروع ، نضغط Shift+f6 فينتج لدينا ملف dll يمثل المجمعة يمكن أن نعطيه لأي شخص ليقوم بعمل Add Reference له ليستعمله في أي لغة تحت منصة .NET.

استعراض محتويات المجمعة

يمكن إستعراض محتويات المجمعة من خلال أداة تسمى ILDASM، تسمح هذه الأداة باستعراض محتويات المجمعة بما فيها شيفرة MSIL و البيانات الوصفية الخاصة بالمجمعة. خلال تطوير التطبيقات غالباً لن نحتاج لاستخدام هذه الأداة.

الأداة موجودة في المسار التالي:

```
C:\programfiles\microsoft.net\frameworkSDK\bin
```

ويمكن تشغيلها من موجه الأوامر (Commad line) الخاص بالvisual studio حصراً، لأن المتحول البيئي Path قد تم إعداده بشكل تلقائي عند تشغيله، لذلك نكتب:

```
>ildasm
```

بيان المجمعة Mainfest

يصف البيان معلومات عن المجمعة، والأهم من ذلك أنه يعرف أي مجمعات خارجية تستخدمها هذه المجمعة، يحوي البيان أيضاً رقم إصدارة المجمعة مما يجعل عملية تحديث التطبيق أمراً أسهل من ذي قبل.

صفات المجمعة

بالإضافة إلى المراجع (References) التي يحويها بيان المجمعة فإنه يحوي أيضاً معلومات تخص المجمعة نفسها أيضاً، تسمى هذه المعلومات بصفات المجمعة.

الملف Assemblyinfo.cs

قد يتساءل أحدنا ما هي وظيفة الملف Assemblyinfo.cs الذي يتولد مع كل تطبيق يتم إنشاءه بواسطة Visual Studio؟

يستخدم هذا الملف لضبط صفات المجمعة في بيانها تتضمن هذه الصفات اسم المجمعة وإصداراها ومعلومات إضافية سنتناولها لاحقاً، لنلق نظرة على هذا الملف.

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Training")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Training")]
[assembly: AssemblyCopyright("Copyright ©  2010")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("4baac969-4d1b-45c0-a680-b1fd6477cf66")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]

[assembly: AssemblyFileVersion("1.0.0.0")]
```

كل تعليمة موضوعة ضمن الأقواس المربعة تمثل صفة Attribute، يكفي أن نعلم أن كل تعليمة من هذه التعليمات تقوم بإعداد خاصية من خصائص المجمعة و تشير الكلمة assembly في بداية كل تعليمة أن الصفة موجهة للمجمعة نفسها وليس لأي صف class أو منهج method.

يوفر visual studio هذا الملف كقالب جاهز، يمكن ملأه بالقيم التي تريدها لمجمعتك ويمكن استعراض هذه الصفات من خلال الأداة ildasm من خلال القسم manifest (بيان المجمعة).

أرقام الإصدارات

كنا قد تكلمنا سابقاً أن لكل مجمعة اسم و رقم إصدار ومعلومات أخرى وهي كلها متوفرة على شكل Attributes، هذه المعلومات لا تهمنا نحن كمطورين للبرامج وإنما في حال إصدار منتج وقمنا بترقيه وتحديث مجمعاتنا سواء كانت dll أو EXE فأننا نقوم بتوقيع المجمعة بأسم فريد لضمان حقوق شركتك من السرقة وللسماح للبرامج من التأكد من أن هذه المجمعة موقعة.

للإصدار في مجمعة NET. أربعة أرقام هي:

1- إصدار رئيسي (Major Version)

2- إصدار ثانوي (Minor Version)

3- رقم البناء (Build Number)

4- رقم الطبعة (Revision)

مثلاً: المجمعة Shape 1.0.0.0 الإصدار الرئيسي هو 1 والإصدار الثانوي هو 0، الجزءان الأخيران يشكلان تفصيلاً أكبر، رقم البناء يشير إلى رقم بناء المجمعة وهو رقم فريد يتغير مع كل عملية بناء للمجمعة، أما رقم الطبعة فهو رقم يستخدم مع ترقيعات المجمعة وهذا يعني أن المجمعة هي تماماً كإصداراتها السابقة إضافة إلى تصحيح لخطأ واحد فقط.

التوافقية بين الإصدارات

يتفحص ال CLR أرقام الإصدارات عند تحميل المجمعات ويتحقق من التوافقية بين إصداراتها، ولا تحدث عملية التحقق هذه إلا مع المجمعات المشتركة، وهي المجمعات التي يتشارك عليها أكثر من تطبيق في آن واحد.

بالعودة إلى بيان المجمعة، رأينا أنه يحوي رقم إصدار المجمعة بالإضافة إلى مراجع للمجمعات الخارجية التي تستخدمها، وعند تنفيذ هذه المجمعة و لنفرض أنها ملف EXE يمثل تطبيق يستخدم مجمعات من نوع ملفاتdll، عندئذٍ سيقوم CLR بمقارنة رقم إصدارة المجمعة الخارجية (التي يستعملها ملف ال EXE والذي يمثل التطبيق) الموجود في بيان المجمعة (مجمعة تطبيقنا) مع رقم إصدارة المجمعة الخارجية (ملف الDLL) في بيانها وذللك للتأكد من التوافقية بين الإصدارات.

إذا كان للمجمعة رقم إصدار رئيسي أو ثانوي مختلف، عندئذٍ يوجد عدم توافق وبالتالي لنُتحمّل المجمعة. مثلاً: إذا كان التطبيق يستخدم مكتبة الأصناف Shape1.1 فأنه لن يعمل مع مكتبة الأصناف Shape1.0 أو Shape1.2.

المجمعات الخاصة والمشتركة

المجمعات الخاصة

الوضع الافتراضي للمجمعة هو أن تكون خاصة، وهذا يعني أنها خاصة بتطبيقك فقط ويجب على البرامج التي تستخدمها أن تضع هذه المجمعة في نفس مجلد التطبيق أو في المجلدات الفرعية.

المجمعات المشتركة

هي مجمعات متوفرة للاستخدام من قبل جميع البرامج في النظام، البرنامج ليس بحاجة لمعرفة مكان المجمعة لأن جميع المجمعات مخزنة في مجلد خاص يسمى ذاكرة المجمعات العامة (**G**lobal **A**ssembly **C**ache) أو إختصاراً GAC وهي موجودة في المسار التالي C:\Windows\assembly ، لأن هذه المجمعات متوفرة على مستوى النظام ككل فأن CLR يقوم بعدد من عمليات التفحص على المجمعات المشتركة للتأكد من إمكانية استخدامها من قبل البرامج التي تطلب ذلك .

الأمن والاسم المركّز Security and Strong name

يجب أن توفر المجمعة إثباتاً أنها لم تستبدل بمجمعة أخرى لها نفس الاسم والإصدار أو قد تم تعديلها بطريقة ما عن طريق الفيروسات، مثلاً: يتم ذلك عبر التحقق من أن المجمعة المشتركة موقعة بمفتاح مشفر قبل تحميلها في GAC، لا يساعد هذا المفتاح على حماية المجمعة من الطفيليات كالفيروسات وحسب، وإنما يمنع تضارب المجمعات الناتج عن تشابه أسماء وأرقام إصداراتها، ويسمى هذا التركيب المؤلف من اسم المجمعة وإصداراتها ومفتاحها (العام والخاص) بالاسم المُركزّ (Strong name).

إنشاء المجمعات المشتركة

لكي نتمكن من إنشاء مجمعة مشتركة باسم مُركزّ، علينا توليد زوج من المفاتيح العام والخاص وذلك للتعرف على المجمعة من خلاله عند استخدامها. تَستخدم أنظمة المفاتيح المشفرة المفاتيح العامة والخاصة مفتاحاً خاصاً معروفاً من قبل مرسل الرسالة المشفرة فقط، أما المفتاح العام فهو متوفر للعالم ككل، يستخدم CLR هذه المفاتيح للتأكد من تطابقها (المفتاح الموجود في بيان المجمعة (التطبيق)) يجب أن يطابق المفتاح الموجود في بيان المجمعة المشتركة التي يستخدمها التطبيق.

يمكن توليد اسم مركز عن طريق سطر الأوامر الخاص بـ Visual Studio:

```
C:\Sn -k  keyfile.snk
```

سيؤدي هذا الأمر لإنشاء ملف المفتاح keyfile.snk ولتسجيل (توقيع) المجمعة بهذا المفتاح فأننا سنعدل الصفة(AssemblyKeyfile (Attribute في أخر جزء من الملف Assemblyinfo.cs للمشروع كما يلي:

```
[assembly:AssemblyKeyFile("keyfile.snk ")] // يمكننا إضافة مسار المفتاح أيضاً
```

عادةً يكون ملف المفتاح موجود في المجلد debug أو release.

ملاحظة: يمكننا أن نضع التعليمة السابقة في الملف المصدري للمجمعة في حال عدم وجود ملف Assemblyinfo.cs كما يلي:

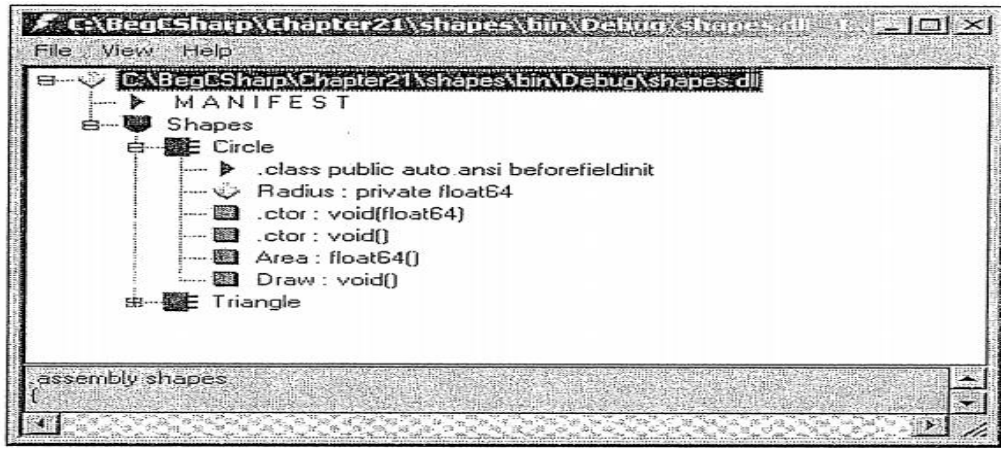
```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly:AssemblyKeyFile("keyfile.snk ")] // يمكننا إضافة مسار المفتاح أيضاً
```

الآن نعيد بناء المجمعة ،فنحصل على مجمعة موقعة، وإذا تفحصنا بيان المجمعة سنجد المفتاح العام الذي تم توليده ووضعه الآن سنقوم بنسخ ملف ال dll الذي نتج لدينا إلى GAC وبذلك نحصل على المطلوب.

ملاحظة: أحياناً نشغل برنامج مثلاً يعتمد على مكتبات (مجمعات) عندها يقوم CLR بالبحث عن المجمعة في المجلد المحلي للبرنامج أولاً فإذا لم يجدها فأنه يبحث عنها في GAC فإذا لم يجدها يقوم برمي استثناء Exception لمستوى نظام التشغيل[1].

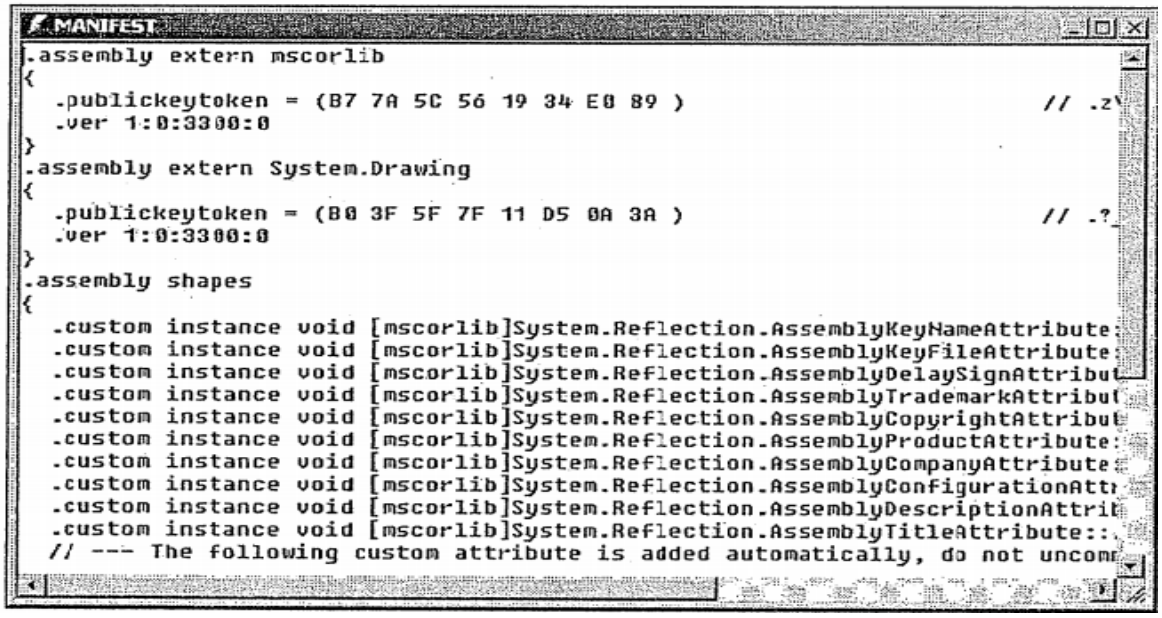
الأداة ildasm

يمكننا من خلال هذه الأداة أن نرى المفتاح العام الذي نتج لدينا عند توقيع المجمعة (وهو موجود في بيان المجمعة)، أو أن نرى المراجع إلى المجمعات الأخرى التي تستخدمها المجمعة الحالية، أو أن نشاهد الصفوف classes ضمن كل Namespace وكافة ال methods وال Data members داخل كل class مثلاً:



[1] للمزيد يمكن الإطلاع على الرابط التالي : http://www.c-sharpcorner.com/uploadfile/72d20e/concept-of-shared-assembly-in-net

سنفرض أن لدينا مجمعة باسم shape.dll تحوي صنفين هما circle، Tringle، نلاحظ وجود قسمين الأول خاص بالمجمعة بحد ذاتها، وقسم أخر مهم جداً هو ال Manifest (بيان المجمعة) ويحوي على البيانات الوصفية والتي هي فعلياً الـ Attributes التي تكلمنا عنها سابقاً، بالإضافة إلى المراجع (References) المستخدمة في هذه المجمعة، ويحوي البيان أيضاً أرقام الإصدارات والمفتاح العام أيضاً وأمور أخرى وكل هذا مكتوب بشيفرة MSIL.



الفصل الثاني

الصفات

Attributes

مقدمة

يُعتبر موضوع الصفات من المواضيع المتقدمة فيNET.، كما يرتبط إرتباطاً وثيقاً مع المجمعات والتي درسناها في الفصل السابق. في البداية سنتكلم عن الصفات وما هي طبيعتها وأين تستخدم ولماذا تستخدم، ومواضيع أخرى سيتم التعرف عليها، وما هي "البيانات الوصفية" MetaData[1] (فعلياً تمثل Attributes)، ثم سنتعرف على الصفات المُبيتة في NET..وأخيراً سنتناول إنشاء صفات مخصصة (Custom Attributes).

ما هي الصفة Attribute؟

مبدئياً، سنعرف الصفة على أنها "معلومات إضافية يمكن تطبيقها على كتلة من الشيفرة البرمجية ضمن المجمعة(DLL أو EXE)"، يمكن أن تمثل هذه الكتل Class أو Method أوData Member أو Property، يمكن الوصول إلى هذه المعلومات من أي صنف (Class) أخر يستخدم هذه المُجمعة.

تحدثنا في الفصل السابق عن المُجمّعات وتعرضنا للملف AssemblyInfo.cs، وهو متوفر كقالب جاهز مع أي مشروع ضمن بيئة التطوير المتكاملة Visual Studio. إذا استعرضننا محتويات الملف سنجد ما يلي.

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Training")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Training")]
[assembly: AssemblyCopyright("Copyright ©  2010")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

لقد عرضنا جزءاً من الملف فقط، وسنجد فيه عدداً من الأسطر تبدأ بعبارة assembly، هذه هي تعريفات الصفات، فعند ترجمة الملف سيتم حفظ أي صفة ضمن المجمعة وهذه العملية تسمى بالتخليل "Pickling"، ولرؤية ذلك سنقوم ببناء مجمعة (EXE أو DLL)، وسنقوم بتغيير الصفة AssemblyTitle ثم نترجم المجمعة.

```
[assembly: AssemblyTitle("Training")]
```

ثم من خصائص المجمعة نشاهد قيمة الصفة السابقة في الحقل Description، هذه الصفات التي يحويها الملف Assemblyinfo.cs تمثل فعلياً "البيانات الوصفية" MetaData. يمكن الحصول على هذه الصفات من خلال

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://msdn.microsoft.com/en-us/library/xcd8txaw(v=vs.110).aspx

خصائص ملف المجمعة، أو من خلال الأداة ildasm، أو برمجياً من خلال كتابة شيفرة باستخدام تقنية الانعكاس Reflection التي سيتم الحديث عنها في الفصل القادم.

من خلال الشكل الذي تُكتب فيه الصفة نلاحظ أن تصريح الصفة تمثل إنشاء غرض من صنف، والبارمترات التي نمررها هي بارامترات الباني (Constructor).

كل هذا يقودنا إلى التعريف التالي :

"الصفة عبارة عن صنف يمكن أن يحوي بيانات إضافية عن المجمعة، تتعلق هذه المعلومات بالمجمعة نفسها أو بأي نوع ضمنها".

لنعد إلى الشيفرة المصدرية:

```
[assembly: AssemblyTitle("Training")]
```

لقد قلنا أن الصفة هي صنف ولكن الصيغة هنا مختلفة تماماً عما نعرفه سابقاً في C#، يشير المعرف assembly إلى مدى الصفة وما تبقى من السطر يمثل تصريح الصفة، للصفة باني يقبل بارمتر من نوع string، يقوم المترجم بتضمين هذه القيمة ضمن المجمعة عند بنائها ويمكن الوصول إليها من خلال خصائص المجمعة في نظام Windows، أو من خلال الأداة ildasm، أو برمجياً بواسطة الانعكاس Reflection.

مقدمة إلى الانعكاس Reflection

يُعبر الانعكاس موضوعاً جديداً في عالم البرمجة لذا فإننا سنتكلم عنه قليلاً، يُمكّننا الانعكاس من تفحص المجمعات والحصول على معلومات عنها برمجياً ويتضمن ذلك كافة أنواع الكائنات المعرفة ضمنها.
يتوفر الانعكاس في .NET. ضمن فضاء الأسماء System. Reflection.

البحث عن الصفات

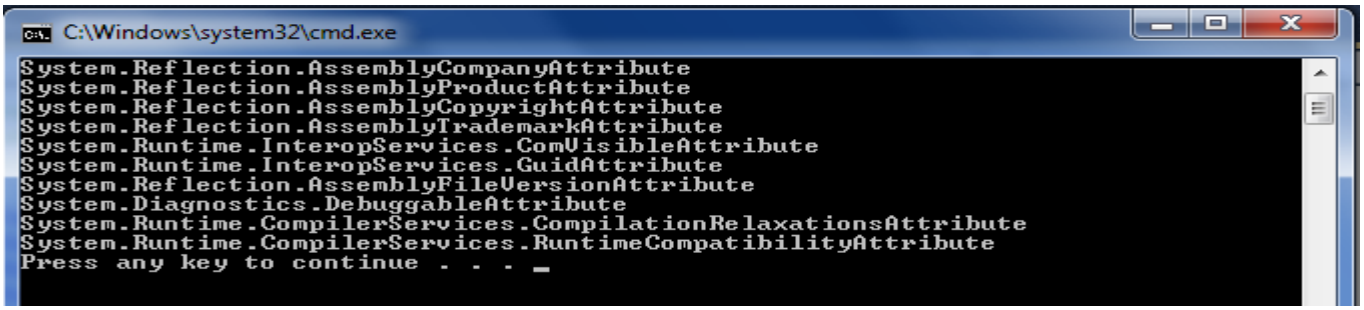
لنتأمل معاً الكود التالي:

```
using System;
using System.Reflection;

static void Main(string[] args)
{

    Assembly a = Assembly.LoadFrom(@"D:\atom.dll");
    object[] attributes = a.GetCustomAttributes(true);

    foreach(object o in attributes)
    {
```

```
        Console.WriteLine(o.ToString());
    }
}
```

- قمنا في البداية بتحميل مجمعة وذلك عن طريق الصنف Assembly وهو موجود في فضاء الأسماء System.Reflection، و جميع الصفات الموجودة في الملف Assemblyinfo.cs هي موجود أيضاً ضمن فضاء الأسماء سابق الذكر.

- ثم حصلنا على جميع الصفات المخصصة وهي موجودة سابقاً في الملف Assemblyinfo.cs وقمنا بعرضها على الشاشة، ويكون خرج البرنامج السابق هو:



الصفات المُبيتة في .NET

إن الصفات الموجودة في الملف Assemblyinfo.cs هي صفات خاصة بالمجمعة بحد ذاتها، أما الصفات التي سنعرضها فهي تخص الأنواع المعرفة ضمنها وأحياناً المجمعة بحد ذاتها، سنستعرض بعض الصفات التي يوفرها .NET Framework. و متى يمكننا استخدامها.

الصفة System.Diagnostics.ConditionalAttribute

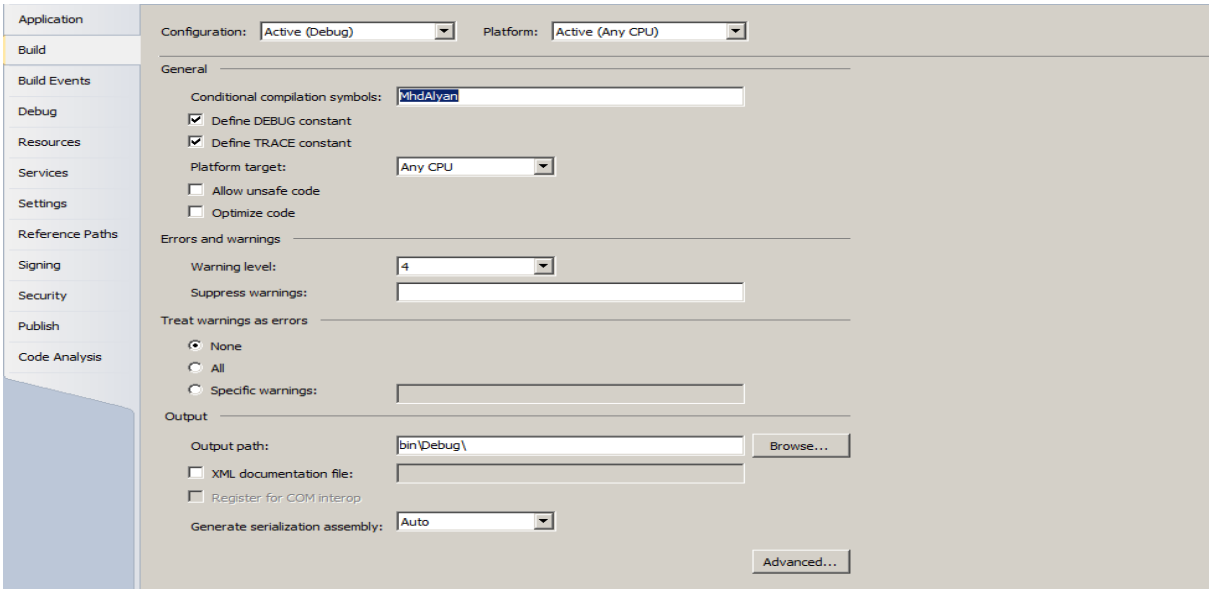
تلك إحدى أهم الصفات على الإطلاق، فهي تسمح بتضمين أو استثناء أجزاء من الشيفرة بالاعتماد على تعريف رمز أثناء الترجمة توجد هذه الصفة ضمن فضاء الأسماء System. Diagnostics والذي يتضمن عدداً من الأصناف (Classes) لتنقيح وتتبع الخرج وتسجيل الأحداث ومراقبة الأداء ومعالجة المعلومات، الآن لنتأمل الشيفرة التالية:

```
using System;
using System.Diagnostics;

[Conditional("MhdAlyan")]
public static void DebugOnly()
{
      Console.WriteLine("this String Only Display in Debug Builds");
}
```

```
static void Main(string[] args)
{
     DebugOnly();
}
```

تقوم هذه الشيفرة بوسم (Taged) المنهج DebugOnly بالصفة Conditional وتمرير رمز تم تعريفه مسبقاً ويتم ذلك وفق طريقتين:

1- في حال استخدام Visual Studio يمكن تعريف الرمز من خلال النافذة التالية علماً أن الرمزين DEBUG,TRACE هما رموز معرفة سابقاً للعمل في نمط التنقيح (DEBUG).



يتم تنفيذ التابع DebugOnly عند وجود رمز معرف سابقاً (MhdAlyan)، وفي حال عدم وجوده لا يتم تنفيذ محتوى هذا التابع. لنرى الخرج الآن في حال قمنا بتعريف الرمز السابق:

وفي حال تغيير الرمز كما يلي:

```
[Conditional("MOHAMMAD")]
```

لن يتم تنفيذ المنهج DebugOnly

2- يمكن تعريف رموز أيضاً من خلال سطر كما يلي:

```
csc /d: MhdAlyan program.cs
```

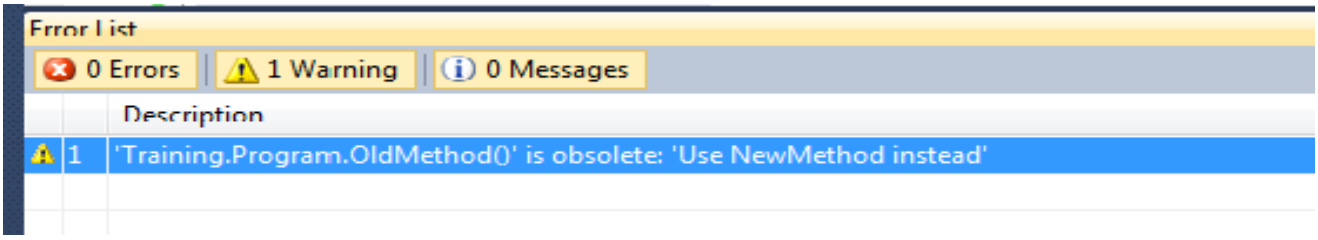
الصفة System.ObsoleteAttribute

- عند التعرف على هذه الصفة يتضح لنا مدى اعتناء مهندسي Microsoft بأدق التفاصيل، تستخدم هذه الصفة لوسم تابع بأنه لم يعد مستخدماً بعد الآن.

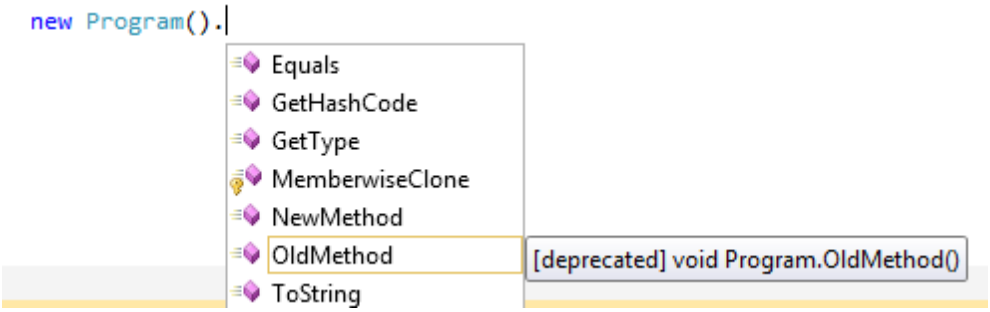
- يمكن أن تكون هذه الصفة مفيدة جداً، على سبيل المثال عندما تود نشر مكتبة من الأصناف ومن المحتم أنه خلال عملية التطوير سيكون بعض المناهج التي لن يكون لها وجود في الإصدارة النهائية من المكتبة، وبذلك يمكن تحضير مستخدمي مكتبتك لغياب ميزة محددة، لنرى المثال التالي:

```
[Obsolete("Use NewMethod instead")]
public void OldMethod()
{
}

public void NewMethod()
{
}
```

عند ترجمة المشروع سنحصل على رسالة تحذير بأن هذا المنهج مهجور "deprecated" أو سُيهجر قريباً وعليك استخدام NewMethod بدلاً منه.



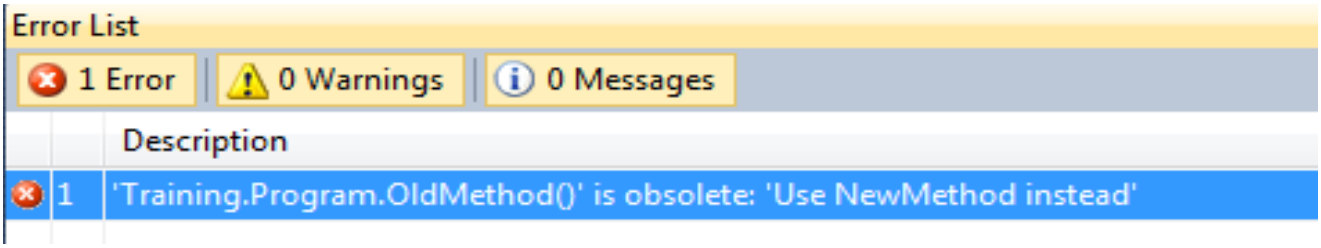
مع مرور الوقت فأن جميع مستخدمين هذا المنهج سيعلمون أن عليهم تجنب استخدام هذا المنهج وحتى عند استدعائه يعطيك أن هذا المنهج deprecated.



مع الوقت يمكنك أن تزيل المنهج OldMethod من شيفرتك تماماً دون خوف، لذلك يمكنك إضافة بارمتر جديد للصفة Obsolete كما يلي:

```
[Obsolete("Use NewMethod instead",true)]
public void OldMethod()
{
}
```

وعندما يحاول المستخدمون استخدام هذا المنهج فأن المترجم سيصدر خطأ ترجمة، وتتوقف عملية الترجمة بالرسالة التالية:



الصفة System.SerializableAttribute

السّلسَلة (Serialization)، هي الاسم الذي يطلق على ترتيب واستعادة الكائنات من الذاكرة أو من الأقراص بصورتها الثنائية، عندما نقوم بسّلسَلة كائن فإن جميع بيانات الكائن تُحفظ ضمن وسط التخزين، وعند إزالة السّلسَلة (Deserialize) سيتم إعادة الكائن من وسط التخزين إلى حالته الأصلية.

ما الهدف من كل الصفات التي عرضناها والتي سنعرضها بعد قليل؟

الهدف هو تخزين معلومات إضافية ضمن المجمعة على شكل بيانات وصفية، هذه الصفات هي التي تُخبر CLR ليقوم بعمل ما وفق ما تُعبر عنه الصفة، فمثلاً: عندما يستقبل CLR طلباً لسّلسَلة كائن فأنه سيتفحص فيما إذا كان الصف يدعم الواجهة Iserializable أم لا، إن لم يكن يحقق الواجهة الأخيرة سيتفحص فيما إذا كان الصنف موسوماً بالصفة Serializable، وعند وجود الصفة السابقة في الصنف سيستخدم .NET الانعكاس للحصول على بيانات أعضاء الصنف سواء كانت عامة أو خاصة أو محمية وسيحفظ هذه البيانات كتمثيل للكائن، أما إزالة السلسلة فهي عكس السلسلة تماماً.

الآن سنرى كيف تتم عملية السلسة Serialization.

```
using System;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]

class Employee
{
    public Employee(int id ,string name)
    {
        _id = id;
        _name = name;
    }

    private int _id;

    private string _name;

    [NonSerialized]
    //this Data Member is not Serialized,Such asTransiet in JAVA
    private string _password;

    public int Id
    {
        get { return _id; }
        set { _id = value; }
    }

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }

}
```

سنقوم الآن بسلّسلة كائن Employee ضمن ملف ثنائي.

```
static void Main(string[] args)
{
    Employee e = new Employee(1, "mohammad");

          FileStream fs = null;
           try
           {
               string path= @"D:\file";

               fs = new FileStream(path, FileMode.Create);

               BinaryFormatter bf = new BinaryFormatter();

                // Serialize Object(e) in the file stream(fs)
               bf.Serialize(fs, e);
             }
           catch (Exception Ex)
           {
              Console.WriteLine(Ex.Message);
           }
           finally
           {
               try
               {
                   fs.Close();
               }
               catch (Exception es)
               {
                  Console.WriteLine(es.Message);
               }

           }
}
```

أما إزالة السلسلة DeSerailization فتتم كما يلي:

```
static void Main(string[] args)
{
        Employee e1 = null;

     try
     {
        string path = @"D:\file";

        fs = new FileStream(path, FileMode.Open);

        BinaryFormatter bf = new BinaryFormatter();

        e1 = (Employee)bf.Deserialize(fs); // DownCasting is mandatory

        Console.WriteLine("Id={0} name={1}",e1.Id,e1.Name);

    }
    catch (Exception Ex)
    {
          Console.WriteLine(Ex.Message);
    }
    finally
    {
        try
          {
                    fs.Close();
          }
          catch (Exception ex) { Console.WriteLine(ex.Message); }
    }


}
```

الصفة System.Reflection.AssemblyDelaySignAttribute

- يوفر فضاء الأسماء System.Reflection عدداً من الصفات تعرفنا إلى بعضها سابقاً في هذا الفصل، إحدى أكثر هذه الصفات تعقيداً في فضاء الأسماء سابق الذكر هي الصفة AssemblyDelaySign.

- لقد تعلمنا في الفصل السابق كيفية بناء وإنشاء مجمعات مشتركة وتسجيلها في GAC، يسمح NET. بتأجيل توقيع المجمعات أيضاً، يعني ذلك أنه يمكننا تسجيل المجمعة في ذاكرة المجمعات العامة لإغراض الفحص والاختبارات ودون إعطائها رقماً خاصاً.

- أحد السيناريوهات التي يمكن استخدام التوقيع المؤجل فيها عندما نود تطوير التطبيقات التجارية، فكل مجمعة يتم تطويرها في المنزل يجب أن توقع بمفتاح خاص على جهاز المطور قبل أن تصل للزبائن.

- على أية حال، لا ترغب المؤسسات البرمجية بتوفير مفتاحها الخاص على جميع أجهزة المطورين لذلك يُمكننّا CLR من توقيع المجمعة بصورة جزئية ووضعها في GAC، دون الحاجة لمفتاح خاص وبعد اختبار المجمعة تماماً يمكننا توقيعها بشكل نهائي ووضعها في GAC من خلال الشخص الذي يملك المفتاح الخاص.

توليد ملف المفتاح

أولا نحن بحاجة لتوليد ملف المفتاح (العام والخاص)، لذلك سنستخدم الأداة sn.exe[1]

```
sn –k Company.key
```

الآن سنستخرج المفتاح العام منه لكي نستخدمه في توقيع المجمعة بشكل مؤقت.

```
Sn –p Company.key Company.public
```

سيقوم هذا الأمر بإنشاء ملف المفتاح العام فقط، هذا المفتاح معروف لدى الجميع لذلك يمكن نسخه إلى جميع أجهزة المطورين، أما الملف الذي يحوي المفتاح الخاص هو الذي يجب أن نحفظه في مكان آمن لأننا سنستخدمه عند توقيع المجمعة بصورة نهائية، لذلك احفظ الملف Company.key في مكان آمن.

[1] استخدام الأداة sn يتم باستخدام Visual Studio Command Prompt الموجود في مجلد Visual Studio Tools بقائمة ابدأ.

توقيع المجمعة بصورة جزئية

الكود التالي يشرح المطلوب:

```
using System.Reflection;
[assembly: AssemblyKeyFile("Company.public")]
[assembly: AssemblyDelaySign(true)]// True is means the Assembly is DelaySigned
```

الأن سنقوم بتسجيل المجمعة في GAC باستخدام الأمر التالي :

```
gacutil  /i program.exe
```

لكن سيعطي خطأ لأن GAC يرفض أي مجمعة لا تحوي اسم مركّز (strong name)، لذلك سنوقف GAC للتخلص من هذا التقييد باستخدام الأمر:

```
Sn -Vr *     // السماح لجميع المجمعات التي لا تحوي اسم مركز بالتسجيل ضمن GAC

Sn -Vr myAssembly.dll
```

إكمال الاسم المركّز (strong name)

المرحلة الأخيرة هي توقيع المجمعة بصورة نهائية، وذلك عن طريق ترجمة المفتاحين العام والخاص ضمن المجمعة وبالتالي توليد اسم مركز،عندها نستطيع تسجيل المجمعة ضمن GAC دون الحاجة لتخطي عملية التحقق من الأسماء المركّزة وهذه المرة سنستخدم الأداة sn مع الخيار R–.

```
//إعادة توقيع المجمعة وإضافة الجزء المتعلق بالمفتاح الخاص
sn -R program.exe  Company.key
```

الصفات المخصصة Custom Attributes

تكلمنا في النصف الأول من الفصل عن الصفات المُعرفة ضمن .NET، يمكننا أيضاً إنشاء صفاتنا الخاصة وهو ما نسميه بالصفات المخصصة[1]، سنتكلم عن كيفية إنشاء الصفات الخاصة.

تُمثل الصفة المخصصة صنف (Class) يجب أن يخضع للمواصفات التالية:

1- يجب أن يرث من الصف System. Attribute.

2- يمكن أن تتضمن منهج بنّاء (Constructor) الصفات المخصصة بارمترات، ويجب أن تكون أنواع البارمترات بسيطة (Primitive) كالأعداد أو السلاسل النصية أو القيم المنطقية.

الصفة TestCaseAttribute

عندما نقوم بإنشاء مكتبة أصناف ما فأننا ننشئ صفوف اختبار لاختبار مكتبة الأصناف التي لدينا، وتتأكد من أنها تؤدي ما هو مطلوب منها، تزداد الحاجة لهذه الأصناف عندما نقوم بإصلاح خطأ ما أو عندما نقوم بوظيفة جديدة وذلك لنتأكد أن كل شي يسير على ما يرام.

الصفة TestCaseAttribute

```
    /// <summary>
    ///  this is  a Custom Attribute
    /// </summary>
[AttributeUsage(AttributeTargets.Class,AllowMultiple=false,Inherited=true)]
        public class TestCaseAttribute:Attribute
        {
            public TestCaseAttribute(Type testcase)
            {
                _testcase = testcase;
            }
            public readonly Type _testcase;
            public void Test()
            {
                //Create an instance of Class underTest
                object o = Activator.CreateInstance(_testcase);
            }
        }
```

[1] للمزيد يمكن الإطلاع على الرابط التالي : http://msdn.microsoft.com/en-us/library/84c42s56(v=vs.110).aspx

الصنف المراد اختباره

لاختبار أي صنف نقوم بوسمه بالصفة TestCase.

```
/// <summary>
/// A Class that used TestCaseAttribute
/// </summary>
[TestCase(typeof(TestAnObject))] //(***)
public class SomeCodeOrder
 {
       public int Do()
         {
             return 999;
         }
 }
```

صنف الاختبار TestAnObject

```
/// <summary>
/// this is a Test Class, that it is test the Other Classes
/// </summary>
class TestAnObject
{
      public TestAnObject()
      {
          SomeCodeOrder s = new SomeCodeOrder();

          if (s.Do() != 999)
            throw new Exception("Error");
     }
}

static void Main(string[] args)
{

            Assembly a = Assembly.GetExecutingAssembly();

            Type[] arr = a.GetExportedTypes();

            foreach (Type t in arr)
            {

                Console.WriteLine("Checking Type {0}", t.ToString());

                object[] atts = t.GetCustomAttributes(typeof(TestCaseAttribute), false);
                if (atts.Length == 1)
                {
                    Console.WriteLine("found TestCaseAttribute : Running Test");
                    TestCaseAttribute tca = (TestCaseAttribute)atts[0];
```

```
                //Perform Testing
                try
                {
                    tca.Test();
                    Console.WriteLine("Passed");
                }
                catch (Exception e)
                {
                    Console.WriteLine("failed");

                    Console.WriteLine(e.Message);
                }
            }
        }

}
```

آلية التنفيذ

في هذا المثال قمنا بتحميل المجمعة الحالية (التطبيق المُنفذ)، ثم قمنا باستعراض كافة الأنماط ضمنه، من أجل كل نمط قمنا باستعراض الصفات المخصصة لكل نمط و فحص فيما إذا كان النمط موسوماً بالصفة TestCaseAttribute، بالتالي سيتم إختبار الصنف SomeCodeorder لأنه موسوم بالصفة TestCaseAttribute من خلال الصنف TestAnObject، يُمكننا من خلال هذا الصنف فحص جميع الصفوف الموسومة بالصفة TestCase وإنشاء غرض في زمن التنفيذ من الصنف الذي نريد اختباره عن طريق الصف Activator وتطبيق منهج Test.

سيناريو تنفيذ الكود

بعد أن نعرف أن الصنف SomeCodeOrder موسوم بالصفة TestCase (TestCaseAttribute)، وهو الصنف الذي سنقوم باختباره، سيؤدي ذلك لإستدعاء باني الصنف TestCase كما يلي:

```
[TestCase(typeof(TestAnObject))]
```

وسيمرر للباني نمط (Type) الصف المسؤول عن عملية إختبار الصف الموسوم بهذه الصفة (SomeCodeOreder)، وبالتالي سيمرر للباني نمط الصنف TestAnObject وسيقوم بإنشاء Object من الصنف السابق وذلك بعد استدعاء التابع Test الخاص بصفة الاختبار، وعند إنشاء object من الصف TestAnObject سيؤدي إلى استدعاء الباني الخاص بالصنف TestAnObject، وبذلك تتم عملية الإختبار حيث داخل باني هذا الصف ننشئ كائن من الصف المراد فحصه (SomeCodeOreder) ثم نتحقق من صحة الخرج الذي لدينا عن طريق المنهج Do.

قد لا نشعر بالفائدة الحقيقية من خلال هذا المثال، إضافةً إلى أن طريقة تنفيذه ليست بالأمر البسيط، ولكن عندما يكون لدينا صنف حقيقي يقوم بوظائف ذات معنى سنشعر بأهمية الصفات.

الصفة System.Attributeuseage

عند تعريفنا لصفة مخصصة فأنه من الضروري تحديد الأنواع التي يمكن وسمها بهذه الصفة، وبالعودة إلى المثال السابق فأننا قررنا أن الصفة TestCase لا تستخدم إلا مع الصفوف وذلك باستخدام التعداد AttributeTarget ونفصل بينهم بواسطة | ، بينما يمثل البارمترالثاني والثالث يمثلان خصائص وهي كما يلي :

الأولى هي Allowmultiple وتسمح باستخدام الصفة أكثر من مرة على العنصر نفسه، بينما تفيد الخاصية الثانية inherited بتوريث الصفات.

نقصد بالخاصية ما يلي:

لنفرض أننا نود في إضافة اسم الشخص الذي قام بالفحص وذلك ضمن الصفة TestCase لذلك سنقوم بإضافة الشيفرة التالية للصف TestCase:

```
private string author;
public string Author
{
    get { return author; }
    set { author = value; }
}
```

وعند وسم صنف ما باستخدام الصفة السابقة، يمكننا من خلال باني الصف أن نمرر قيم للمتحولات التي قمنا بعمل خاصية لها وذلك كما يلي:

```
/// <summary>
/// A Class that used TestCaseAttribute
/// </summary>
[TestCase(typeof(TestAnObject),|)]
c▲ 2 of 2 ▼ TestCaseAttribute.TestCaseAttribute(Type testcase, Named Parameters...)
{   Named Parameters:  Auther = string
                       _testcase = Type
        return 999;
    }
}
```

يمكننا إضافة تفاصيل إضافية، وذلك عن طريق الخاصية Author والمتحول العام testCase_(نلاحظ أنه أصبح لدينا نسختين من الباني مع أننا لم نقم بعمل Overloading).

نكتب اسم الخاصية ثم القيمة، مثلاً : Author=”mohammad”، ونفس الكلام يُطبق مع المتحول العام.

مدى الصفات

تعرفنا في القسم السابق من الفصل على الصفات التي تبدأ بالبادئة Assembly وقلنا أنها تعني أن هذه الصفة موجهة للمجمعة نفسها، ولكن إذا أردنا أن نضيف صفة لقيمة مُعادة من تابع فعلينا أن نخبر المترجم بمدى الصفة مثلاً في حال أردنا أن نطبق صفة معينة لقيمة مُعادة من تابع فعلينا أن نقوم بما يلي:

```
[return:myAttribute]
public int DoSomething()
{ }
```

في حال أردنا أن نطبق صفة على متحول ضمن صف فإننا سنكتب:

```
[field:myAttribute]
```

بنفس الطريقة نطبق الصفة على Event أو Property.

الفصل الثالث

الإنعكاس

Reflection

مقدمة

يُعتبر الانعكاس من المواضيع المتقدمة والهامة جداً في .NET، والذي يرتكز بشكل أساسي على موضوع المجمعات Assemblies والصفات Attributes، وقد تكلمنا عن كل منهما في الفصول السابقة. يُعرف الانعكاس Reflection ببساطة بأنه عملية استكشاف للمجمعة للحصول على محتوياتها بشكل كامل. يمكن أن نقوم باستكشاف DLL أو EXE باستخدام تقنية الانعكاس لنستعرض مخطط الصفوف والوراثة ومحتويات كل صنف، والأهم من ذلك أنه يمكننا استكشاف الصفات ضمن المجمعة يعتمد CLR على هذه المعلومات لأداء عمله.في البداية سنتكلم عن بعض المواضيع النظرية ثم سننتقل لمجموعة من التمارين والتي ستغطي كامل المواضيع النظرية التي سنتلكم عنها.

مقدمة فلسفية عن الانعكاس

في الماضي القديم كان القدماء يُعرّفون كلمة الانعكاس (Reflection) بأنها:

"شي ما يُعكس مثل الضوء، الإشعاع الحراري، الصوت أو الصورة، التركيز الفكري والاعتبار الحذر للأمور، تعبير غير مباشر عن انتقاد أو لوم، مظهر أو نتيجة"، كما في قولنا:
"إنجازاته هي انعكاس لشجاعته وجرأته".

قد نعتقد أن معنى الانعكاس في لغة C# لا يندرج تحت أي من المعاني السابقة، ولكن الحقيقة هي العكس تماماً وسنرى ذلك لاحقاً، إذ أن الانعكاس له علاقة وثيقة بعملية تشريح التطبيقات (التعرف على كامل محتوياتها في زمن التنفيذ).

ويُعرف الانعكاس في لغة C# بأنه العملية التي يتمكن بواسطتها CLR من اكتشاف المعلومات حول الصفوف والأنواع الأخرى المعرفة ضمن المجمعات (DLL,EXE).

تُقسم المعلومات في المجمعة إلى مكونين هما:

مكون الشيفرة ويمثل شيفرة MSIL أو تسمى IL أحياناً، و معرف البيانات Metadata وهي وصف لما يدور حوله البرنامج وتصف تلك البيانات الصفوف والأنواع الأخرى والتي تشكل بمجموعها التطبيق، بدون تلك البيانات Metadata لن نتمكن من تحميل تطبيق ما بصورة آمنة في لغة C# وسيستحيل علينا توفير السرية بدون مُعرف البيانات (metadata) التي تشرح لبيئة زمن التنفيذ كيفية عمل تجُّمع ما (Assembly) وما تحتاجه من قيم للبيانات. وقد أمكن تطبيق الانعكاس Reflection في لغة C# بواسطة معرف البيانات metadata هذا، ويدور موضوع هذا الفصل حول البيانات الوصفية أو معرف البيانات "metadata" وطرق استرجاعه.

ما هو الانعكاس Reflection؟

الانعكاس هو عملية استكشاف داخلية لتطبيق ما دون الولوج إلى شيفرته، يمكن استخدام الانعكاس لإيجاد جميع الصفوف في تجُّمع (Assembly) ما بالإضافة إلى جميع الmethods وال Properties والEvents داخل صنف ما إنطلاقاً من الصنف الأساس الذي اشتُقّ منه هذا الصنف وانتهاءً بال interfaces التي يحققها.

يسمح الانعكاس لبرنامج C# بالتحقق بالتعامل مع ذاته من وجهة نظر خارجية. إذ يمكنك التحقق من كافة المعلومات حول صنف (صف) ما، ويمكن أيضاً استخدام الانعكاس كي تستدعي (Invoke) مناهج (methods) هذا الصنف ديناميكياً أثناء زمن التنفيذ، يمكنك أيضاً إنشاء شيفرة بصورة ديناميكية وتعتبر هذه التقنية خطرة ولا يُنصح بإتباعها لتوليد شيفرة ما بوساطتها.

يسمح لك الانعكاس باستعراض المعلومات في مقطع معرف البيانات metadata (يسمى manifest ضمن المجمعة) كل ذلك يتم في زمن التنفيذ، وبشكل مختصر يمكنك رؤية المعطيات كما يراها المترجم عندما يقوم ببناء التطبيق وإنشاء ملف تنفيذي، عند ذلك سيكون لديك المرونة لتوسيع تطبيقاتك أثناء زمن التنفيذ المشترك، فمثلاً: يمكنك استخدام الانعكاس للتحقق من وجود صفوف معينة في مجلدك الرئيسي وأنْ وجدت تقوم بتحميلها واستخدامها ضمن التطبيق الخاص بك، يُعتبر الانعكاس بمثابة مفهوم مكتبات الربط الديناميكي DLL التي وجِدَت خصيصاً لمعرفة محتوياتها أثناء زمن التنفيذ.

وبدون الانعكاس تتطلب منك ملفات DLL معرفة الصفوف والتوابع والأنواع الموجودة بداخلها قبل البدء باستخدامك لها، يمكن مثلاً أن نقوم بتحميل ملفDLL بشكل ديناميكي ولكن لا يمكنك البحث في مجموعة من ملفات DLL عن تابع ما ذو اسم مُعطى وحتى لو تمكنت من ذلك فلا يوجد طريقة لسؤال النظام عن البارمترات الممكن تمريرها لهذا التابع، أما بواسطة الانعكاس فيمكنك القيام بكل ما سبق.

لماذا نحتاج إلى الانعكاس ؟

يتطلب المترجم ونظام زمن التنفيذ إمكانيات الانعكاس في لغة C#، سواء كنت أو لم تكن تريدها أو تحتاجها، فالانعكاس ضروري في أي لغة تفسيرية، لأن CLR يجب أن يعرف ما يقوم بتحميله.

ويعتبر الانعكاس ضرورياً من أجل إعطاء البرنامج أرقام إصدار دقيقة ومحكمة (باستخدم فضاء الأسماء System.Reflection.Emit)، والتحميل الديناميكي للصفوف ومعالجة مسائل السِّرية والأمن بالشكل المناسب.

بدون الانعكاس لا يمكن ل CLR إنجاز مهام السرية والآمن بشكل مناسب، وذلك يعود لعدم قدرتها على تحديد فيما إذا كان من الضروري تحميل كتلة من الشيفرة أم لا قبل تواجدها مسبقاً في الذاكرة.

كيف يُستخدم الانعكاس ؟

لكي نستوعب الانعكاس يجب بداية علينا فهم المجمعات Assemblies والبيانات الوصفية.

المجمعة Assembly هي الوحدة الأساسية للنشر في تطبيق .NET، و قد رأينا سابقاً أن المجمعة تحوي شيفرة MSIL و بيانات وصفية metadata والقوانين التي يُنفذ وفقها التطبيق ويُشَغَل داخل بيئة زمن التنفيذ(هذه القوانين تتمثل في السرية وفي إعطاء البرنامج أرقام إصدارات خاصة به كما رأينا في الفصول السابقة). ملف المجمعة هو ملف يصف نفسه بنفسه ويتضمن شيفرة MISL أو IL والمحمَّلة من قِبَل CLR عندما يُحمل التطبيق إلى الذاكرة.

يتضمن البيان (manifest) معلومات حول المجمعة ككل، وتشمل هذه المعلومات على رقم الإصدار والتسمية الممكن استخدامها من قِبَل CLR للتأكد من أنك تقوم بتحميل الإصدار المناسب لمجمعة عندما تريد استخدامها في تطبيقك، بالإضافة إلى ذلك يمكنك تحميل كامل الرزمة (Package) إلى الذاكرة في وقت واحد، وذلك يعود إلى أن البيان يتضمن إشارات ومراجع لمجمعات أخرى وبالتالي يمكن لبيئة زمن التنفيذ أن تتأكد من تواجد كامل أجزاء التطبيق قبل البدء بتشغيل البرنامج. هذه التقنية تتجنب مشكلة فقدان ملف DLL المتفشية في السابق في البرمجة لنظام التشغيل Windows.

يتضمن البيانات (manifest) معلومات أمن وسرِّية تُفصِّل الحدود المسموح للتطبيق الولوج إليها أثناء تشغيله. هذا المفهوم هام جداً لأنه يُمكنك من إنشاء العديد من أنواع التطبيقات في لغة C# (يمكن التحكم بصلاحيات التطبيق قبل الولوج إلى خدمات النظام).

ما هو التحميل الديناميكي ؟

لنلقي نظرة على أبسط شكل من أشكال الانعكاس ألا وهو تحميل مجمعة أثناء زمن التنفيذ والتعرف على نوع المعلومات التي يمكن أن يجدها والمتعلقة بهذا التجمع، وللقيام بذلك عليك استخدام فضاء الأسماء التالي:

```
using System.Reflection;
```

يمكن تحميل أي مجمعة وذلك بواسطة الصف Assembly كمايلي:

```
Assembly a = Assembly.LoadFrom(@"D:\atom.dll");
```

الآن يمكن استخراج كافة الأنماط ضمن المجمعة التي قمنا بتحميلها وذلك يتم بواسطة التابعGetTypes حيث يعيدها المنهج مصفوفة كائنات من الصف Type التي تتضمن معلومات عن الأنواع المختلفة المُعرفة ضمن هذه المجمعة.

```
Type[] types=a.GetTypes();//Get an All Types in the Assembly
```

الآن نقوم بسؤال الكائن Type عن ماذا يمثل؟

يمكننا تحديد ماهية النوع الذي نتعامل معه وذلك من خلال الخصائص التالية :

الخاصية Property	Description
isClass	إذا كانت هذه الخاصية true يكون هذا النوع هو Class
isEnum	إذا كانت هذه الخاصية true يكون هذا النوع هو Enum
isInterface	إذا كانت هذه الخاصية true يكون هذا النوع هو Interface
isValueType	إذا كانت هذه الخاصية true يكون هذا النوع هو ValueType

هناك خاصيتين لتحيد نطاق الولوج (Access Modifier) هما: IsPublic وIsNotPublic، فمثلاً لاستعراض كافة الأنواع العامة في تجمع ما يمكنك كتابة ما يلي:

```
Assembly a = Assembly.GetExecutingAssembly(); // Get the Current Assembly

Type[] types = a.GetTypes(); // Get an All Types in the Current Assembly

foreach(Type t in types)
{
      if (t.IsPublic)
            Console.WriteLine("Class {0} is Public", t.FullName);
}
```

هنا يظهر سؤال هام حول الانعكاس، لما كان بإمكاننا النظر إلى داخل صف ورؤية ما يزودنا به هذا الصف وأنواع البارمترات التي تتقبلها توابع هذا الصف، فهل يمكننا أن نحصل على نسخة من شيفرة التابع (أو المنهج)؟ إذا كان الجواب نعم، سيكون من السهل سرقة الخوارزميات والشيفرات من مجمعة .NET. بغرض بيعها من قِبَل الأشخاص عديمي الضمير ولهذا السبب فالجواب لا، أنت غير قادر على استرجاع أي شيفرة كانت.

الجدول التالي يحوي التوابع الخاصة بالصف Type والتي تعطي معلومات عن الأنواع بشكل عام، مثلاً: في حال كان الكائن من الصفType يمثل Class عندها يمكن استدعاء التابع GetMethods لاسترجاع كافة الـ methods ضمن الصف.

اسم المنهج method	Description
GetTypes	استرجاع كافة الأنواع لتجمع مُعطى (خاص بالصف Assembly)
GetMethods	استرجاع كافة مناهج (method) صف ما مُعطى
GetConstructors	استرجاع كافة بواني صف ما مُعطى
GetProperties	استرجاع كافة خصائص صف ما مُعطى
GetEvents	استرجاع كافة أحداث صف ما مُعطى
GetInterfaes	استرجاع كافة الواجهات التي يحققها صف ما مُعطى.

الحصول على معلومات الصف والنوع من تجمُّع ما

الغاية من الانعكاس هي إيجاد معلومات حول الصفوف والأنواع في لغة C# دون الحاجة إلى النظر إلى شيفرة البرنامج يدوياً.يُعتبر الانعكاس أداة مفيدة لأعمال التوثيق Documentation، تصور أنك تبحث عن صف يزودك بمناهج معينة عندئذٍ يمكنك استخدام الانعكاس لتكتب أداة تسمح لك برؤية كافة الصفوف الموجودة في كافة التجمعات الموجودة في شركتك.

لنرى المثال التالي :

```
public static void ShowInterfaces(Type t)
{
   Type[] arr = t.GetInterfaces();

    Console.WriteLine("Interface is : ");
    foreach (Type type  in arr)
    {
          Console.WriteLine("Interface  : {0}",type.FullName);
          if(type.IsPublic)
                Console.WriteLine("Scope:Public ");
          else
                Console.WriteLine("Scope: Not Public ");
    }
 }

public static void ShowEvents(Type t)
{
    EventInfo[] events = t.GetEvents();

    Console.WriteLine("Implemented Events  is : ");
    foreach (EventInfo e in events)
    {
        Console.WriteLine("Event name: {0}",e.Name);

        Console.WriteLine("MultiCast : {0}", e.IsMulticast?"Yes":"No");

        Console.WriteLine("Member Type {0}",e.MemberType.ToString());
    }
}

public static void ShowTypes(string name,Assembly assembly)
{
    Type[] typearr = assembly.GetTypes();

    Console.WriteLine("assembly name: {0}",name);

    foreach (Type type in typearr)
    {
```

```
        //Check if type is Class
        if(type.IsClass)
        {
             Console.WriteLine("namespace : {0}",type.Namespace);

             Console.WriteLine("class : {0}",type.FullName);

             if(type.BaseType!=null) // if the Calss has Father(Base Class)

                 Console.WriteLine("Base Class:{0}",type.BaseType.FullName);
             else
                 Console.WriteLine("there is No Father");

             //Check if Abstract
              if(type.IsAbstract)
                   Console.WriteLine("Abstract Base Class");
              else
                   Console.WriteLine("Instantiable Class");

             if(type.IsPublic)
                  Console.WriteLine("Scope:Public ");
             else
                  Console.WriteLine("Scope: Not Public ");

            ShowInterfaces(type); //Get an  interfaces in this Class

            ShowEvents(type); // Get an Events in this Class
         }
         else if(type.IsInterface)
         {
             Console.WriteLine("namespace : {0}", type.Namespace);

             Console.WriteLine("Interface : {0}", type.FullName);

             if (type.IsPublic)
                 Console.WriteLine("Scope:Public ");
             else
                 Console.WriteLine("Scope: Not Public ");

         }
         else if(type.IsEnum)
          {
             Console.WriteLine("Enumeration  : {0}", type.FullName);
          }
          else//may be type is Primitive Data Type Such as int,string...
                 Console.WriteLine("Interface : {0}", type.FullName);
    }
}
```

```
static void Main(string[] args)
{
    Assembly a = Assembly.LoadFrom(@"D:\myAssembly.dll");

    ShowTypes(a.FullName, a);
}
```

الحصول على معلومات حول عضو ما من صنف

قد نرغب أحياناً في أن نحصل على معلومات تخص كل صنف ضمن المجمعة، المثال التالي يوضح كيفية الحصول على الخصائص والمناهج :

```
public static void ShowMethods(Type t)
{
     MethodInfo[] methods = t.GetMethods();
     foreach (MethodInfo method in methods)
     {
          Console.WriteLine("Method name : {0}", method.Name);
          Console.WriteLine("Return Type : {0}", method.ReturnType);
      }
}

public static void ShowProperties(Type t)
{
   PropertyInfo[] properties = t.GetProperties();
   foreach (PropertyInfo property in properties)
   {
       Console.WriteLine(" property name : {0}", property.Name);
   Console.WriteLine("property Type : {0}", property.MemberType);
   }
}

static void Main(string[] args)
{
     Assembly a = Assembly.LoadFrom(@"D:\myAssembly");

     Type[] arr = a.GetTypes();

     Console.WriteLine(" Assembly name : {0}",a.FullName);

     foreach (Type type in arr)
     {
          if(type.IsClass)
          {
                Console.WriteLine(" Class name : {0}", type.Name);

                 ShowMethods(type);
```

```
                ShowProperties(type);

            }
        }
}
```

الاستدعاء الديناميكي لمناهج صفوف تنتمي إلى تجمع ما

إذا كنت تريد أن تُنشئ لغة برمجة ذات نصوص قصيرة (Scripting Language)، أو أي شكل آخر من أشكال المفسرات، سيكون من المفيد إنشاء كائنات يمكنك إيجاد معلومات خاصة بها. بالإضافة إلى ذلك إمكانية استدعاء مناهج (methods) وخصائص تلك الكائنات بعد أن تحصل عليهم، كل هذا يتم بواسطة الانعكاس إذ لا يقتصر الأمر على إمكانية الحصول على معلومات حول صنف ضمن تجمع ما، وإنما يمكن استدعاء المناهج لتلك الكائنات وبشكل ديناميكي هذا إذا كنت تعرف اسم التابع المستدعى وبارمتراتهُ التي يأخذها، كما يمكنك أيضاً أن تحصل على القيمة التي يردها التابع الذي تستدعيه.

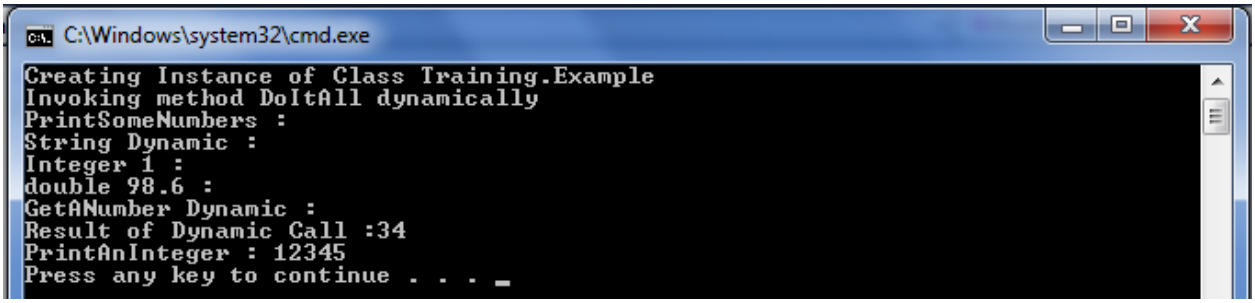
يجب على الصف الذي نود إنشاء كائن منه أن يحقق واجهة بينية معينة (Interface).

```
interface IExample
{
        void PrintAString(string s);

        void PrintAnInteger(int i);

        void PrintSomeNumbers(string s, int i, double d);

        int GetANumber(string s);
}

class Example:IExample
{
    public void PrintAString(string s)
    {
         Console.WriteLine("PrintAString : {0}", s);
    }

    public void PrintAnInteger(int i)
    {
          Console.WriteLine("PrintAnInteger : {0}", i);
    }

    public void PrintSomeNumbers(string s, int i, double d)
    {
         Console.WriteLine("PrintSomeNumbers :");

         Console.WriteLine("String :{0} :",s);
```

```
        Console.WriteLine("Integer :{0} :", i);

        Console.WriteLine("double :{0} :", d);
    }

    public int GetANumber(string s)
    {
            Console.WriteLine("GetANumber {0} :", s);
            return 34;
    }

    public int DoItAll(string s, int i, double d)
    {
            IExample ex = this;

            ex.PrintSomeNumbers(s, i, d);

            return ex.GetANumber(s);
    }
}

static void Main(string[] args)
{

          // الحصول على مجمعة التطبيق الحالية
         Assembly a = Assembly.GetExecutingAssembly();

         Type[] arr = a.GetTypes();

         foreach (Type type in arr)
         {
              if (type.IsClass == false)
                       continue;
              if (type.GetInterface("IExample") == null)
                       continue;
              Console.WriteLine("Creating Instance of Class {0}"
                       , type.FullName);

             object obj = Activator.CreateInstance(type);

             object[] arg1 = {"Dynamic",1,98.6};

             object result;

             Console.WriteLine("Invoking method DoItAll dynamically");

              try
              {
                   result = type.InvokeMember("DoItAll"
                       ,BindingFlags.Default | BindingFlags.InvokeMethod
```

```
                        , null, obj,arg1);
                    Console.WriteLine("Result of Dynamic Call :{0}",result);

                     //Call an interface method
                     object[] arg2 = {12345};

                     type.InvokeMember("PrintAnInteger"
                        , BindingFlags.Default | BindingFlags.InvokeMethod
                        , null, obj, arg2);
                }
                catch(Exception e)
                {
                     Console.WriteLine(e.Message);
                }
            }
}
```

ويكون الخرج كما يلي :



التحقق من احتواء صف ما على منهج

الكود التالي يوضح كيفية البحث عن منهج ضمن صنف معين بداخل مجمعة.

```
static void Main(string[] args)
{
     string methodName="DoItAll"; // اسم التابع المراد البحث عنه

      // if you Want to search in A Specific Assembly you must be Call
     // Assembly.LoadFrom(path of Assembly)
      Assembly a = Assembly.GetExecutingAssembly();

     foreach (Type type in a.GetTypes())
     {
         // Is this a Class ?
         if (type.IsClass == false)
            continue;// No, Skip it
         //Get All Methods in Class
         foreach (MethodInfo  m in type.GetMethods())
         {
              if (m.Name== methodName)
                    Console.WriteLine("Class {0} Contains method {1}"
                        ,type.FullName, m.Name);
         }
      }
}
```

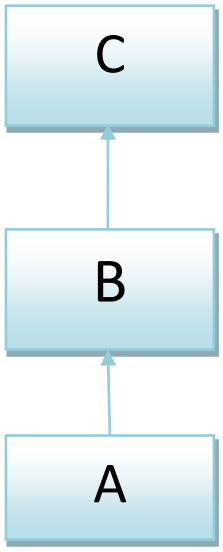
والخرج كما يلي :

تحديد فيما إذا كان صف ما مُشتق من صف آخر

قد نحتاج أحياناً لمعرفة ما هو الصف الأساس(الأب) لصف مُعطى، أو إيجاد الصفوف التي ترث من صف معين (إيجاد جميع أبناء هذا الصف)، أو حتى إيجاد الواجهات التي يحققها صف معين.

نريد معرفة جميع الصفوف التي ترث من الصف C بشكل مباشر مثل B، أو بشكل غير مباشر مثل A.

لنرى الكود التالي:




```
class A:B
 {

 }

 class B:C
 {

 }
 class C
 {

 }

  static void Main(string[] args)
   {
        string classname = "C";

          Assembly a = Assembly.GetExecutingAssembly();// if you Want //to search
          in A Specific Assembly you must be Call
         //Assembly.LoadFrom(path of Assembly)

        foreach (Type type in a.GetTypes())
        {
            // Is this a Class ?
            if (type.IsClass == false)
                continue; // No, Skip it
            //
            Type Basetype = type.BaseType;
            while (Basetype != null)
            {
               if (Basetype.FullName.EndsWith(classname))
                       Console.WriteLine("Class {0} is derived
                      from{1}",type.FullName, classname);

               Basetype = Basetype.BaseType;
```

```
            }
        }
    }
```

الخرج كما يلي:

استعراض المناهج والبارمترات الخاصة بصف معين

قد تحتاج أحياناً إلى كتابة أداء توثيق (Documentation tools)، فإنك تحتاج في ذلك الحين إلى إخراج كافة المعلومات الخاصة بالصف بما فيها البواني Constructors وال methods مع بارمترات كل method، لنرى الشيفرة التالية:

```
//Print Paramerters information for method
public static void PrintParameters(ParameterInfo [] pars)
{
     foreach (ParameterInfo parameterInfo in pars)
     {
           Console.WriteLine("Parameters name : {0}"
                             , parameterInfo.Name);
          Console.WriteLine("Parameters Type : {0}"
              , parameterInfo.ParameterType);

           // is the Parameters is input Parameters
          Console.WriteLine("Parameters is in  ? : {0}"
                  , parameterInfo.IsIn);

          Console.WriteLine("Parameters is out ? : {0}"
                  ,parameterInfo.IsOut);
      }
 }

//Get All method and Constructors in the Calss
public static void GetMethodandConstructor(Type t)
{

     ConstructorInfo[] cons = t.GetConstructors();
     foreach (ConstructorInfo constructorInfo in cons)
     {
           //Get All Parameters
            ParameterInfo[] pars =constructorInfo.GetParameters();
```

```
            Console.WriteLine("Constructor : {0}"
                  , constructorInfo.Name);

            PrintParameters(pars);
    }


    MethodInfo[] methods = t.GetMethods();
    foreach (MethodInfo methodInfo in methods)
     {
          Console.WriteLine("Method : {0}",methodInfo.Name);

       Console.WriteLine("Return Type: {0}", methodInfo.ReturnType);

          PrintParameters(methodInfo.GetParameters());
      }

 }

static void Main(string[] args)
{
      string classname = "Example";

      Assembly a = Assembly.GetExecutingAssembly();
    // if you Want to search in A Specific Assembly you must be Call
        //Assembly.LoadFrom(path of Assembly)

        foreach (Type type in a.GetTypes())
        {
            // Is this a Class ?
            if (type.IsClass == false)
                continue; // No, Skip it
            //Only the One We Want

            //if type is Class then You can reach to this Point
            if (classname == type.Name)
                GetMethodandConstructor(type);

        }
    }
```

للمزيد يمكن الإطلاع على المقال التالي:

http://www.codeproject.com/Articles/55710/Reflection-in-NET

الفصل الرابع

التوافقية بين .NET

ونموذج COM

COM & Win API

مقدمة

يُعتبر NET Framework. حجر بناء جديد في عالم Windows، ويتوقع من تطبيقاتNET. أن تتعامل في الفترة القادمة مع تقنيات WINDOWS الموجودة وينطبق هذا الكلام خصوصاً على مجالين:

1- نموذج كائن المكون (**C**omponent **O**bject **M**odel) والمعروف إختصاراً COM.
2- Windows API.

كانت COM تقنية Microsoft الأصلية لبناء مكونات برمجية مستقلة عن لغة البرمجة. تستخدم COM بشكل واسع في مستوى النظام لكونها حجر البناء الرئيسي في نظام ويندوز، وتشاهد غالباً في التطبيقات لأنها تشكل أساساً للعديد من عناصر تحكم ActiveX المستخدمة بكثرة في مشاريع Visual Basic و C++. سنرى لاحقاً أن NET. تقدم طريقة للعمل المشترك بين كائناتNET. و COM وتُعرف هذه التقنية ب COM Interop[1].

إن واجهة Windows API تعبر عن مجموعة من التوابع المستخدمة من قبل مبرمجي windows لكتابة تطبيقات windows.، يقدم NET. طبقة غرضية التوجه فوق Windows API، قد نحتاج في بعض الحالات إلى استدعاء تابع API لا يمكن الوصول إليه عبر NET. (تابع غير مُدار Unmanaged)، يمكن أن يقوم هذا التابع بتشغيل الكاميرا في الحاسب أو أن يكون له وظيفة في مستوى منخفض من نظام التشغيل.
نستطيع في هذه الحالات استخدام آلية إقحام NET. (NET Platform Invoke.) أو اختصاراً P/Invoke وذلك الاستدعاء توابع C أوC++ من خلال NET.. بما أن توابع Windows API موجودة ضمن ملفات DLL، فأن P/Invoke تقدم طريقة عامة لاستدعاء توابع C أو C++ في ملفات DLL ضمن شيفرة NET..

ما هي COM ؟

إذا كانت لديك فكرة مسبقة عن COM يمكنك الانتقال إلى الفقرات التالية، حيث سنشرح كيفية استخدام مكونات COM داخل شيفرة NET.
COM هي عبارة عن مواصفات (Specification) لكتابة العناصر البرمجية المصممة لكي تكون مستقلة عن لغة البرمجة والمنصة، وهذه المواصفات هي مجموعة من القواعد التي على المطورين إتباعها إذا كانوا يريدون لكائناتهم أن تعمل ككائنات COM، إذا كانت شيفرتك تتبع هذه المواصفات عندها ستكون هذه الشيفرة هي مكون COM وستستطيع العمل مع كائنات COM الأخرى.
أن أساس COM هو الواجهات Interfaces وهي ببساطة مجموعة من عناوين التوابع داخل كائن COM. إذا تمكن العميل (الشخص الذي سيستخدم هذا المكون) من الحصول على هذه المؤشرات يمكن عندها استخدام العناوين لاستدعاء التوابع. تكون الواجهات على شكل مصفوفة من المؤشرات في الذاكرة، وتوجد آليات لمساعدة شيفرة العميل لإيجاد الواجهات المعروضة من قبل الكائن.

[1] للمزيد يمكن الإطلاع على الرابط التالي : http://msdn.microsoft.com/en-us/library/aa645736(v=vs.71).aspx

تُعرّف الواجهات بِمُعرفين هما :
1- اسم الواجهة : عبارة عن اسم يبدأ ب حرف I مثل IDispatch.
2- رقم مُعرف ID: يسمى عادةً بمعرف الواجهة، و واجهات ID هي أحد الاستخدامات الخاصة للمعرفات الوحيدة العامة Globally Unique Identifiers أو اختصاراً GUI، و GUIDهو معرف فريد ب 128 بت لا تتكرر مع غيرها لتعريف عناصر في COM.

تستخدم (GUIDs) لتعريف أنواع كائنات COM (أو الأصناف المشتركة) والواجهات وأي شيء آخر يحتاج إلى هوية فريدة وتكتب على شكل سلسلة من الخانات الست عشرية.
تُخزن جميع المعلومات المتعلقة بـ COM وواجهاتها في سجل ويندوز Windows Registry مستخدمة (GUIDs) لتعريفها.
أحد مشاكل العمل مع COM هي أن المعلومات حول الكائنات (مكان تواجدها ووظيفتها) تُحفظ خارجاً في سجل ويندوز. في حال تم كسر الارتباط بطريقة أو بأخرى (ربما يتشوه مدخل التسجيل أو بنقل الكائن إلى مكان آخر دون تحديث التسجيل)، عندها لا يمكن الوصول إلى كائن COM وسيصبح العملاء غير قادرين على استخدامه. ولكن مع NET. تم تجاوز هذه المشكلة من خلال ميزة الوصف الذاتي Self Description.
نظراً لأن COM أنتجت بعد جميع لغات برمجة Windows المعروفة، وجب على COM تعريف مجموعة من الأنواع كي تستخدم في التواصل بين اللغات، بالإضافة إلى مجموعة من القواعد والآليات التي تضمن أن استدعاءات التوابع بين اللغات المختلفة ستؤدي إلى استمرار العمل بشكل صحيح. الأمر الذي أدى إلى وجوب الدقة في استخدام COM باستخدام C++.

مكتبات الأنواع (Type Libraries) ولغة تعريف الواجهات IDL

أحد مبادئ COM الأساسية أنه يتوجب على العملاء البحث عن ما يستطيع كائن COM به، أي أنهُ يستطيع العملاء معرفة الواجهات التي يدعمها كائن COM والتوابع التي تشكل هذه الواجهات، هذا يعني أنه يجب على كائن COM أن يملك مكافئاً لبيانات التعريف ''Metadata'' في NET.، أي أنه يجب أن يؤمن طريقة لوصف ما يقوم به كائن COM من خلال تبيان ما تدعمه من واجهات. يحدث هذا في COM باستخدام لغة تعريف الواجهات Interface Definition Language أو اختصاراً IDL و معلومات الأنواع.

تسخدم IDL كطريقة لتدوين وصف كائنات COM واجهاتها وقد اعتاد مبرمجو COM في عالم C++ استخدام IDL. وعند الترجمة تستخدم أداة تسمى MIDL لترجمة IDL بتحويلها إلى معلومات ثنائية تُعرف باسم مكتبة النوع Type Library، وتكون مكتبة النوع جزءاً من شيفرة كائن COM مرتبطة بملف EXE أو DLL أو قد تكون في ملف منفصل وأحياناً لا يكون لدينا مكتبة أنواع.

تحوي مكتبة النوع Type Library معلومات عن أسماء وأرقام تعريف (IDs) الواجهات بالإضافة إلى التوابع والبارمترات الخاصة بها ، كل هذه المعلومات متوفرة في مجمعة NET. على شكل ''Metadata''،الآن لدينا مشكلة وهي أن مكتبة النوع غير قابلة للتوسع (أي لا يمكن تعديلها لأنها تحوي شيفرة ثنائية وهي ثابتة)، هذه المشكلة غير موجودة أصلاً في NET MetaData. لأنها قابلة للتوسيع وذلك باستخدام الصفات المخصصة Custom Attributes.

- تسمح NET. باستكشاف إمكانيات COM بطريقتين :

1- من خلال قراءة مكتبة النوع أثناء برمجتك للشيفرة، وبذلك يعلم المترجم ما يستطيع كائن COM القيام به ويقوم بإنجاز فحوصات وقت الترجمة، مثل فحص أنواع الوسائط للتوابع ، تُعرف هذه الطريقة بالربط المبكر Early Binding، وذلك لحدوثه في وقت الترجمة يمكنك إنجاز الربط المبكر بإضافة Reference إلى كائن COM في مشروع NET. (Add Reference ثم نحتار تبويبCOM ثم نختار ما نريد).

2- يمكن من خلال NET. إنشاء كائن COM بشكل ديناميكي أثناء وقت التشغيل ومعرفة ما يقوم به، وحتى يمكنك أن تسأل كائن COM إذا كان يدعم واجهة معينة، تسمى هذه الآلية بالربط المتأخر Late Binding لأنها تحدث في وقت التشغيل.

COM and .NET

في هذه الفقرة سنوضح بعض المفاهيم المشتركة بينNET. و COM، وسنوضح طرق تحقيقها علماً أن هذه الطرق مختلفة بينهما.

1- MetaData:

تم التفصيل في شرحها في الفقرة السابقة، كما وضحّنا كيفية تخزين البيانات الوصفية في كل من NET. و COM.

2- freeing Memory "تحرير الذاكرة" :

نعلم أن تحرير الذاكرة في NET. يتم من خلال "Garbage Collector"، ولكن في COM الطريقة تختلف تماماً، حيث تعتمد COM على "Reference Counts" وهو: عدد المؤشرات Pointers التي تؤشر على هذا الكائن، وعندما يصبح مساوياً للصفر يتم قتل "Destroy" الكائن.

الواجهة IUnKnown وهي من أهم الواجهات التي يجب على كائن COM أن يحققها، تقدم هذه الواجهة ثلاثة مناهج methods اثنان منهما يتعلقان بال "Reference Counts"، التابع (method) ()AddRef يجب أن يُستدعى من قبل العميل إذا قام مؤشر من واجهة معينة بالإشارة إلى هذا الكائن، وعندها هذا التابع سيقوم بزيادة قيمة ال "Reference Counts"، التابع ()Release يقوم بإنقاص ال "Reference Counts" عندما ينتهي العميل من استخدام المؤشر على الكائن، وإذا كانت ال "Reference Counts" تساوي الصفر تتم إزالة الكائن من الذاكرة.

3- الواجهات Interfaces : وقد تكلمنا عنها في الفقرة السابقة .

4- method Binding : تحدثنا عنها سابقاً.

5- Data Types : نقصد بها الأنماط المشتركة البارمترات التوابع المستدعاة بين لغات البرمجة المختلفة وقد تكلمنا عنها في الفقرة السابقة.

6- Registration :

تكلمنا في فصول سابقة عن مزايا المجمعات الخاصة والمشتركة، أما في COM جميع المعلومات حول الكائن موجودة في ال Registry Configuration.
لكل كائن COM مُعرف GUID مكون من 128 بت ويسمى أيضاً ب Class ID اختصاراً CLSID، إنشاء كائن COM يتطلب استدعاء التابع ()CreateInstance والبحث ضمن التسجيل عن CLSID وإيجاد مسار ملف ال DLL أو EXE وتحميله والقيام بعملية إنشاء كائن COM المطلوب.
نظراً لأن ال CLSID صعب الحفظ أو التذكر، يوجد طريقة أخرى لتعريف كائنات COM وهي ProgID وهي عبارة عن اسم فريد مقابل ل CLSID.

7- Threading.

8- Error Handling :

في NET. يتم توليد الأخطاء عن طريق رمي استثناء، أما في COM فالأخطاء تُعرف عن طريق القيمة المُعادة من التوابع (HRESULT)، عندما تكون قيمة HRESULT هي s_ok يكون التابع قد تم تنفيذه بشكل صحيح.
في حال أردت الحصول على تفاصيل إضافية يجب على كائن COM أن يحقق الواجهة ISupportErrorInfo، والتي تزودنا برسالة الخطأ بالإضافة إلى رابط لملف المساعدة بالإضافة إلى مصدر الخطأ، وتَرد كائن يحوي معلومات عن الخطأ الذي حدث.

9- Events Handling.

استخدام مكون COM في شيفرة NET.

لقد كانت تطبيقات COM منذ فترة وجيزة المعيار لقابلية تشغيل التبادلية لتطبيق على منصات تشغيل ويندوز. قد يكون لديك فكرة عظيمة عن شيفرة مكتوبة ومجمعة كمكون COM ولا تريد إعادة إنشاءها، بل تريد البدء بالاستفادة من إمكانيات NET.، فهل يتوجب عليك إعادة كتابة جميع الوظائف؟

لحسن الحظ لا يتوجب عليك فعل ذلك فقد امتلكت شركة Microsoft بعض ميزات قابلية تشغيل COM التبادلية COMIntrop، إضافة إلى سهولة استخدامها في زمن تنفيذ اللغة المشتركة CLR ومكتبة أصناف NET..

يمكنك ببساطة استخدام كائنات COM من شيفرة NET. باستخدام المغلف القابل للاستدعاء وقت التنفيذ إختصاراً RCW، وهو صنف وكيل (proxy Class) يجعل كائن COM يعمل تماماً كما يعمل كائن NET.، يقوم ال RCW بتبيان كل طرائق Methods وخصائص كائن COM واستخدام واجهته وذلك في وقت التشغيل Runtime.

يمكن إنشاء RCW بإحدى طريقتين:

1- من القائمة Project نختار Add Reference ثم نختار تبويب COM ومن ثم نختار كائن COM الذي نريده ، وعندها يتم إنشاء RCW باسم افتراضي هوIntrop.COMServerLib.dll بعد ذلك نقوم بعمل using namespace للصف الذي نود إنشاء كائن منه ويمكننا معرفة اسم الصنف الجديد من خلال Object Browser.

2- من خلال سطر الأوامر باستخدام الأداة TlbImp.exe كما يلي:

```
>tlbimp COMServer.dll /out :Introp.COMServer.dll
```

حيث COMServer.dll يمثل كائن COM و Introp.COMServer.dll يمثل مجمعة التغليف RCW، تحوي هذه المجمعة بيانات وصفية "metadata" توصف الأنماط في كائن COM وهذه المعلومات يستخدمها CLR في وقت التنفيذ.

ملاحظة: يمكن استخدام كائن NET. من داخل شيفرة COM، كما يمكن إنشاء كائن من صف يمثل مكون COM (موجود في DLL) باستخدام ال Later Binding والصفين Type و Activator.

استدعاء تابع غير مُدار من ملف DLL باستخدام .NET P/Invoke

في بعض الأحيان يكون من الضروري استدعاء تابع موجود في ملف DLL خارج .NET، ربما يكون أحد توابع Win32 API ولا يوجد مكافئ له في .NET، لذلك تقدم لنا .NET آلية إقحام (P/Invoke) لكي نتمكن من استدعاء التوابع الموجودة في ملفات DLL من داخل شيفرة .NET، للقيام بهذه العملية يجب أن نقوم بما يلي:

1-إيجاد ملف Dll الذي يحوي التابع.
2- تحميل ملف DLL إلى الذاكرة.
3- إيجاد عنوان التابع ودفع الوسائط إلى المكدس.
4- استدعاء التابع.

لكي يتم استدعاء تابع من ملف DLL يجب أولاً معرفة اسم التابع أو رقمه الترتيبي (يمكن استدعاء التابع عن طريق أحدهما) واسم ملف DLL الذي يحوي التابع.

سنقوم في المثال التالي باستدعاء تابع غير مُدار من ملف DLL من داخل شيفرة .NET، سنستخدم التابع MessageBox() من Win32 API وذلك كما يلي:

```
using System;
using System.Runtime.InteropServices;

[DllImport("user32.dll", CharSet = CharSet.Auto)]
public static extern int MessageBox(int hwnd, string text
            , string caption, uint type);

static void Main(string[] args)
{
     MessageBox(10, "Title", "welcome", 20);
}
```

الفصل الخامس

التعامل مع

XML

XML

مقدمة

تحتل XML موقعاً هاماً في .NET، إذ تقدم طريقة تخزين بسيطة وبنيوية لتخزين وتبادل المعطيات من أجل استخدامها في البيئة الموزعة. جعلت Microsoft تقنية XML الطريقة الأساسية لتبادل المعطيات بين أجزاء تطبيقات ويب الموزعة، وفي توصيف البروتوكول SOAP المستخدم في خدمات الويب[1]، كما تستخدم أيضاً مع قواعد البيانات باستخدام ADO.NET، وفي التوثيق التلقائي Documentation حيث يتم حفظ التوثيق التلقائي للشيفرة المصدرية Source Code على شكل XML، حتى أن ملفات التكوين (الإعدادات Configuration) لمشروع معين تُحفظ على هيئة XML.

سنتكلم في البداية قليلاً عن ماهية XML ثم سننتقل إلى آليات للتعامل معها.

ما هي XML ؟

XML هي عبارة عن صيغة أو هيئة (Format) لتخزين البيانات على هيئة نصية بسيطة، وسنرى لاحقاً أن سر قوة XML يكمن في بساطتها وهرمية بنائها.
الجميع في هذه الأيام يعرفون HTML واستخدامها في بناء صفحات ويب، ومعظمهم يعرفون العلامات (Tags) ولو بشكل سطحي، على الرغم من أن HTML مفيدة جداً لغرضها الأساسي وهو تنسيق صفحات الويب، إلا أن معظم الناس يجدونها قاصرة حيث أن أنها تصف تنسيق المعطيات فقط، وليس ما تمثله المعطيات، مثال:

```
<h1> Book1 </h1>
<h2> Book2 </h2>
```

تشير العلامات h1 و h2 إلى المستويين الأول والثاني للعناوين (حجم الخط). لكن لا يوجد في المعطيات ما يشير إلى تمثله هذه المعطيات، لنفرض أن لنفرض أن هذه المعطيات تمثل كتاب لكن لا يوجد أي شيء في العلامات يدل على أنها كذلك!. إذاً تُستخدم HTML فقط لتمرير معلومات تنسيقية إلى المستعرض، المشكلة أيضاً أن Tags الموجودة في HTML ثابتة لذلك يصعب نقل معلومات إضافية من مستند HTML دون اللجوء إلى توسيعات غير قياسية.

نشأت XML نتيجة اكتشاف أن التعقيد المتزايد للويب يتطلب طرقاً أقوى من HTML لتوصيف المعطيات.
لقد كان يوجد حل عام معقد من خلال لغة التوصيف العامة القياسية SGML (وهي لغة قادرة على توصيف قواعد صرفية وتعليمات لغات تحوي Tags)، لكنها معقدة للغاية وبذلك قامت XML بتبسيط SGML بشكل كبير، حيث قامت مجموعة معايير الويب W3C[2] بعمل نسخة مصغرة من SGML نزعت منها المعايير المعقدة والصعبة والزائدة[3]. تستخدم XML في مجال واسع من التطبيقات منها:

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://www.w3schools.com/webservices/ws_soap_intro.asp
[2] للمزيد يمكن الإطلاع على الرابط التالي: http://www.w3.org/
[3] للمزيد يمكن الإطلاع على الرابط التالي: http://www.w3.org/MarkUp/SGML/Overview.html

- **توصيف المستندات:** أصبح العديد من المستخدمين يخزنون المعلومات النصية على شكل XML عوضاً عن صيغة Word أو PDF، ويمكن لاحقاً تحويلها إلى HTML لعرضها على الشبكة أو يمكن تحويلها إلى صيغة PDF.
- **تبادل البيانات:** يمكن استخدام XML لتبادل البيانات بين تطبيقات مختلفة ومتنوعة لأنها سهلة التوليد والنقل والتفسير. إنَّ طبيعة XML الهرمية تسهل تمثيل المعطيات المركبة مثل قواعد المعطيات، كما يعتمد البروتوكول SOAP في تبادل الرسائل البعيدة على مستندات XML.
- **تخزين المعطيات:** تقدم XML طريقة بسيطة و معيارية لتخزين المعطيات وأصبحت معظم التطبيقات تتخاطب مع بعضها بواسطة ملفات XML.
- **عمليات قواعد المعطيات:** أصبحت العديد من قواعد البيانات بما فيها Access و SQL Server تعيد نتائج الاستعلامات على هيئة ملفات XML، مما يسهل العمل على المعطيات من قبل برامج أخرى، كما يمكن استخدام XML في NET. من اجل تصميم مخططات (Schema) قواعد المعطيات.

بنية مستندات XML

تسمى مجموعة بيانات كاملة في XML بمستند XML، يمكن أن يكون هذا المستند يمثل ملفاً فيزيائياً على جهازك أوعلى شكل سلسلة نصية في الذاكرة إذ أنه يجب أن يخضع إلى مجموعة من القواعد الأساسية والتي سنتناولها لاحقاً.
لننظر إلى مقطع HTML التالي بعد تحويله إلى XML:

```
<auther> mohammad </auther>
<title> how to program</title>
```

على الرغم من أن العلامات تبدو متشابهة إلا أنك ستلاحظ الفرق، فالعلامات Tags هنا تصف ماهية المعلومات وليس مجرد كيفية إظهارها، يعني ذلك أنه من السهل مثلاً البحث ضمن XML لعرض المؤلفين ضمن مجموعة كتب. تختلف XML عن HTML في القدرة على تخصيص علامات XML وجعلها قابلة للتوسع لذلك هي واسعة الانتشار.
أولاً يجب أن نعلم أن مستندات XML مُحكمة الهيئة Well Formed هي تلك المستندات التي تخضع للقواعد التي نصت عليها لغة XML. إن أسماء العناصر متحسسة لحالة الأحرف Case Sensitive ويجب أن نعلم أن لكل معرف بداية <t1> معرف نهاية </t1>أيضاً، في حال مخالفة الشروط السابقة يكون مستند XML غير مُحكم الهيئة وعندها لن تستطيع التطبيقات التي تقرأ ملفات XML والتي تُسمى مُعرِبات XML (XML Parser) سترفض هذا المستند. لنرى المثال التالي:

```
<name>
      <firstname> nizar </firstname>
      <lastnaem> Ganoom Showish </lastname>
      <phone> 45123131 </phone>
</name>
```

لاحظ أن للمستند بنية شجرية وهذا يعني أن هناك عناصر أبناء هي العناصر <firstname> و <firstname> و <phone> لعنصر أب هو العنصر <name> ، يجب أن تعلم أنه لا يمكن التداخل بين العناصر لأن التداخل يخلق بنية شبكية وليس هرمية، أي أن المستند التالي لا يمثل مستند XML محكم الهيئة:

```
<name>
      <firstname> Mu_nizar </firstname>
      <lastnaem> Ganoom Showish </lastname>
      <phone> <mobile> 45123131 </phone> <mobile> // Error
</name>
```

المستند السابق غير نظامي وذلك العنصر <mobile> هو عنصر أبن للعنصر <phone> كما هو واضح من معرفي البداية للعنصرين، وهذا يعني أن معرف النهاية للعنصر <mobile>يجب أن يأتي قبل معرف النهاية للعنصر <mobile> لذلك هو غير محكم الهيئة.

الصفات

بالإضافة إلى حفظ المعطيات في جسم العنصر فأنه يمكن حفظ الصفات ضمنه أيضاً، وتمثل الصفة باسم وقيمة موضوعة ضمن علامتي تنصيص " " كما يلي:

```
Name=”value”
```

وتوضع الصفات ضمن معرف البداية وقبل إغلاقه كما يلي:

```
<book title="Learn C#"> </book>
```

يمكن أن نضع أكثر من صفة كما يلي:

```
<book title="Learn C#" description="learning"> </book>
```

لنرى المثال التالي:

```
<?xml version="1.0"?>
<!-- this is a XML Document--> //this is A Comment
<employees>
  <employee empID="1">
    <fullname>Nancy Davolio</fullname>
    <extension>5467</extension>
  </employee>
  <employee empID="2">
    <fullname>Andrew Fuller</fullname>
    <extension>3457</extension>
  </employee>
  <employee empID="3">
    <fullname>Janet Levering</fullname>
    <extension>3355</extension>
  </employee>
  <employee empID="4">
    <fullname>Margaret Peacock</fullname>
    <extension>5176</extension>
  </employee>
```

```
  <employee empID="5">
    <fullname>Steven Buchanan</fullname>
    <extension>3453</extension>
  </employee>
  <employee empID="6">
    <fullname>Michael Suyama</fullname>
    <extension>428</extension>
  </employee>
</employees>
```

السطر الأول هو تصريح XML وهو ضروري لتفسير الملف ومعظم البرامج لا تستطيع تفسيرهذا المستند بدون تصريح XML، نلاحظ أن تصريح XML هو Tag عادي وله الخاصية (أو الصفة) Version، ويوجد العديد من الخواص الأخرى. السطر الثاني يمثل تعليق، أما العنصر الموجود في السطر الثالث فهو عبارة عن عنصر الجذر (Root)وهو العنصر <employees>.

تقدم XML هرمية لحفظ البيانات وكما نعلم فأن معظم أنظمة البيانات تحفظ البيانات في جداول مرتبطة مع بعضها البعض من خلال عدد من الأعمدة. تحفظ البيانات ضمن أسطر وأعمدة في الجدول حيث يمثل السطر سجلاً واحداً ويمثل العمود في السجل حقلاً واحداً، ويجب أن تكون جميع العناصر داخل المستند موجودة بداخل عنصر واحد يسمى الجذر (شاهد المثال ولا حظ كيف أن العنصر <employees> يحوي بيانات جميع الموظفين وكل موظف يتم توصيفه بمجموعة من العناصر)، يمكن اعتبار العنصر الجذر <employees> هو اسم الجدول وجميع العناصر بداخله (سجلات الموظفين) والتي يتم تخزينها على شكل هرمية من العناصر كما نرى فإن أسماء العناصر Tags ضمن العنصر Tags الجذر تبقى كما هي والذي يختلف هو قيمة هذه العناصر، لهذا يمكننا أن نقوم بتصدير Export قاعدة بيانات كاملة (مجموعة من الجداول) على هيئة مستند XML.

قد نقوم بإرسال قاعدة المعطيات عبر الشبكة مثلاً، ويمكن أيضاً أن نقوم باستيراد (Import) لملف XML وتحميله ضمن قاعدة معطيات.

أن المستند السابق هو مستند XML محكم الهيئة، ونقصد أنه يحقق القواعد التالية:

1- أن يوجد عنصر جذر واحد للمستند ككل.

2- لكل عنصر بداية يجب أن يكون له عنصر نهاية كما يلي:

```
<TAG></TAG>
```

إذا كان فارغ أي لا يحوي قيمة فيمكن كتابته كما يلي:

```
<TAG/>
```

3- قيم الصفات يجب أن توضع ضمن علامتي تنصيص " ".

ويوجد العديد من القواعد الأخرى إلا أننا وضعنا أهمها.

تحدثنا عن المستندات محكمة الهيئة Well Formed فماذا عن المستندات النظامية؟
في الحقيقة إن مستندات xml محكمة الهيئة هي المستندات المحكومة من قبل قواعد XML، بينما مستندات XML النظامية هي التي تخضع إلى قواعد التطبيق المُعرِب (الذي يقرأ مستند XML)، ولكي نتفحص من كون مستند XML فيما إذا كان يخضع لقواعد هذا التطبيق أو لا فأنه يجب علينا أن نجد طريقة لتعريف هذه القواعد.

فضاءات أسماء XML

قد تواجه مشاكل كبيرة إذا حاولت استخدام مستندي XML معاً لأن كاتبي المستندين يكون قد استخدم نفس أسماء العناصر، لنفرض أنه لدينا ملفين XML يحويان معلومات عن الموظفين، المستند الأول يحوي العنصر Address والذي يحوي عنوان الموظف وفي المستند الثاني العنصر Address يحوي البريد الإلكتروني للموظف وذلك كما يلي:

```
    <!--in the employee file -->
    <address> Damascus </address>


   <!--in the email file -->
   <address> muhamed_807@hotmail.com </address>
```

إذا أردت دمج هذين الملفين كيف سنميز بين العناصر المشتركة؟ يتم ذلك بواسطة فضاءات الأسماء، بإعطاء مستوى إضافي لتسمية العناصر باستخدام الصفة xmlns.

```
<employee empID="6"
xmlns:emp="http://microsoft.com/schemas/VisualStudio/TeamTest/2010">
    <fullname>Michael Suyama</fullname>
    <extension>428</extension>

    <!--in the email file -->
     <emp:address> muhamed_807@hotmail.com </emp:address>
  </employee>
```

التحقق من الصحة في XML

قد نتساءل أحياناً هل من الممكن إنشاء العلامات Tags الخاص بنا وإعطاءها المعنى الذي نريده مثلاً: كيف نعرف ما يمكن وضعه في مستند الموظفين السابق وما لا يمكن وضعه وهل يمكن وضع أكثر من مؤلف واحد لكتاب؟ أو هل من الضروري وجود مؤلف؟

الجواب على ذلك أنه يمكن التحقق من صحة مستندات XML .بمقارنتها بمستندات أخرى تصف بنية نوع محدد من مستندات XML يوجد ثلاثة أنواع من هذه المستندات الوصفية:

1- مستند DTD طريقة قديمة وموروثة من SGML لوصف محتويات مستند XML.

2- المخططات Schemas آلية جديدة لوصف المعطيات.

3- مخططات XDR وهي آلية جديدة خاصة بمايكروسوفت.

جميع هذه الآليات الثلاثة تسمح بوضع شروط لمحتويات مستندات XML وذلك بتحديد:

- العلامات Tags التي يمكن وضعها في المستند.
- إذا ما كانت العلامات اختيارية أم لا.
- إذا كان من الممكن (ظهور العلامات أكثر من مرة).
- ترتيب ورود هذه العلامات Tags.
- طريقة تداخل العلامات.

كل هذا يتم في .NET. بواسطة الصنف System.Xml.XmlValidatingReader.

معالجة XML

إن تفسير XML في عالم Windows سهل جداً لأن مفسر XML الخاص بمايكروسوفت وهو جزء من Internet Explorer أو من أي متصفح ويب آخر، وبالتالي فهو موجود في كل جهاز في العالم.

هناك طريقتان شائعتان للتعامل مع مستندات XML عند استخدام مفسر، الأولى هي أن يقرأ المفسر كامل المستند ويفسره ويبني شجرة في الذاكرة، حالما تبنى الشجرة تستطيع عبورها حسب الحاجة وتعديلها (إضافة وحذف وتعديل العناصر)، أما الطريقة الثانية فهي تتم بقراءة المستند سطراً تلو الأخر وتمييز الأوامر عند ورودها.

وضعت مجموعة W3C نموذجاً لكيفية تمثيل مستندXML في الذاكرة وسمته Document Object Model اختصاراً DOM[1]، يمكن العمل على النموذج DOM في .NET. باستخدام الصف System.Xml.XmlDocument.

نموذج DOM مرن جداً ولكنه يعاني من مشكلة أن حجم الذاكرة التي يحتاجها لتخزين شجرة متناسب طرداً مع حجم المستند مما يمنع المفسرات من تفسير المستندات الكبيرة، وقد لا نحتاج في بعض الحالات إلى وجود البنية كاملة في الذاكرة لحظة واحدة !.

[1] للمزيد يمكن الإطلاع على الرابط التالي:http://www.w3schools.com/Dom/

تحقق العديد من المفسرات طرقاً لتفسير مستندات XML بشكل بسيط وفعال وللأمام فقط.
أحد المعايير القياسية الشهيرة هي هو SAX (Simple API for XML)[1]، حيث يقرأ المفسر مستند XML عنصراً تلو الأخر ويستخدم إجراءات الاستدعاء الخلفي التي تستخدمها أنت.
قدمت شركة Microsoft آلية تفسير أمامية فقط و سريعة وللقراءة فقط وسمتها بنموذج السحبmodel Pull، حيث تطلب من المفسر العنصر التالي عندما تكون جاهز وتتجاوز العناصر التي لا تهمك، يمكن العمل مع هذا النموذج من خلال الصنفين :System.Xml.XmlTextReader و System.Xml.XmlTextWrite.

تحويلات XSL (XML Style Sheet Language)

تصف العلاماتTags في مستند XML البيانات، لكنها لا تقدم أي معلومات عن طرق تمثيلها عند عرضها في مستعرض، تُستخدم XML لتخزين المعطيات ونقلها ولكنها ليست ذات فائدة كبيرة إلا إذا أمكن تحويلها إلى أشكال مفيدة أخرى مثل HTML لعرضها من قبل مستعرض أو PDF للطباعة أو حتى كدخل لعمليات تحديث قواعد المعطيات.
يمكن تحويل مستندات XML من شكل لأخر بواسطة لغة أوراق تنسيق XML (XML Style Sheet Language)، ذلك يشبه CSS مع HTML، كما يمكن استخدام XSL بطرق عديدة منها:

- تحويل XML إلى HTML لعرضها في مستعرض أو إلى PDF.
- تحويل XML إلى مجموعات مختلفة في XML لعرضها على أجهزة مختلفة.

ليكن لدينا العنصر التالي:

```
<fullname>Margaret Peacock</fullname>
```

قد نرغب بإخراج هذا الاسم كعنوان من المستوى 2 في HTML كمايلي:

```
<h2> Margaret Peacock</h2>
```

نستطيع عمل ذلك في XSL كما يلي:

```
<xsl:template match="employees/employee">
  <h2> <xsl:value-of select="."/> </h2>
</xsl:template>
```

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://en.wikipedia.org/wiki/Simple_API_for_XML

الصنف XmlTextReader

هو عبارة عن قارئ يؤمن وصولاً سريعاً للأمام فقط لمعطيات xml، يقرأ XmlTextReader مستند xml عنصراً تلو الأخر وذلك عوضاً عن تحميل المستند بأكمله كما في DOM، تعكس خصائص كائن XmlTextReader خصائص العقدة الحالية، ولا يمكن العودة إلى عقدة بعد قراءتها إلا بالرجوع إلى بداية المستند هذا يعني أن XmlTextReader يستهلك الذاكرة بالحد الأدنى، لأنه يحتاج إلى تخزين عقدة واحدة فقط في الذاكرة.
سنقوم الآن بسرد أهم الخصائص الموجودة في الصف XmlTextReader:

الخاصية	الوصف
AttributeCount	تعيد عدد الواصفات Attributes في العقدة الحالية(تذكر أن كائنXmlTextReader يمثل العقدة الحالية التي تمت قراءتها من ملف xml)
Depth	تدل على عمق العقدة الحالية في مكدس العناصر
Encoding	تدل على طريقة تشفير المستند
EOF	تعيد Trueإذا كان القارئ في نهاية مجرى الدخل
HasValue	تعيد True إذا كان للعقدة نص
IsEmptyElement	تعيد True إذا كانت العقدة فارغة مثل <empty/>
Item	تعدل قيمة سمة
LineNumber,LinePosition	تدل على رقم السطر الحالي وموقع الحرف الحالي
LocalName	اسم العقدة الحالية من دون فضاء أسماء
Name	اسم العقدة الحالية مع فضاء أسماء
Namespace	قراءة أو كتابة قيمة تدل على إمكانية فضاء الأسماء
NodeType	تعيد نوع العقدة الحالية
Prefix	تعيد حالة مجرى الدخل
Value	تعيد القيمة النصية للعقدة الحالية

أهم طرق Methods الصف XmlTextReader:

الطريقة Method	الوصف
Close	تغلق مجرى الدخل
GetAttribute	تعيد قيمة سمة أو صفة Attribute
MoveToAttribute	تنتقل إلى سمة محددة
MoveToElement	تنتقل إلى العنصر الذي يحتوي على السمة الحالية
MoveToFirstAttribute	تنتقل إلى السمة الأولى
MoveToNextAttribute	تنتقل إلى السمة الأخيرة
Read	تقرأ العقدة التالية من المجرى
ReadAttribueValue	تعيدالقيمةأوالقيم المقترنة بسمة
ReadBase64	تقرأ المعطيات النصية وتزيل الشيفرة بطريقة Base64
ReadBinHex	تقرأ المعطيات النصية وتزيل الشيفرة بطريقةBinHex
ReadVhars	تقرأ محتويات العنصر النصية وتضعها في Buffer
ReadInnerxml	تقرأ جميع محتويات العقدة الحالية بما في ذلك علامات Xml
ReadOutterxml	تقرأ جميع محتويات العقدة الحاليةوجميع أبنائها
ReadString	تقرأ محتويات العنصر النصية على شكل سلسلة نصية

الجدير بالذكر أن الصنف XmlTextReader مشتق من الصنف المجرد XmlReader والذي يؤمن وظائف القراءة الأساسية لثلاثة صفوف هي:
XmlTextReader و XmlValidationReader و XmNodeReader.

XmlValidationReader

يستخدم هذا الصنف للتحقق من صحة XML أثناء قراءتها، أنواع التحقق المتوفرة بشكل عام:

- تعاريف نوع المستند(DTDs).
- مخططات W3C القياسية.
- مخططات XDR لمايكروسوفت[1].

يحتوي الصنف XmlValidationReader خاصية اسمها ValidationType تحدد نوع التحقق من الصحة الذي يستخدم (واحد من الأنواع السابقة).
عندما يكتشف XmlValidationReader خطأ فأنه يقدح (Fire) حدثاً يحتوي معلومات عن الخطأ الذي حدث، يجب أن نمسك هذا الخطأ ونعالجه كما سنرى لاحقاً.

[1] للمزيد يمكن الإطلاع على الرابط التالي: https://en.wikipedia.org/wiki/XDR_Schema

XmlTextWriter

إذا أردت كتابة XML إلى ملف أو مجرى Stream بشكل متسلسل وكنت تعرف ماذا تريد فأن XmlTextWriter يقدم طريقة سريعة وللأمام فقط لكتابةXML.
يرث هذا الصنف من الصنف المجرد XmlWriter.
لنرى الآن أهم الخصائص والطرق في الصنف XmlTextWriter:

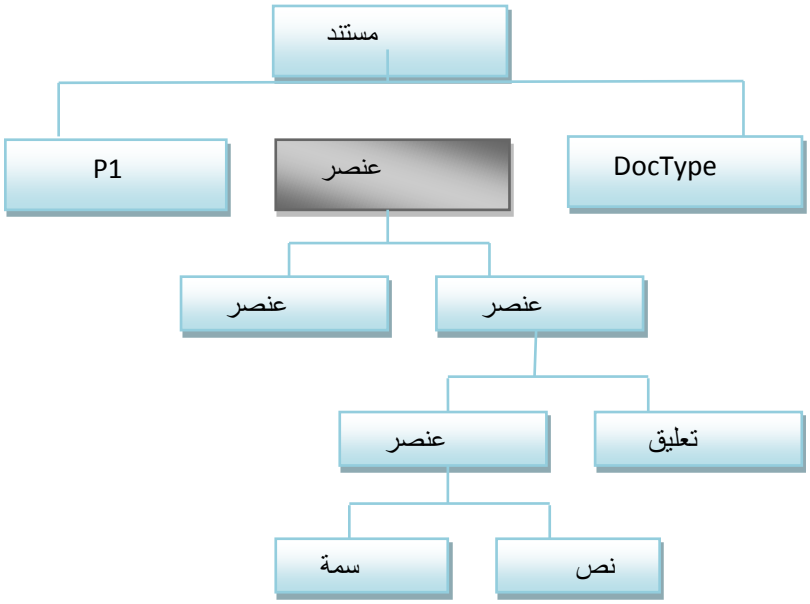
الخاصية	الوصف
Formatting	تحدد كيفية تنسيق الخرج
Indentation	تقرأ أو تكتب مستوى المسافة البادئة
indentChar	تقرأ أو تكتب الرمز المستخدم للمسافة البادئة
QuoteChar	تقرأ أو تكتب الرمز المستخدم للإحاطة بقيم السمات.
WriteState	تعيد حالات المجرى

أهم طرق Methods الصف XmlTextWriter:

الطريقة Method	الوصف
Close	تغلق مجرى الخرج
Flush	تفرغ مجرى الخرج
WriteBase64	تكتب بيانات مشفرة بطريقة Base64
WriteBinHex	تكتب بيانات مشفرة بطريقة BinHex
WriteChar,WriteChars	تكتب رمز واحد أو أكثر
WriteComment	تكتب تعليق XML
WriteDocType	تكتب تصريح DocType
WriteEndElement	تغلق أي عنصر أو أوسمة مفتوحة وتعيد الكاتب إلى وضعية البداية
WriteEndElement	تغلق العنصر الحالي (تكتب علامة نهاية)
WriteFullEndElement	تغلق العنصر الحالي وتكتب دائماً علامة نهاية كاملة
WriteName	تكتب اسماً
WriteStartAttribute	تبدأ سمة
writeEndAttribute	تنهي سمة
WriteStartDocument	تكتب تصريح بداية xml
WriteStartelement	تكتب علامة بداية
WriteString	تكتب قيمة نصية

XmlDocument

يمثل الصنف XmlDocument مستند xml بكامله على شكل شجرة DOM، يمكن استخدامه لإضافة أو حذف أو تعديل عقد ضمن الشجرة، لنرى بنية مستند xml:



لاحظ أن السمات والمحتويات النصية للعناصر عبارة عن عناصر مستقلة، المستند نفسه يمثل بعقدة مستند تسمى عادة بعنصر المستند، يوجد تحته عقدة عنصر تمثل جذر مستند XML (المظللة باللون الأسود)، الجدول التالي يسرد أنواع العقد في مستند xml:

الصنف Class	الوصف
XmlAttribute	يمثل سمةAttribute في عقدة
XmlCDataSection	يمثل مقطع CDATA
XmlComment	يمثل تعليقاً
XmlDeclaration	يمثل تصريح XML
XmlDocumentType	يمثل تصريح DocType
XmlElement	يمثل عنصراً

XmlNode

جميع أنواع العقد المذكورة في الفقرة السابقة مشتقة أصلاً من الصف المجرد XmlNode، يقدم هذا الصنف معظم الوظائف التي تحتاجها عند العمل مع أشجار DOM.
أيضاً الصف XmlDocument هو مشتق من الصفXmlNode وبالتالي فأنه يحتوي على جميع الطرق والخصائص الموجودة في العقدة، سنبدأ بشرح طرق وخصائص الصنف XmlNode:

خصائص هامة للصنف XmlNode

الخاصية	الوصف
Attributes	تعيد كائن من النوع XmlAttributeCollection يحوي جميع سمات هذه العقدة
ChildNodes	تعيد كائن XmlNodeList يحوي جميع أبناء هذه العقدة
FirstChild	تعيد الابن الأول لهذه العقدة في حال كان لها أبناء
HasChildNodes	تعيد True إذا كان للعقدة أبناء
Item	تسترجع عقدة ابن محدد
LastChild	تعيد الابن الأخير لهذه العقدة
NodeType	تعيد نوع العقدة
OwnerDocument	تعيد كائن XmlDocument التي تنتمي إليه هذه العقدة
ParentNode	تعيد أب هذه العقدة
Value	تكتب أو تقرأ قيمة هذه العقدة

طرق هامة للصنف XmlNode

الطريقة Method	الوصف
AppendChild,PrependChild	تضيف عقدة إلى نهاية أو بداية لائحة أبناء هذه العقدة
Clone	تستنسخ هذه العقدة
CreateNavigator	إنشاء مُبحر للعما مع هذه العقدة وأبنائها
GetEnumerator	تعيد مُعدد Enumerator لمجموعة العقدة
InsertBefore,InsertAfter	تدخل عقدة قبل أو بعد عقدة أخرى
WriteContentTo	كتابة محتويات العقدة إلى xmlwriter

أعضاء XmlDocument

يحوي الصنف XmlDocument عدداً من الطرق والخصائص وسنراها بعد قليل.
تستطيع أن ترى أن عدداً من الطرق تنشئ مختلف أنواع العقد التي يمكن وجودها ضمن شجرة DOM.

خصائص هامة للصنف XmlDocument

الخاصية	الوصف
DocumentElement	تعيد عنصر المستند لهذه الشجرة (الجذر)
DocumentType	تعيد معلومات DocType عن هذا المستند
Name,LocalName	اسم العقدة الحالية مع أو بدون فضاء أسماء
NodeType	تعيد نوع العقدة الحالية

طرق هامة للصنف XmlDocument

الطريقة Method	الوصف
CreateAttribute	تنشئ كائن XmlAttribute لتمثيل سمة
CreateElement	تنشئ XmlElement
CreateNode	تنشئ عقدة من نوع محدد
Load	تحمل Xml من Stream أو Reader
LoadXml	تحمل Xml من سلسلة نصية
Save	تخزين Xml إلى ملف أو مجرى معين

XSL و XPath

يتوفر دعم XSL و XPath في فضائي الأسماء System.Xml.Xsl و System.Xml.XPath، يحتوي فضاء الأسماء XPath مفسر ومحرك تقييم (Validation)، إلا أن العضو الأكثر فائدة في فضاء الأسماء هذا هو XPathNavigator والذي يؤمن طريقة مفيدة للإبحار عبر المستند.
يقدم الصنف System.Xml.Xsl.XslTransform تحويلات Xsl ويحتوي فقط على طريقتين (two Method) هما:

1- Load والتي تحمل ورقة تنسيق XSL إلى كائن المعالجة.
2- Transform والتي تستخدم لإنجاز التحويل.

تتم معالجة المستندات كما يلي:

1- إنشاء كائن XmlTransform.
2-استخدام الطريقة Load لتحميل ورقة تنسيق إلى الكائن.
3- تحميل معطيات XML إلى XmlDocument ثم تغليف الأخير ضمن XmlNavigator.
4- استخدم الطريقة Transform لتحويل XML.

XPathNavigator

يقدم الصنف System.Xml.XPath.XPathNavigator طريقة لقراءة المعطيات من مخزن معطيات باستخدام نموذج المشيرة (Cursor Model)[1]، استخدام المشيرة يعني قراءة عنصر واحد كل مرة من جدول أو مستند بحيث لا يمكنك رؤية كامل مجموعة المعطيات بل ترى فقط العنصر الموجود تحت المشيرة.
يمكن أن تكون مجموعة المعطيات مستندات XML أو كائنات DataSet في ADO.NET، تؤمن الطريقة MoveTO في هذا الصنف وصولاً عشوائياً للقراءة فقط إلى المعطيات، وتعكس خصائص الصنف خصائص العقدة الحالية في مجموعة المعطيات.

خصائص هامة للصنف XPathNavigator:

الخاصية	الوصف
HasAttributes	تكون True إذا كان للعقدة الحالية سمات
HasChildren	تكون True إذا كان للعقدة الحالية أبناء
IsEmptyElement	تكون True إذا لم يكن للعقدة الحالية محتويات
LocalName	تعيد اسم العقدة الحالية بدون فضاء اسماء
Name	تعيد الاسم الكامل للعقدة الحالية
NodeType	تعيد نوع العقدة الحالية
Value	تقرأ أو تكتب القيمة المرتبطة بالعقدة الحالية

أنواع العقد الشائعة المعروفة في التعداد XPathNodeType (Enum):

قيمة التعداد	الوصف
Attribute	سمة XML مثل "name="fred
Comment	تعليق XML
Element	عقدة عنصر
NameSpace	عقدة فضاء أسماء
Root	جذر شجرة العقدة
Text	المحتويات النصية للعنصر

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://msdn.microsoft.com/en-us/library/ms191179.aspx

طرق مهمة للصنف XPathNavigator:

الطريقة	الوصف
Compile	تترجم تعبير XPath للاستخدام المستقبلي
Evaluate	تقييم تعبير XPath
GetAttribute	تعيد قيمة السمة المحددة في العقدة الحالية
Matches	تحدد فيما إذا كانت العقدة الحالية تطابق تعبير XPath مُعطى
MoveTo	تنتقل إلى نفس الموقع الذي يشير إليه كائن أخر .
Select	تحدد مجموعة من العقد باستخدام تعبير XPath

سننتقل الآن لمجموعة من الأمثلة لتغطية كافة المواضيع التي تم الحديث عنها.

تفسير مستند XML باستخدام XmlTextReader

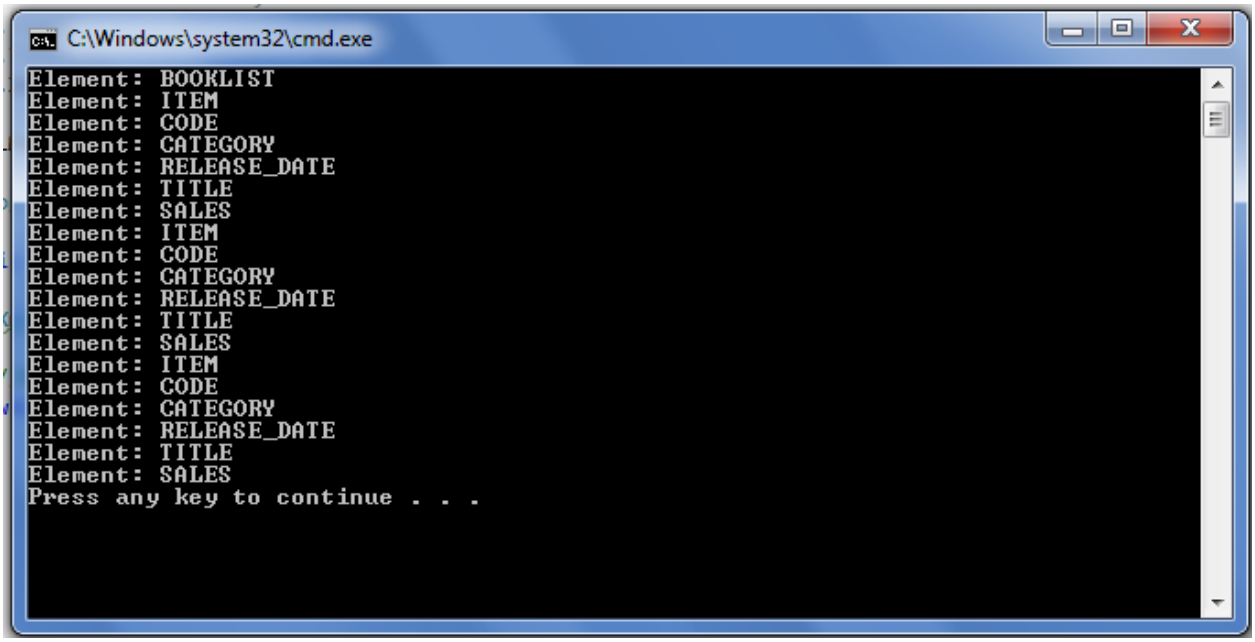
يقدم الصنف XmlTextReader طريقة بسيطة لقراءة(تفسير) مستند XML بسرعة وكفاءة عالية من حيث استخدام الذاكرة، تشبه هذه الطريقة من حيث المبدأ SAX وهي معيار شهير يستخدم في مفسرات XML، ولكنها تختلف عنها في أن SAX يستخدم نموذج الدفع Push (يقوم المفسر باستدعاء شيفرة برنامجك عندما يجد عقدة)، بينما يستخدم XmlTextReader نموذج السحب Pull أي أنه يتم الحصول على العقدة التالية والحصول على معلوماتها عندما تكون (أنت) جاهزاً لذلك. وهكذا نرى أن استخدام XmlTextReader يعطيك تحكماً أكثر بعملية القراءة.
لنرى المستند التالي، الأمثلة القادمة ستستخدم هذا المستند:

```
<?xml version="1.0"?>
<BOOKLIST>
  <ITEM isbn="12323" topic="C#">
    <CODE>16-041</CODE>
    <CATEGORY>HTML</CATEGORY>
    <RELEASE_DATE>1998-03-07</RELEASE_DATE>
    <TITLE>Instant HTML</TITLE>
    <SALES>127853</SALES>
  </ITEM>
  <ITEM isbn="532341" tpoic="JAVA">
    <CODE>16-048</CODE>
    <CATEGORY>Scripting</CATEGORY>
    <RELEASE_DATE>1998-04-21</RELEASE_DATE>
    <TITLE>Instant JavaScript</TITLE>
    <SALES>375298</SALES>
  </ITEM>
  <ITEM isbn="232413" topic="Python">
    <CODE>16-105</CODE>
    <CATEGORY>ASP</CATEGORY>
    <RELEASE_DATE>1998-05-10</RELEASE_DATE>
    <TITLE>Instant Active Server Pages</TITLE>
    <SALES>297311</SALES>
  </ITEM>
</BOOKLIST>
```

لنشاهد الكود التالي:

```
using System.Xml;
static void Main(string[] args)
{
    XmlTextReader xtr = new XmlTextReader(@"D:\booklist.xml");
     //Read the Next Node from Stream and Retrurn bool if the Stream is ended
     while (xtr.Read())
        // if The Current Node is Element
        if (xtr.NodeType == XmlNodeType.Element)// XmlNodeType is Enum
              Console.WriteLine("Element: " + xtr.Name);
}
```

يقوم هذا الكود بقراءة ملف Xml بطريقة السحب Pull (عقدة تلو الأخرى حيث أن الصف XmlTextReader يمثل العقدة الحالية في الذاكرة)، ثم طباعة جميع الأوسمة Tage الموجودة بداخله.

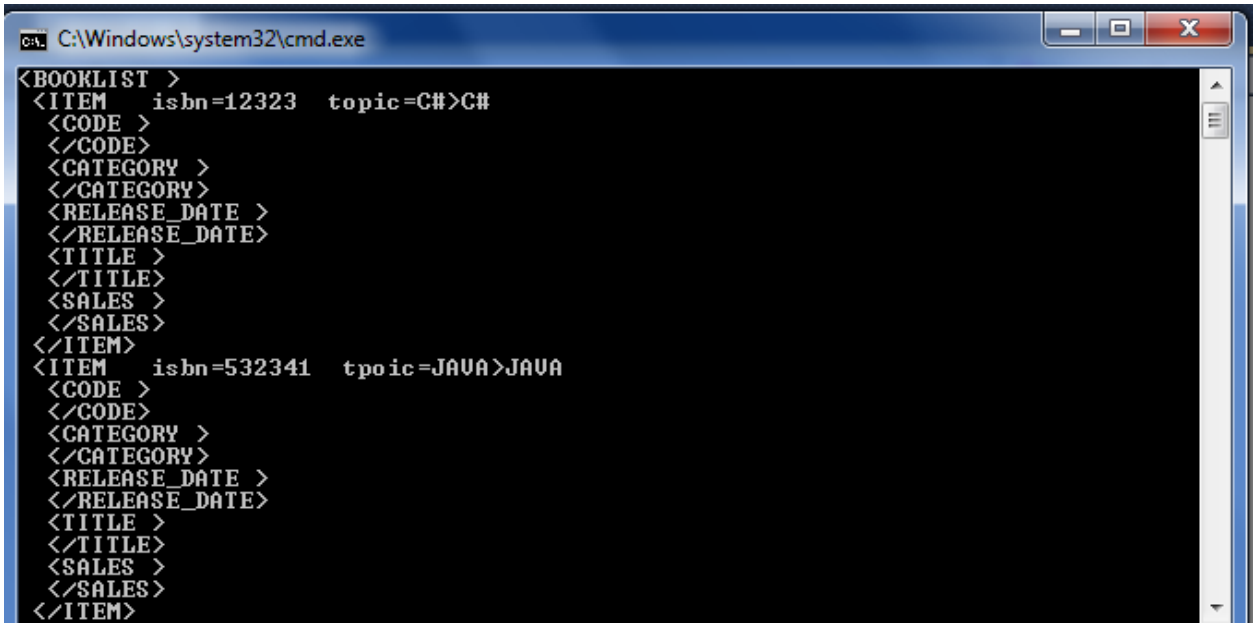


لنشاهد الكود التالي:

```
 //To Format Printed Text the Represent XML File
static string shift(int n)
{
       string s = "";
        if (n == 0)
            return " ";
        for (int i = 0; i < n - 1; i++)
            s += " ";
        return s;
}
```

```
static void Main(string[] args)
{
   XmlTextReader  xtr = new XmlTextReader(@"D:\booklist.xml");

  int level = 0;

  string istr="";

  //Read the Next Node from Stream and Retrurn bool if the Stream is ended
  while (xtr.Read())
      // if The Current Node is Element that means (<)
      if (xtr.NodeType == XmlNodeType.Element)// XmlNodeType is Enum
      {
          istr = shift(level);
          Console.Write(istr+"<"+xtr.Name+" ");
          if (xtr.AttributeCount > 0)//you have an Attributes
            while (xtr.MoveToNextAttribute())//Move to next Attribute
                 Console.Write("  "+ xtr.Name + "="+xtr.Value);//Attribute=value
          Console.WriteLine(">");

          level++;
       }
       else if (xtr.NodeType == XmlNodeType.EndElement)//if xtr.NodeType==">"
       {
           level--;
           istr = shift(level);
           Console.WriteLine(istr + "</" + xtr.Name + ">");
       }
 }
```

ويكون الخرج كما يلي:



تفسير(قراءة) مستند XML مع التحقق من الصحة

لا يقوم XmlTextReader بالتحقق من صحة XML عند قراءتها، إذا أردنا ذلك يجب علينا استخدام الصنف XmlValidationReader.

نقصد بالتحقق من الصحة أنه لو كان لدينا ملف XML ونريد أن نتحقق من أنه يحقق شكل معين من الهرمية (مثل قواعد المعطيات، مثلاً: أن تكون عقدة العنوان مؤلفة من عقدتين هما اسم المدينة والشارع) عند ذلك نضع في بدية ملف XML تعليمات خاصة بالتحقق من الصحة تقوم بتوصيف بنية المستند وعناصره (طبعاً لكل نوع Syntax معين للكتابة)، سنستخدم DTD ونفس الكلام يطبق على غيرها من الأنواع الأخرى.

يعتمد نوع التحقق من الصحة الذي ينجزه المفسر على الإعدادات المخزنة في الخاصية ValiationType والتي لها القيم التالية:

قيمة التعداد (Enum)	الوصف
Auto	يتحقق القارئ من الصحة تبعاً لمعلومات التحقق الموجودة بداخل المستند
DTD	يتحقق القارئ من الصحة مستخدماً DTD
None	لا يقوم القارئ بالتحقق من الصحة
Schema	يتحقق القارئ من الصحة مستخدماً مخطط W3C القياسي
XDR	يتحقق القارئ من الصحة مستخدماً مخطط XDR الخاص بمايكروسوفت

يقوم القارئ XmlValidationReader بقدح (Fire) لحدث معين في حال وجود خطأ في التحقق لهذا المستند أثناء تفسيره، لذلك من الضروري إضافة معالج للحدث وربطه بتابع كما نقوم في كافة الأحداث.لنتأمل الكود التالي بتمعن:

```
 using System.Xml;
using System.Xml.Schema;
namespace XMLManipulating
{
    class Program
    {
        private static bool success = true;

        static  void validate(string filename)
        {
            success = true;

            XmlTextReader xtr = new XmlTextReader(@"D:\booklist.xml");

            XmlValidatingReader xvr = new XmlValidatingReader(xtr);

            xvr.ValidationType = ValidationType.DTD;


          // Set the validation event handler;
          // When Error is Happend the method will be Called
            xvr.ValidationEventHandler += new
            ValidationEventHandler(xvr_ValidationEventHandler);
```

```
            //Reading The Data
            while(xvr.Read()){  }

            Console.WriteLine("Validation Finished. Validataion{0}"
                  ,success==true?"Successed":"Failed");

            xvr.Close();
        }


   //Display the validation error
 static void xvr_ValidationEventHandler(object sender, ValidationEventArgs e)
  {
      success = false;
      Console.WriteLine("Validation Error: " + e.Message);
  }


 static void Main(string[] args)
 {
      validate(@"D:\XMLFile4Validation.xml");
 }

    }//End of Program Class
}//End of NameSpace
```

قمنا في البداية بإنشاء كائن XmlTextReader ثم غلفناه بكائن أخر XmlValidationReader ثم حددنا نوع التحقق (في الحقيقة يوجد مشاكل عند العمل مع النوع DTD لأن VS.NET يحول DTD إلى XSD ولكن ما يهمنا هو فهم الفكرة).

الآن لنشاهد مثال الإختبار وهو ملف XML يحوي DTD مضمنة في بدايته:

```
<?xml version="1.0" encoding="utf-8"?>
<!--Inline DTD schema-->
<!DOCTYPE invoice[
  <!ELEMENT invoice (Customer,Adderess)>
  <!ELEMENT Customer (#PCDATA)>
  <!ELEMENT Address (Street,Town)>
  <!ELEMENT Street (#PCDATA)>
  <!ELEMENT Town (#PCDATA)>
]>
<!-- End of DTD schema-->
<invoice>
  <Customer>Hammod </Customer>
  <Address>Jabadeen
    <Street> Yarmook </Street>
    <City>Damascus </City>
  </Address>
</invoice>
```

تنص معلومات DTD على أن لكل فاتورة (invoice) زبون و عنوان ولكل عنوان شارع ومدينة. في حال خالفنا القواعد يقوم المترجم بإصدار خطأ ثم استدعاء التابع المربوط بالحدث.

كتابة مستندات XML باستخدامXmlTextWriter

يزودنا الصنف XmlTextWriter بمجموعة أدوات لكتابة XML بشكلها المتسلسل كاملة مع الأقواس المنكسرة (<>) وتعليمات المعالجة (تعليمات خاصة بالمفسر) وجميع العناصر التي يمكن أن نراها في مستندات XML.
لنرى الكود التالي:

```
static void Main(string[] args)
{
     XmlTextWriter xtw = new XmlTextWriter(@"D:\xmlfile1.xml", null);
     //Generating the output  formatted
     xtw.Formatting = Formatting.Indented;

      //Write XML Declaration
       xtw.WriteStartDocument(true);
       xtw.WriteStartElement("Books"); //Write <Books>
          xtw.WriteAttributeString(@"ISBN", "1-123-12345");
          xtw.WriteStartElement("title"); //Write <title>
             xtw.WriteString("Moby Dick");
          xtw.WriteEndElement(); //Write </title>
       xtw.WriteEndElement(); //Write </Books>

       xtw.Flush(); // this statment is Very Important

       xtw.Close();
 }
```

ويكون الخرج بداخل ملف XML كمايلي:

```
<?xml version="1.0" standalone="yes"?>
<Books ISBN="1-123-12345">
  <title>Moby Dick</title>
</Books>
```

استخدام XPathNavigator

يقدم الصنف XPathNavigator طريقة أخرى لقراءة وتعديل ملفات XML وهي تشبه طريقة الصنف XmlTextReader في أنها تستخدم نموذج المشيرة، أي أن هذا الكائن يشير إلى عقدة واحدة في الشجرة ويعكس هذا الكائن خصائص العقدة الحالية، في الحقيقة أن الصنف XPathNavigator هو صنف مجرد لذلك يوجد صفوف أخرى مثل: XmlDocument,XmlDataDocument,XPathDocument تقدم طرقاً لإنشاء كائنات إبحار Navigation عبر ملفات XML.

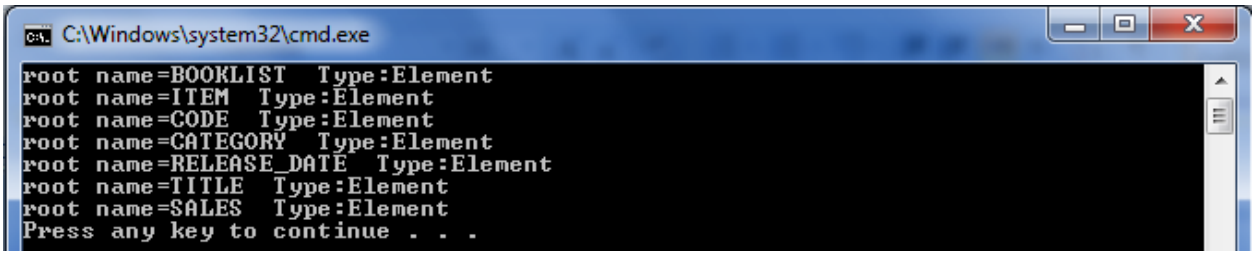
يتميز XPathNavigator عن XmlTextReader في أنك غير مقيد بالقراءة إلى الأمام عبرالشجرة (يمكن التحرك إلى الأمام وإلى الخلف وهذا ما لا يمكن فعله في XmlTextReader).

يقوم البرنامج التالي بطباعة جميع العقد الموجودة داخل العقدة الأولى (يستخدم ملف XML الذي أوردناه في البداية).

```
static void Main(string[] args)
{
    XmlDocument doc = new XmlDocument();

     doc.Load(@"D:\booklist.xml");

     XPathNavigator nav = doc.CreateNavigator();

        //Move Navigator to the Root of the DOM Tree this Node is Empty
       //and Refer to the xmlDeclaration and first Tag(Node) in the XML File
      nav.MoveToRoot();

      //Move Navigator to The first Child(Node or Tag) in the DOM Tree is<BOOKLIST>
      nav.MoveToFirstChild();
    Console.WriteLine("name=" + nav.Name + " Type:" + nav.NodeType );

      //Move Navigator to The first Child(Node or Tag) in the DOM Tree is <ITEM>
      nav.MoveToFirstChild();
      Console.WriteLine(" name=" + nav.Name + "  Type:" + av.NodeType);

      //Move Navigator to The first Child(Node or Tag) in the DOM Tree  is <CODE>
      nav.MoveToFirstChild();
      Console.WriteLine(" name=" + nav.Name + "  Type:" + av.NodeType);

     while (nav.MoveToNext())
      Console.WriteLine(" name=" + nav.Name + "Type:" +nav.NodeType);
}
```

في البداية قمنا بإنشاء كائن XmlDocument وقمنا بتحميله ضمن الذاكرة على شكل شجرة DOM، ثم قمنا بالحصول على مُبحر Navigator باستخدام الطريقة CreateNavigator، ثم قمنا بالانتقال إلى جذر الشجرة باستخدام الطريقة MovetoRoot والتي تأخذ مؤشر المُبحر إلى بدية الملف(حالياً المُبحر لا يشير إلى أية عقدة)، وللانتقال إلى أول عقدة في الشجرة (أو في ملف الXML) نستدعي الطريقةMovetoFirstchild والتي تضع المُبحر عند العقدة الأولى وهي <BOOKLIST>، ثم نقوم بالتحرك بنفس الطريقة (نقوم باستدعاء الطريقة السابقة)

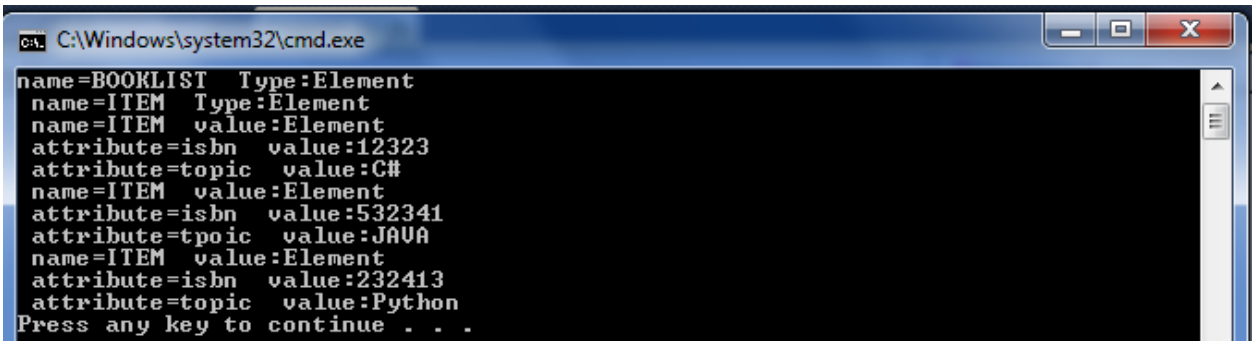
مرة أخرى من أجل الوصول إلى العقدة <ITEM> ومن ثم نكرر ماسبق للوصول إلى العقدة <CODE> ثم عن طريق حلقة While نطبع العقد الباقية عن طريق استدعاء الطريقة MovetoNext وبذلك يكون الخرج كما يلي:



يتم التعامل مع السمات Attribute بنفس الطريقة، حيث نستخدم الطريقة MoveToNextAttribute() ضمن حلقة While وذلك بعد التأكد من وجود سمات ضمن العقدة الحالية وذلك باستخدام الطريقة MoveTofirstAttrubute، لنشاهد الشيفرة التالية:

```
static void Main(string[] args)
{
     XmlDocument doc = new XmlDocument();

      doc.Load(@"D:\booklist.xml");

      XPathNavigator nav = doc.CreateNavigator();

     //Move Navigator to the Root of the DOM Tree this Node is Empty
     //and Refer to the xmlDeclaration and first Tag(Node) in the XML File

     nav.MoveToRoot();

     //Move Navigator to The first Child(Node or Tag) in the DOM Tree is <BOOKLIST>
     nav.MoveToFirstChild();
     Console.WriteLine("name=" + nav.Name + "  Type:" + nav.NodeType );

    //Move Navigator to The first Child(Node or Tag) in the DOM Tree is <ITEM>
    nav.MoveToFirstChild();
    Console.WriteLine(" name=" + nav.Name + "  Type:" + nav.NodeType);

     do
    {
          Console.WriteLine(" name=" + nav.Name + "  value:" + nav.NodeType);
          //Handle Attributes
         if (nav.MoveToFirstAttribute())
          {
                Console.WriteLine(" attribute=" + nav.Name + "  value:" + nav.Value);
                while (nav.MoveToNextAttribute())
                    Console.WriteLine(" attribute=" + nav.Name + " value:" + nav.Value);
           }
           nav.MoveToParent();//This Statement is Very importnat

     } while (nav.MoveToNext());
}
```

يقوم البرنامج السابق بطباعة كل عقدة بالإضافة إلى السمات الموجودة داخلها، لنرى الخرج الآن:

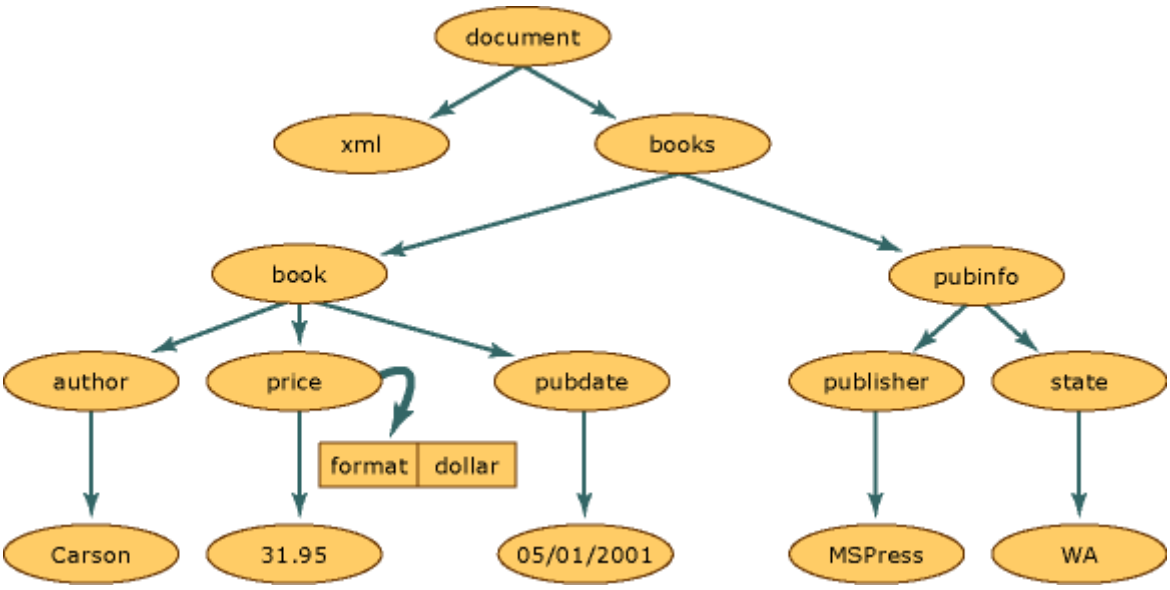


إنشاء واستخدام أشجار DOM باستخدام XmlDocument

يحقق الصنف XmlDocument نموذج W3C DOM للعمل مع مستندات XM في الذاكرة، يسمح هذا الصنف بالتعامل مع مستند XML (إضافة,حذف, تعديل).
لنفرض أنه لدينا الملف التالي:

```
<?xml version="1.0"?>
<books>
  <book>
    <author>Carson</author>
    <price format="dollar">31.95</price>
    <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
    <publisher>MSPress</publisher>
    <state>WA</state>
  </pubinfo>
</books>
```

عندئذٍ تكون شجرة DOM لهذا الملف كما يلي:



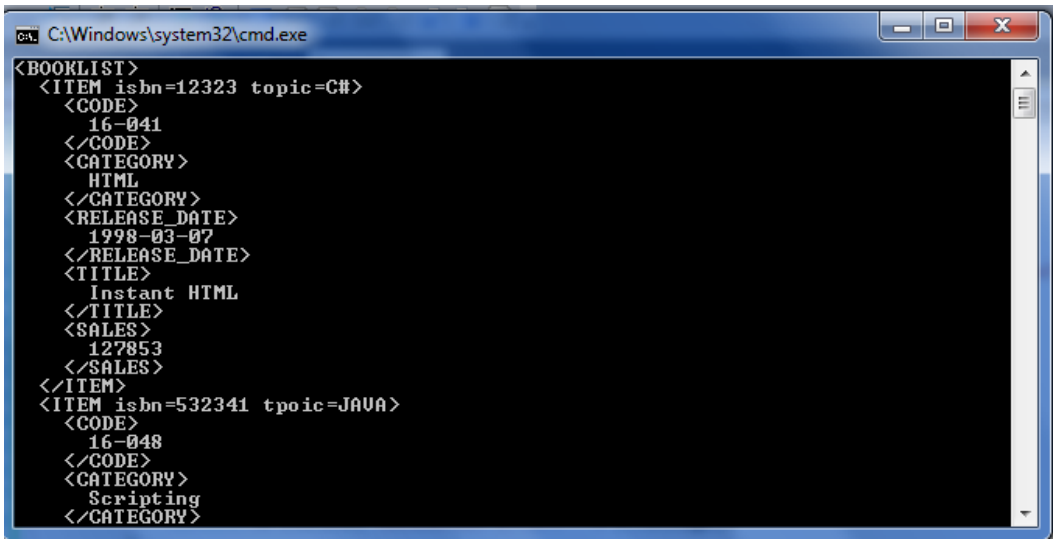
والآن لنرى الكود التالي بتمعن:

```
private static void ProcessChildren(ref XmlNode root, int level)
{
      string istr = "";
      istr = shift(level);

      switch(root.NodeType)
      {
           case XmlNodeType.Text:
                 Console.WriteLine(istr + root.Value);
                 break;

           case  XmlNodeType.Element:
              // Get All Child Nodes  of the Current Node
              XmlNodeList childs = root.ChildNodes;// Very Important

              Console.Write(istr + "<" + root.Name);

             //Handle Atttributes
              XmlAttributeCollection atts = root.Attributes;
              if(atts!=null)
              {
                   IEnumerator enumerator = atts.GetEnumerator();
                   while (enumerator.MoveNext())
```

```
                    {
                         // Casting is Important here
                       XmlNode attribute = (XmlNode) enumerator.Current;
                        onsole.Write(" " + attribute.Name + "=" +
                    attribute.Value);
                       }
                }
                Console.WriteLine(">");

                 //Recursively Process Child Nodes
                IEnumerator ie = childs.GetEnumerator();
                while(ie.MoveNext())
                {
                      XmlNode node = (XmlNode) ie.Current;

                      ProcessChildren(ref node, level + 2);
                }

               Console.WriteLine(istr+"</"+root.Name+">");

                break;
              }//End Switch

        }


//To Format Printed Text the Represent XML File
static  string shift(int n)
{
     string s = " ";
     if (n == 0)
         return "";
     for (int i = 0; i < n - 1; i++)
         s += " ";
     return s;
}


static void Main(string[] args)
{
    XmlDocument doc = new XmlDocument();

        //1-Loads XML File and Create DOM Tree in Memory
       //2- Check if it has an Error in XML File
        doc.Load(@"D:\booklist.xml");

        // Get root of XML Document
        XmlNode root = doc.DocumentElement;
        if (root.HasChildNodes)
```

```
            ProcessChildren(ref root, 0);
}
```

في البداية قمنا بإنشاء كائن XmlDocument، ثم قمنا بتحميل مستند XML وإنشاء شجرة DOM في الذاكرة عن طريق التابع Load الخاص بالصف XmlDocument، ثم قمنا بالحصول على جذر المستند عن طريق الخاصية DocumentElement ثم قمنا بفحص هذه العقدة (الجذر) فيما إذا كان لها أبناء أم لا وذلك عن طريق الخاصية HasChildNodes، في حال وجود عفد أبناء لهذه العقدة (الجذر) فأننا نستدعي التابع ProcessChildren والذي يفحص كل عقدة ويحضر جميع أبنائها ومن ثم يبحث على هؤلاء الأبناء وكل عنصر منهم يحضر أبنائه وهكذا بطريقة عودية يتم استكشاف مستند XML بسهولة ويسر.
لنرى الخرج الآن (طباعة مستند XML بجميع محتوياته):



إنشاء وتعديل العقد باستخدام XmlDocument

سنرى من خلال هذا المثال عدد من الصفوف التي يستخدمها الصنف XmlDocument لتحقيق عمله، والجدير بالذكر أنه عادةً لا يكون لهذه الصفوف بواني (Constructors) وإنما يمكن إنشاء كائن فقط عن طريق كائن المستند ويتم التقاطه بواسطة مرجع Reference من الصف الذي ليس له باني (في الحقيقة له باني لكنه Protected) كما في التعليمة التالية مثلاً:

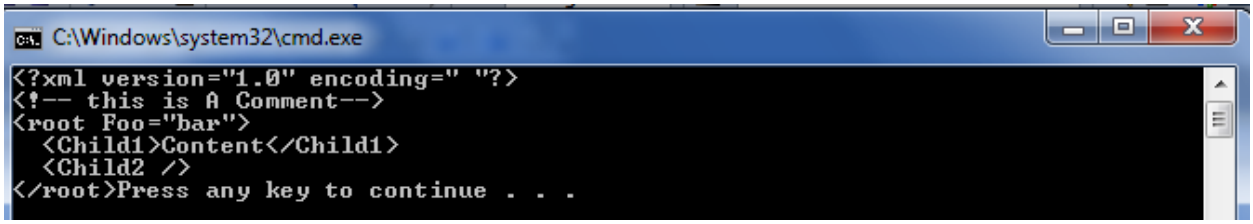
```
//Create XML Declaration and Put it in Object from XmlDeclaration Class
XmlDeclaration decl = doc.CreateXmlDeclaration("1.0"," ","");
```

لنرى الكود التالي بتمعن :

```
static void Main(string[] args)
{
     XmlDocument doc = new XmlDocument();

    //Create XML Declaration and Put it in Object from XmlDeclaration Class
    XmlDeclaration decl = doc.CreateXmlDeclaration("1.0"," ","");

   //Add  XML Declaration Object to doc

    doc.AppendChild(decl);

   //Add Comment
    XmlComment comment = doc.CreateComment(" this is A Comment");


   //Add Comment after  XML declatation
    doc.InsertAfter(comment, decl);

    // Add an Element
    XmlElement element = doc.CreateElement("root");

   //insert element(root) after comment
   doc.InsertAfter(element, comment);

   //Set an Attribute using XmlAttribute Class
   XmlAttribute att = doc.CreateAttribute("Foo"); //1
   att.Value = "bar";//2

   //set Attribute(att) to element
   element.SetAttributeNode(att); //3

   // you Can Replace Line 1,2,3 in this Statment:
   // element.SetAttribute("foo", "bar");

   //Add Some Children to element
   XmlElement ch1 = doc.CreateElement("Child1");
   XmlElement ch2 = doc.CreateElement("Child2");
```

```
    element.AppendChild(ch1);

    element.AppendChild(ch2);

    //Add Some Text
    XmlText tx1 = doc.CreateTextNode("Content");
    ch1.AppendChild(tx1);

    // Writeing DOM Tree To Screen
     XmlTextWriter xtw = new XmlTextWriter(Console.Out);

     xtw.Formatting = Formatting.Indented;

     doc.WriteTo(xtw);

     xtw.Flush();

     xtw.Close();
}
```

لنرى الخرج الآن على الشاشة:



استخدام الصنف XPath

يقدم XPath لغة لتحديد مجموعة من العقد ضمن مستند XML، مثل: "جميع الكتب التي لها أكثر من مؤلف" أو "الكتاب ذو السعر الأعلى"، XPath لغة خاصة للاستعلام من مستند XML وهي خارج إطار هذا الكتاب[1].
يستخدم الصنف XPathNavigator للعمل مع XPath في .NET. وذلك باستخدام الطريقة Select، إذ أنها تُستدعى مع تعبير XPath الذي يحدد مجموعة العقد التي تريد استخلاصها وتعيد لك مجموعة من العقد التي تقابل التعبير.
لنرى الكود التالي (يعيد عناوين الكتب من مستند XML):

```
static void Main(string[] args)
{
  XmlDocument doc = new XmlDocument();

    doc.Load(@"D:\booklist.xml");

    //Create Navigator at  XML Document
    XPathNavigator nav = doc.CreateNavigator();

    //Move navigator to Root of Documnet
    nav.MoveToRoot();

    //Select All Books
    XPathNodeIterator ni = nav.Select("//ITEM/TITLE");

    //Print number of Retrived Nodes
    Console.WriteLine("retrived " + ni.Count.ToString() + " Nodes");

    //Print Out their titles
    while (ni.MoveNext())
    {
         XPathNavigator nav2 = ni.Current; // now navigator at ITEM Nodes

         nav2.MoveToFirstChild(); //Move navigator to first child Node(title)

         Console.WriteLine("title: " + nav2.Value);
    }
}
```

قمنا في البداية كالعادة بإنشاء مستند XML ثم قمنا بإنشاء مُبحر XPathNavigator ثم نقلنا المُبحر إلى بداية المستند.

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://www.w3schools.com/xml/xml_xpath.asp

تأخذ الطريقة Select() تعبير XPath ثم تقوم بالتأكد من صحته (Validation).في هذه الحالة التعبير هو "ITEM/TITLE" والذي يطابق جميع عناصر <TITLE> الموجودة ضمن العنصر(العقدة <ITEM>، تعيد هذه الطريقة مرجعاً إلى XPathNodeIterator والذي يمثل مُعدد(Iterator) على مجموعة محددة من العقد. يدعم الصف XPathNodeIterator جميع الطرق والخصائص المعروفة بالنسبة للمُعدد (Iterator أو Enumerator) بما فيها MoveNext() و Current وتعيد الخاصية Current كائن من XPathNavigator للتعامل مع كل عقدة على حدا (الآن نحن عند العقدة ITEM) ثم ننتقل إلى العقدة TITLE بواسطة الطريقة MoveToFirstChild() ثم نطبع المطلوب وهكذا ضمن حلقة While.

تحويل XML إلى صيغ أخرى باستخدام XmlTransform

يبين المثال التالي كيفية استخدام ورقة تنسيق XSL مع كائن XmlTransform لتحويل مستند XML إلى شكل أخر (مُنسق).

في البداية يجب علينا كتابة ملف ورقة تنسيق XSL Spread Sheet وهي باسم Style.xsl:

```
<?xml version="1.0" encoding="utf-8"?>
 <xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">

    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="BOOKLIST">
      <html>
        <head> <title>List of Books </title> </head>
        <body>
          <ul>
            <xsl:for-each select="ITEM">
              <li>
                <xsl:value-of select="TITLE"/></li>
            </xsl:for-each>
          </ul>
        </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

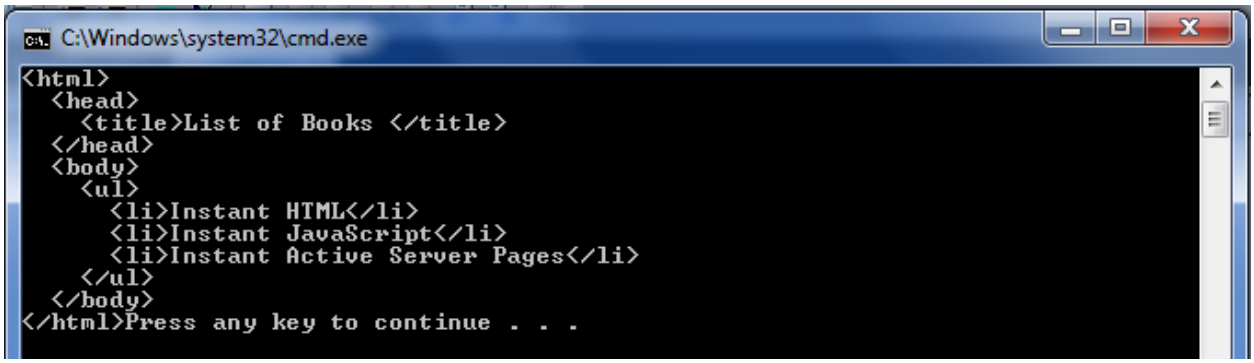
باختصار: يقوم هذا الملف بتعريف القواعد اللازمة والتي يجب على الصنف XmlTransform أن يفسرها ويفهمها، تقوم هذه الشيفرة بتعريف قالب يمكن وضع عناصر XML بداخله وهذه العناصر يتم استخلاصها بواسطة التعليمة for-each، والتي تطبق هذا القالب (مجموعة الأوسمة Tags التي قمنا بكتابتها بأنفسنا مثل <ul>) على كل عقدة ضمن المجموعة الحالية، وقد حددنا بداية المستند (الجذر) من خلال الخاصية match="BOOKLIST" ، ومن ثم نحصل على قيمة كل عقدة ضمن من خلال التعليمة value-of. كل هذه التعليمات موجودة ضمن فضاء الأسماء xsl. تقصدنا تسمية الأوسمة(Tags) التي عرفناها من خلال الأوسمة الموجودة في HTML وذلك غير ضروري (يمكن اختيار أسماء الأوسمة Tagsكما نريد) ولكننا قمنا بذلك لكي نسمح بتفسيرها من قبل مستعرض انترنت فقط.

الآن لنرى الشيفرة التالية:

```
using System.Xml;
using System.Xml.Schema;
using System.Xml.XPath;
using System.Xml.Xsl;


static void Main(string[] args)
{
    XmlDocument doc = new XmlDocument();

    doc.Load(@"D:\booklist.xml");

    //Create Navigator
     XPathNavigator nav = doc.CreateNavigator();

   //Move navigator to Root of Documnet
    nav.MoveToRoot();

   //Create XslTransform Object
   XslTransform xt = new XslTransform();

   //Loading the xsl Style Sheet file
    xt.Load(@"D:\Style.xsl");

   //Print the Result on Screen ,so we choose Console.Out
   XmlTextWriter writer = new XmlTextWriter(Console.Out);
  //We Can store Result on File at hardDisk by Choosing the overloads number2

   writer.Formatting = Formatting.Indented;

    //Do the Transform
    xt.Transform(nav, null, writer);
}
```

في البداية قمنا بإنشاء كائنXmlDocument ثم قمنا بتحميل مستند XML المطلوب، ومن ثم إنشاء مُبحر ونقلناه إلى بداية المستند، ثم قمنا بإنشاء كائن من الصنف XslTransform، ثم حَمّلنا ورقة التنسيق والتي قمنا بمشاهدتها قبل قليل، ثم أنشأنا كائن XmlTextWriter وحددنا وجهة الخرج (إلى الشاشة)، ثم قمنا بعملية التحويل بواسطة الطريقة Transform والتي تأخذ كائن المُبحر وتأخذ بارمترات عملية التحويل(في حالتنا نضع null) بالإضافة إلى كائن XmlTextWriter والذي سنضع فيه الخرج.

لنرى الخرج الآن في الحقيقة قمنا بتحويل xml إلى تنسيق HTML:



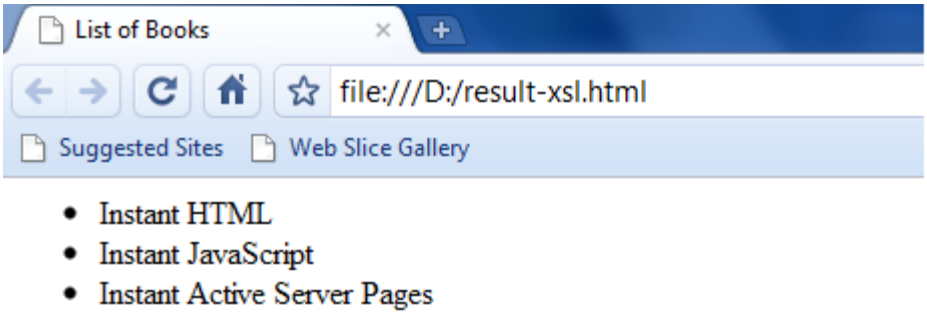
نلاحظ أننا وضعنا عناصر XML ضمن تنسيق HTML وعند حفظ الخرج في ملف HTML بدلاً من الشاشة عندئذٍ يستطيع أي مستعرض انترنت أن يفهم هذه ال Tags وينسق الخرج، ولتحقيق ذلك نستبدل السطر التالي:

```
//Print the Result on Screen ,so we choose Console.Out
 XmlTextWriter writer = new XmlTextWriter(Console.Out);
```

بالسطر التالي:

```
XmlTextWriter writer = new XmlTextWriter(@"D:\result-xsl.html",null);
```

عندما نقوم بفتح الملف result-xsl.html يظهر لنا كما يلي:



نلاحظ أهمية التحويل من XML إلى تنسيقات وأشكال أخرى، علماً أن xsl لا تؤمن لنا تنسيقاً وحسب وإنما تُمكننا من استخلاص بيانات معينة، كما شاهدنا سابقاً باستخدام for-each و value-of وتعليمات أخرى.

سَلسلة (Serialize) كائن في مستند XML

يستخدم الصنف System.Xml.Serialization.XmlSerializer لسلسة كائن في مستند XML، كما يسمح لنا أيضاً بأن نتحكم في عملية سلسلة الكائنات (يتم سَلسَلة كافة محتويات الكائن بما فيها المتحولات و الخصائص)، في الحالة الافتراضية يضع كافة الDataMember على شكل أوسمة <tag> </tag> ولكن يمكن أيضاً أن نغير طريقة تخزين الكائن مثلاً أن نضع الDataMember على شكل Attributes، أو أن لا نسمح بسلسلة خاصية مثلاً : كلمة السر لا نريد أن نضعها في مستند XML لذلك نضع قبل الخاصية أو المتحول ضمن الصنف الصفة [XmlIgnore] أو هي نفسها [XmlIgnoreAttribute].

ملاحظات:

1- في حال كان المتحول ضمن الصنف (Datameber أو يسمى Attribute) من نمط Public (أي لا يوجد خصائص) فأننا نطبق الصفات السابقة على المتحول العام نفسه كما يلي :

```
[XmlIgnore]
public int id;
```

وعندها لن يتم سَلسَلة المتحول id ضمن مستند XML.

2- أما في حال كان المتحول ضمن الصنف (Datamember أو يسمى Attribute) من نمط Pirvate عندها يكفي أن نضع الصفة فوق الخاصية الخاصة بالمتحول وذلك كما يلي:

```
string phoneNumber;

[XmlIgnoreAttribute]
public string PhoneNumber
{
    get { return phoneNumber; }
    set { phoneNumber = value; }
}
```

3- يمكننا أن نخزن متحول ما (ضمن صنف طبعاً)على شكل Attribute باستخدام الصفة [XmlAttribute()] وذلك كما يلي:

```
int id;
[XmlAttribute()]
public int ID
{
     get { return id; }
     set { id = value;}
}
```

لنرى الآن المثال التالي:

```
public class Person
{
    int id;
    string name;
    string phoneNumber;

     public Person()
     {
     }

   [XmlAttribute()]
   public int ID
   {
            get { return id; }
            set { id = value; }
    }

    public string Name
    {
            get { return name; }
            set { name = value; }
    }

    [XmlIgnoreAttribute]
    public string PhoneNumber
    {
            get { return phoneNumber; }
            set { phoneNumber = value; }
    }

    public void WriteToXMLFile(string path)
    {
        FileStream fileStream = new FileStream(path, FileMode.Create);

        XmlSerializer serializer = new XmlSerializer(this.GetType());

       serializer.Serialize(fileStream, this);

        fileStream.Close();
    }
    public void ReadFromXMLFile(string path)
    {
         FileStream fileStream = new FileStream( path, FileMode.Open);

         XmlSerializer serializer = new XmlSerializer(this.GetType());

         Person file;
```

```
        file = (Person)serializer.Deserialize(fileStream);

        fileStream.Close();

        this.ID = file.ID;
        this.Name = file.name;
         this.PhoneNumber = file.PhoneNumber;
       }
       #endregion
   }

static void Main(string[] args)
{
     Person p = new Person();
     p.ID = 10;
     p.Name = "mohammad";
     p.PhoneNumber = "12323";

     //Serialization
     p.WriteToXMLFile(@"D:\serializeoutput.xml");

     //DeSerialization
     p.ReadFromXMLFile(@"D:\serializeoutput.xml");

}
```

الفصل السادس

التعامل مع قواعد البيانات

ADO.NET

مقدمة

تُعد قواعد المعطيات أحد أهم العناصر في النظم المعلوماتية، بل إن قلب أي عمل برمجي هو البيانات، لهذا السبب سنتكلم عن طريقة جديدة للوصول إلى البيانات وهي مُقدمة من شركة مايكروسوفت هذه الطريقة هي ADO.NET.
نفترض في هذا الفصل وجود معرفة مُسبقة عن قواعد البيانات.

لمحة مختصرة عن تاريخ الوصول إلى البيانات

عندما أُنشئت أولى أنظمة قواعد المعطيات مثل Oracle وDB2 كان على المطورين ربط تطبيقاتهم مع قواعد البيانات من خلال مجموعة من التوابع الخاصة بنظام محدد لقواعد البيانات. المشكلة تكمن في أن لكل نظام إدارة قواعد معطيات DBMSمكتبته الخاصة من التوابع، فمثلاً Oracle تستخدم واجهة OCI و IBM DB2 يستخدم واجهة Sybase SQL Server. فإذا أرادت الشركة تغيير نظام قاعدة البيانات فأن عليها إعادة كتابة التطبيق من البداية.

لقد تم حل هذه المشكلة من خلال ODBC (**O**pen **D**ata **B**ase **C**onnectivity) الذي تم تطويره من قبل شركة مايكروسوفت وشركات أخرى.ولكن مشكلة ODBC هي صعوبة التعامل معها ولم تكن واجهاتها مبنية على COM.
عملت مايكروسوفت على جعل الوصول إلى البيانات أسهل وذلك بتطوير تقنيات متعددة بُنيت أساساً على ODBC منهاDAO (**D**ata **A**ccess **O**bject) الخاصة ب Access و RDO (**R**emote **D**ata **O**bject) ثم تم ضم هذين الأخيرين ضمن ADO وعلى الرغم من أن ADO كانت تستطيع استخدام ODBC للاتصال مع قواعد المعطيات إلا أن مايكروسوفت قد بنت (**A**ctiveX **D**ata **O**bject) إختصاراً ADO بحيث تستخدم مزود قاعدة بيانات جديد اسمه OLEDB(Object Linking and Embedding Data Base)[1]، والذي صُمم ليكون طبقة مبنية على COM فوق مزود ODBC، سنرى الآن كيفية عمل ADO و OLEDB معاً للوصول إلى قاعدة البيانات .
عندما أنشأت مايكروسوفت ADO أرادت تزويد المطورين بواجهة لمصادر بياتهم والتي تتلاءم طبيعياً مع مشاريعها المطورة والمبنية على COM.
بدت ال API المقدمة من ADO مريحة جداً لمطوري COM واستطاعوا العمل مع اتصالات ADO بطريقة غرضية التوجه، وقد قامت ADO باستخدام COM لتوجيه مجموعات السجل RecordSets التابعة لها من مكان أخر.تم تطوير عدة نسخ من ADO أخر هذه النسخ تدعم القراءة والكتابة في XML من مجموعات السجل التابعة لك.

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://msdn.microsoft.com/en-us/library/windows/desktop/ms722784(v=vs.85).aspx

مقارنة ADO مع ADO.NET

على الرغم من أن الاسمADO.NET مشتق من الاسم ADO. إلا أنهما تقنيتان مختلفتان لدرجة كبيرة سنتكلم عن هذه الاختلافات:

- مجموعة سجل ADO أصحبت تُسمى الآن مجموعة بيانات ADO.NET DataSet.
- يمكن لمجموعة البيانات (DataSet) أن تحوي عدة جداول مع وجود علاقات بين الجداول.
- تستخدمADO.NET تنسيق XML لنقل محتويات DataSet من مكان لأخر.
- مجموعة البيانات DataSet هي شكل غير مُتصل من ولوج البيانات، سنفصل لاحقاً في هذا المعنى.
- لا يحتاج المطورين إلى إقلاق أنفسهم بمؤشرات قاعدة المعطيات المستخدمة في ADO والأقفال المستخدمة من قبل المخدّم عند استخدام مجموعة البيانات DataSet في ADO.NET.

فيADO.NET ثم الاستغناء عن مجموعة السجل وأصبحت مجموعة البيانات DataSet هي محور هذا العالم الجديد.

سابقاً كانت مجموعة السجل هي الكائن الحاوي للبيانات الرئيسية التي تعمل معه في ADO. إن المعلومات الموجودة ضمن السجل تعادل جدولاً واحداً من البيانات، عندما يقوم أحد الكائنات المبنية على الشيفرة بنقل مجموعة سجل إلى كائن أخر فأنه يستخدم COM كواسطة اتصال للقيام بذلك.

تستخدم ADO.NET أيضاً XML كطريقة لنقل محتويات DataSet من مكان لأخر.

سنتفحص بعض الفروق بين ADO و ADO.NET من خلال الفقرة التالية.

الولوج المتصل وغير المتصل إلى قاعدة المعطيات

قبل ADO.NET كان الولوج إلى قاعدة المعطيات نموذجياً (يتم بشكل متصل)، مثلاً:لنفرض أنك تريد تعديل سجل موظف في قاعدة البيانات .لبرمجة ذلك عليك أن تؤسس اتصالاً إلى قاعدة البيانات، ثم تحدد موقع السجل بتحريك مؤشر قاعدة البيانات، ثم تقوم بإجراء تعديلاتك على هذا السجل ومن ثم تقطع الاتصال مع قاعدة البيانات(كل هذه العملية والاتصال مفتوح مع المخدم Server). في هذه الحالة كلما أردنا أن نحصل على قيمة سجل من قاعدة المعطيات أو إجراء عملية إضافة أو حذف أو تعديل فأن ذلك يجب أن يتم في وضع متصل وقد لا تستطيع الاتصال مع مخدم قاعدة البيانات، قد يكون مشغولاً بإتصالات أخرى، إن تحرير هذه الاتصالات هي الهدف الأول لمطوري قواعد المعطيات.

تعمل ADO.NET مع البيانات بأسلوب مختلف تماماً هو أسلوب عدم الاتصال ويكون الاتصال مع مُخدّم قاعدة المعطيات قصيراً وموجهاً، مما يسمح ل NET. بتمرير الاتصال بسرعة لشخص أخر. إذا أردت تعديل سجل موظف باستخدام ADO.NET ستتم العملية كما يلي:

1. يؤسس ADO.NET اتصالاً مع قاعدة المعطيات.
2. سيتم استخراج البيانات التي تريد العمل معها من قاعدة المعطيات لتُصبح على ذاكرة الجهاز المحلي في DataSet.

الآن يمكننا معالجة هذه البيانات للفترة التي نرغبها دون أن نحجز اتصالاً، وعندما نكون جاهزين لتخزين البيانات المُعدّلة نتصل مع مخدم قاعدة المعطيات لفترة قصيرة من الوقت لإجراء التعديلات.

ماهي ADO.NET؟

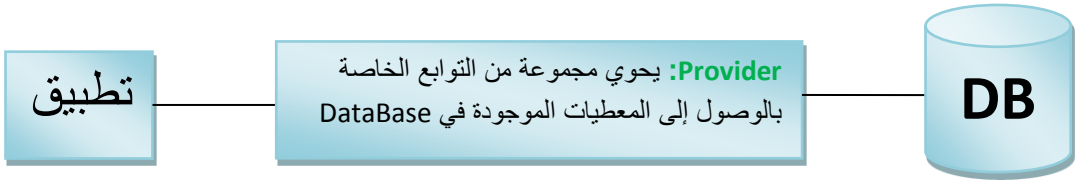
هي مجموعة من الأصناف المُدمجة مع .NET Framework والتي تُمكّننا من الوصول إلى البيانات من خلال اللغات التي يدعمها .NET FrameWork، توجد أصناف ADO.NET ضمن المُجمعة System.Data.dll، لقد وجدت ADO.NET لتحقيق عدد من الأهداف:

1. توفير الوصول إلى البيانات العلائقية في حال كانت البيانات مُخزنة في جداول وهذا النموذج يُسمى النموذج العلاقاتي، وغير العلائقية وتعني أي مصدرمعطيات يمكن الوصول إليه والتعامل معه.
2. لتوحيد الوصول إلى بيانات XML والبيانات العلائقية، وذلك من خلال توفير جسر بين البيانات العلائقية المرتبة ضمن أسطر وأعمدة وبين مستندات XMLالتي تأخذ هيكلاً هرمياً.
3. دعم التطبيقات متعددة الطبقات (Multi-tier) عبر الانترنت.

إن الهدف الرئيسي ل ADO.NET هو توفير وصول بسيط للبيانات العلائقية. ويتم ذلك من خلال أصناف سهلة تمثل الجداول والأعمدة والحقول في قواعد البيانات العلائقية. بالإضافة إلى ذلك يقدم ADO.NET الصنف Dataset وهو عبارة عن قاعدة معطيات ضمن الذاكرة (محطة مؤقتة للعمل ريثما يتم تعديل نسخة الذاكرة مع نسخة قاعدة المعطيات الفيزيائية).

مزودات البيانات في .NET

المقصود بمزود البيانات Provider، هو أنهُ طبقة مجردة أو وسيطة بين التطبيق و نظام إدارة قواعد المعطيات DBMS وذلك كما يلي:



هناك قسمان رئيسيان لكائنات ADO.NET: الكائن DataSet ومزود بيانات .NET.

يتألف مزود البيانات Provider من عدد من مكونات البيانات الخاصة التي تسمح لنا بالوصول للبيانات وبالتواصل مع مصادر البيانات كلاً على حده وهي كالتالي:

1- مزود البيانات OLEDB: ويستخدم مع قواعد بيانات Access موجودة ضمن فضاء الأسماء System.Data.OleDb.

2- مزود بيانات SqlClient: يستخدم مع قواعد بيانات SQL Server موجودة ضمن فضاء الأسماء System.Data.SqlClient.

3- مزود بيانات ODBC: مزود عام لجميع أنظمة قواعد المعطيات (يمكن استخدامه للاتصال مع قاعدة معطيات Oracle مثلاً) موجودة ضمن فضاء الأسماء System.Data.Odbc.

أهم أصناف ADO.NET

نستعرض الآن أهم الأصناف المستخدمة في ADO.NET وهي تنقسم إلى قسمين:

1. **كائنات مزود البيانات Provider:** وهي كالتالي:

كائن الاتصال Connection
كائن الأمر Command
مُنشئ الأوامر CommandBuilder
قارئ البيانات DataReader
مُكيف البيانات DataAdapter

في الحقيقة أن كائنات المزوّد Provider السابقة موجهة لنوع محدد من مصادر البيانات سندرس منها SqlServer الموجودة ضمن فضاء الأسماء System.Data.SqlClient، حيث أن جميع عمليات القراءة والكتابة ستتم من خلال هذه الأصناف.

2. **كائنات المستهلك .NET Consumer Object:** ونقصد بها كائن DataSet وجميع الكائنات التي تندرج تحته وهي كالتالي :

أن كائنات المستهلك DataSet موجهة لجميع المزودات حتى المزودات التي سوف تُنشئها مايكروسوفت لاحقاً، في الحقيقة أن DataSet هي طبقة مجردة في الذاكرة تستخدم عند استرجاع نتائج الاستفسارات من قاعدة المعطيات، وأهم ما يميز هذه الكائنات (كائنات المستهلك) هي أنها تعمل في نمط عدم الاتصال هذا يعني أن أي تعديل على البيانات الموجودة في هذه الكائنات لا ينعكس تلقائياً على البيانات الموجودة في مصدر البيانات وإنما على نسخة البيانات الموجودة في الذاكرة.

تتطلب كائنات المزود اتصالاً مستمراً بمصدر البيانات لكي تتمكن من العمل، حيث أن مع كل عملية قراءة أو كتابة تعديلات في DataBase نحتاج إلى الاتصال بمصدر البيانات، غالباً يكون التعامل مع عملية القراءة والكتابة وفقاً للسيناريو التالي:

1. نقوم بفتح اتصال مع مصدر البيانات من خلال الكائن Connection (في الحقيقة لا يوجد صنف اسمه Connection ولكن يوجد OleDbConnection و SqlConnection وقد استخدمنا كلمة Connection للتعميم).
2. نقل البيانات من مصدر البيانات ووضعها ضمن كائنات المستهلك في الذاكرة (DataSet).
3. قراءة البيانات ومعالجتها من خلال كائنات المستهلك.
4. نقل التحديثات التي طرأت على البيانات في الذاكرة إلى مصدر البيانات من خلال كائنات المزوّد.

نلاحظ أن الخطوة الثالثة لا تحتاج إلى اتصال مباشر مع مصدر البيانات(DB)، وهذا يعني أننا لن نحتاج لوجود اتصال مباشر مع مصدر البيانات في الوقت التي نقرأ البيانات أو نعالجها وإنما سنقتصر على الاتصال مع مصدر البيانات عند جلب البيانات و عند عكس التحديثات بين نسخة DataSet الموجودة ضمن الRAM ونسخة البيانات الموجودة في مصدر البيانات (قاعدة المعطيات الفيزيائية).

كائنات المزوّد Provider

تكلمنا سابقاً أن هذه الكائنات مُعرّفة في كل مزود من مزودات NET. تكلمنا أن أسماءها تختلف من مكون لآخر مثلاً: كائن الاتصال في OLEDB يسمى OleDbConnection أما في مزود SQL فيسمى SqlConnection وهكذا بالنسبة لبقية الكائنات، سنقوم الآن بشرح كائنات المزود.

كائن الاتصال Connection

يوفر الاتصال مع مصدر البيانات، فإذا كنت تود الاتصال مع قاعدة بيانات فانك بحاجة لتحديد مسار قاعدة المعطيات واسم المخدم وقد نحتاج إلى username وPassword للولوج إلى ضمنها.

كائن الأمر Command

يمثل هذا الكائن استعلام SQL Query والذي يتم تنفيذه داخل DBMS الخاص بقاعدة المعطيات، نفترض وجود معرفة مسبقة عن SQL.

كائن مُنشئ الأوامر CommandBuilder

يستخدم لبناء أوامرSQL لمعالجة البيانات من كائنات تعتمد على استعلام لجدول وحيد، سنشرح هذا الكائن بتفصيل أكبر عند دراسة مجموعة من الأمثلة.

كائن قارئ البيانات DataReader

يعتبر هذا الكائن الأبسط من حيث الاستخدام و الأسرع من حيث الوصول لقراءة البيانات وبشكل أمامي فقط Forword-Only من مصدر البيانات.

كائن مُكيف البيانات DataAdapter

يستخدم هذا الكائن لتأدية عمليات عديدة محددة وموجهة لمصدر البيانات. يتضمن ذلك تحديث البيانات وملئ مجموعة البياناتDataSet من خلال نتيجة الاستعلام المُنفذة داخل ال DBMS (أو البيانات القادمة من مصدر البيانات).

كائنات المستهلك Consumer Object

تمثل هذه الكائنات الوسيلة التي سنستخدمها لمعالجة البيانات والوصول إليها بعد الحصول عليها من مصدر البيانات، لذلك فقد سُميت بكائنات المستهلك (أو المبرمج) وهي كالتالي:

كائن مجموعة البيانات DataSet

يمثل كائن DataSet مجموعة من الجداول المرتبطة ببعضها البعض والتي يُشار إليها كوحدة واحدة في التطبيق. وهو يعمل في نمط عدم الاتصال، هذا يعني أن أي تعديل على كائن DataSet لا يعني تعديل البيانات في مصدر البيانات، يتوجب علينا بعد التعديل نقل وتحديث البيانات التي طرأت على كائن DataSet إلى مصدر البيانات بواسطة عملية واحدة وفعّالة.

يحوي كائن DataSet عدداً من الخصائص التي تمكننا من الوصول إلى كائنات المستوى الأدنى، مثل:كائنات الجداول، كائنات الأسطر، أو كائنات الأعمدة أو حتى العلاقات الرابطة بين الجداول الموجودة ضمن كائن DataSet ,هذه الكائنات هي:

كائن الجدول DataTable

يمثل الكائن DataTable جدولاً في DataSet، مثلاً : يمكن أن يمثل هذا الجدول جدول الزبائن أو أي جدول آخر في مصدر البيانات.

كائن العمود DataColumn

يمثل هذا الكائن عموداً (أو حقلاً) في جدول DataTable.

كائن السطر DataRow

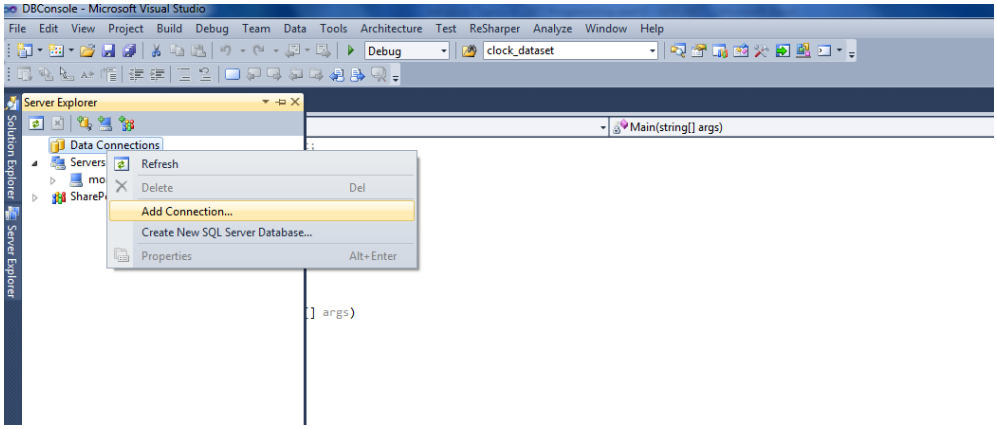
يُمثل هذا الكائن سطراً (أو سجلاً) في جدول.

كائن العلاقة DataRelation

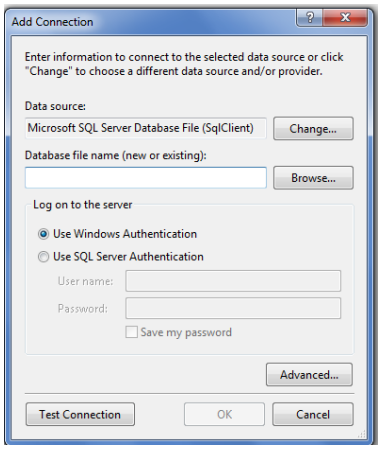
يُمثل هذا الكائن العلاقة بين جدولين من خلال عمود مُشترك، مثلاً:يمكن أن يرتبط جدول الطلبيات مع جدول الزبائن من خلال عمود مشترك في كلا الجدولين وهو عمود رقم الزبون. يمثل هذا العمود بالنسبة لجدول الزبائن رقماً فريداً (Primary Key) للزبون أما بالنسبة لجدول الطلبيات فهو يمثل رقم الزبون الذي قام بالطلبية (Foreign Key).

الربط مع قاعدة المعطيات

بداية سنستخدم في جميع الأمثلة قاعدة المعطيات المعروفة NorthWind.
للقيام بعملية الربط مع قاعدة المعطيات نقوم بما يلي:
1- من قائمة View نختار Server Explorer لعرض جميع قواعد المعطيات التي نتصل معها ثم نقوم كما تشرح الصورة التالية:



تكون قاعدة المعطيات موجودة لدينا على الحاسب لدينا وهذا ماسنقوم به، مثلاً يمكن أن تكون قاعدة المطيات عبارة عن ملف Access بلاحقة mdb أو أن تكون SQL Server عندها يكون لدينا ملف المعطيات بلاحقة mdf وملف Log (سجل) بلاحقة Ldf.



يمكننا اختيار ما نريد من خلال الضغط على الزر ..Change في الصورة السابقة لنختار الربط مع ملف خاص بقاعدة المعطيات Access أو SQL Server أو Oracle أو غيرها.
نختار SQL Server لأننا سنربط مع قاعدة معطيات SQL Server، ثم نضغط على الزر Browse لتحديد مكان قاعدة المعطيات ثم نتفحص الاتصال عن طريق الضغط على الزر TestConnection ثم نضغط موافق.

الآن تظهر قاعدة المعطيات لدينا، كما يمكن استعراض كافة محتوياتها ال Solution Explorer بما فيها من مخططاتDiagrams وجداول ومناظير Views وإجراءات مُخزّنة وأمور أخرى.

الوصول إلى جدول وقراءة محتوياته باستخدام قارئ البيانات Reader

إلى الآن قمنا بالربط مع قاعدة المعطيات والآن لنرى الكود التالي:

```
using System.Data.SqlClient;

staticvoid Main(string[] args)
 {   //1: Connection To DB
SqlConnection conn =new SqlConnection(@"Data
      Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\moammad\Desktop\Northwind
      && pubs DataBases\northwnd.mdf;Integrated Security=True;Connect
      Timeout=30;UserInstance=True");

//2: Open Connection
conn.Open();

//3: Creating  SQL Command for this Connection (conn)
SqlCommand command = conn.CreateCommand();
command.CommandText = "Select  CustomerID, CompanyName from Customers";

//4: this is the fastest Way to Reading Data from DB - Forword-only    //because
thisClass is using Cursor Model
SqlDataReader reader = command.ExecuteReader();//Execute SQL Command

      //5: reading the Next Record in the Reader and Return false
      //if the Reader is Ended
      while (reader.Read())
      {
      //OutPut CusomerID and CompanyID
      Console.WriteLine("CustomerID= {0} CompanyName={1}",
            reader["CustomerID"], reader["CompanyName"]);

      }

// 6:free Resources
reader.Close();

       //7:Close Connection
      conn.Close();
}
```

عادة نستخدم السناريو التالي عند التعامل مع قاعدة المعطيات:

1- في البداية قمنا بإنشاء اتصال باستخدام كائن SqlConnection وقمنا بتحديد ConnectionString في الباني، ونحصل عليه من خلال النقر بالزر الأيمن على قاعدة المعطيات من Server Explorer ثم نختار Properties وننسخ الخاصية ConnectionString تحوي هذه الخاصية المعلومات المطلوبة للولوج إلى قاعدة المعطيات مثل اسم المزود ومسار قاعدة المعطيات ونمط الأمن المستخدم وأمور أخرى.

2- ثم قمنا بفتح الاتصال مع قاعدة المعطيات.

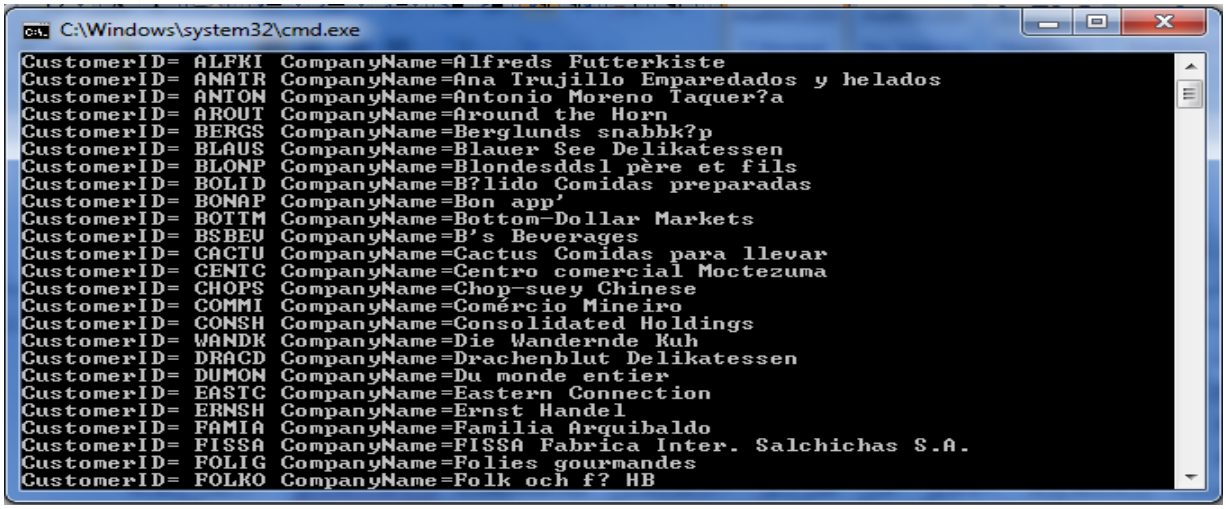
3- ثم قمنا بإنشاء كائن أمر SQL وهو موجه لكائن الاتصال Conn.

4- قمنا بتنفيذ الأمر عن طريق الكائن Command باستخدام الطريقة ExecuteReader والتي ترد مؤشر على كائن SqlReader والذي يُعتبر الأسرع في عملية القراءة إلى الأمام، ويستخدم هذا الصنف نموذج المشيرة Cursor Model وهو نموذج مأخوذ من ADO.

5- بعد أن قمنا بتنفيذ الاستفسار وحصلنا على النتيجة سنقوم بعرض هذه النتيجة من خلال حلقة (While) وسنقوم بعرض محتويات الأعمدة التي قمنا بإحضارها بواسطة تعليمة SQL فقط أي لا يمكن تعديل البيانات.

6,7- تحرير الموارد.

لنرى الخرج الآن:



تحديث البيانات

الآن أصبح بإمكاننا قراءة البيانات من قواعد البيانات وسنتعلم الآن كيفية تحديث البيانات. لنرى الشيفرة التالية حيث سنقوم بتغيير محتويات العمود CompanyName في السطر (السجل) الأول من الجدول Customers ثم سنشاهد النتائج قبل وبعد التعديل:

```
staticvoid Main(string[] args)
{

//1: Connection To DB
SqlConnection conn =new SqlConnection(@"Data
       Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\moammad\Desktop\Northwind && pubs
       DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User
       Instance=True");


//2: Open Connection
conn.Open();


//3:Creating Adapter to Represent  SQL Command To Be Executed  , Fill DataSet and  // Update
DataBase
SqlDataAdapter adapter = new SqlDataAdapter(@"Select  CustomerID, CompanyName from
                     Customers",conn);


//4: Automatically generates single-table commands that are used to reconcile changes //madeto a
DataSet with the associated SQL Server databaser
SqlCommandBuilder builder = newSqlCommandBuilder(adapter);//Wrapping


//5: Creating DataSet to Fill it by SqlDataAdapter
DataSet ds = newDataSet();

//6: Excuting SQL Query and fill DataSet(Table name in the DataSet is  "Cusomers") By the Result
of Query .
adapter.Fill(ds, "Customers");//the Name of the Source Table to use for Table Mapping

//7: Writing CompanyName in the First Record  Befor  Changed it
Console.WriteLine("Befor updates: CompanyName is :{0} ",
       ds.Tables["Customers"].Rows[0]["CompanyName"]);

//8: Updating the first Record - updates the CompanyName  Column
ds.Tables["Customers"].Rows[0]["CompanyName"] = "Ganoom & Alyan";


   // Now All this update are StilL in DataSet( in the RAM)

//9:Reflect  the Changes(insert,update,Delete) in the Table Customers in the DataSet      on the
Same Table (Customers) in The Physical DB
adapter.Update(ds, "Customers");


//Writing CompanyName in the First Record  Befor  Changed it
```

```
Console.WriteLine("After updates: CompanyName is :{0} ",
       ds.Tables["Customers"].Rows[0]["CompanyName"]);

conn.Close();
}
```

شرح الكود:

في البداية قمنا بفتح اتصال مع قاعدة المعطيات ثم قمنا بإنشاء كائن مكيف البيانات DataAdapter وقمنا بتهيئة هذا الكائن باستعلام SQL وربطناه مع كائن الاتصال، يُعتبر الكائن DataAdapter كائن ذو غرض عام يستخدم لعدد من أنواع العمليات على البيانات مثل:

1- ملئ Fill مجموعة البيانات DataSet بواسطة نتيجة استعلام كما فعلنا في السطر 6 عندما استخدمنا الطريقة Fill.

2- تحديث update مجموعة بيانات DataSet وذلك بمطابقة التغييرات التي تحدث في DataSet مع النسخة الموجودة في قاعدة البيانات الفيزيائية وهذا ما قمنا به في السطر 9.

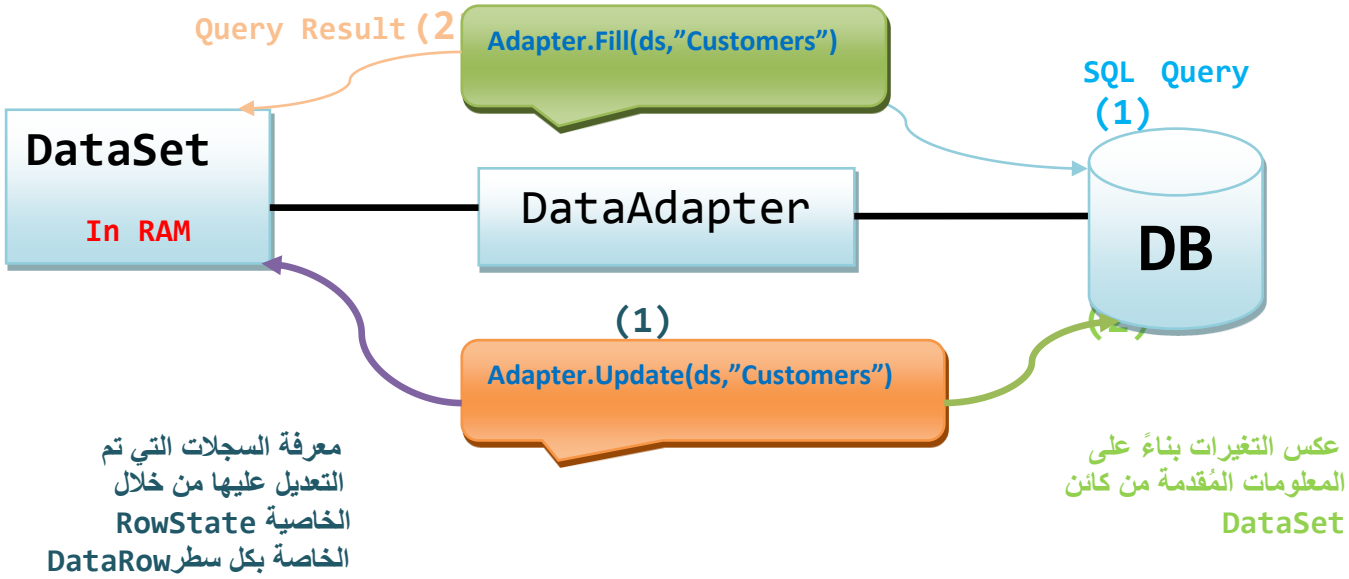
كائن مُنشئ الأوامر ObjectCommandBuilder

بالنسبة لحالة تحديث جدول وحيد فأننا لسنا بحاجة لتعلم كيفية كتابة تعليمات SQL للقيام بعملية التحديث، وهذا ما سيزودنا به الصنفSqlCommandBuilder والذي يُغلف كائن SqlDataAdapter.

الطريقة Fill الخاصة للكائن DataAdapter

قبل أن نتمكن من استخدام كائن DataSetعلينا أن نملأه بالبيانات من قاعدة البيانات, سنستخدم المنهج Fill للكائن DataAdapter للقيام بذلك، ولكن لماذا نستخدم المنهج Fill للكائن DataAdapter ولا نستخدم المنهج Fill الخاص بكائن DataSet؟

في الحقيقة إن ذلك يعود إلى أن الكائن DataSet يُمثل تجريداً للبيانات في الذاكرة بينما يُمثل كائن DataAdapter الكائن (الجسر) الذي يربط بين كائن DataSet مع قاعدةالمعطيات الفيزيائية (انظر الرسم التوضيحي في الأسفل)، أما البارمتر الثاني فيمثل اسم الجدول (كائن DataTable) ضمن كائن DataSet الذي نود تحميل البيانات إليه (يمكن أن نختار أي اسم).



ملئ Fill وتحديث Update مجموعة بيانات DataSet

الطريقة Update الخاصة للكائنDataAdapter

كما لاحظنا من الشكل السابق، إن الكائن DataAdapter يمثل جسر بين كائن DataSet والذي يوجد في الذاكرة RAM وقاعدة المعطيات الفيزيائية، وبعد تعديل (Insert,Update,Delete) قيم السجلات في الذاكرة (في الDataSet)، في حالتنا نقوم بتعديل عمود اسم الشركة في السجل الأول يجب أن نقوم بعكس التغيرات التي طرأت على DataSet بما يقابلها في قاعدة المعطيات الفيزيائية. وهذا ما تقوم به الطريقة Update والتي تعدل على جدول واحد يتم تمرير اسمه كبارمتر ثاني للطريقة Update ويجب أن يكون له نفس اسم الجدول عند استخدام الطريقة Fill.

تعديل البيانات ضمن مجموعة البيانات DataSet

كما تكلمنا سابقاً فأن كائن DataSet يحوي مجموعة من كائنات DataTable يمكن الوصول إلى أي جدول من خلال الخاصية Tables ثم عن طريقIndexer[] وذلك كما يلي:

```
DataTable T=ds.Tables[“ColumnName”];
```

كما يمكن أيضاً أن نضع رقم العمود أيضاً بدلاً من اسمه.

ثم من خلال هذا الجدول نستطيع الوصول إلى سطر معين من خلال الخاصية Rows حيث أن لكل جدول DataTable مجموعة من الأسطر، ثم عن طريق Indexer[] نستطيع تحديد رقم السطر وبعد ذلك يمكننا تحديد رقم العمود من خلال Indexer[] (أو يمكن نحدد اسمه) وبهذا نكون قد عدلنا قيمة خليه Cell ضمن جدول DataTableضمن DataSet.

```
ds.Tables["Customers"].Rows[0]["CompanyName"] = "Ganoom & Alyan";
```

ويكون الخرج كما يلي:

إضافة سجلات جديدة

سنقوم في هذا المثال بإضافة سجل جديد إلى الجدول Customers، لنرى المثال التالي:

```
staticvoid Main(string[] args)
{
       //1: Connection To DB
SqlConnection conn =newSqlConnection(@"Data
       Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\moammad\Desktop\Northwind && pubs
       DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User
       Instance=True");


//2: Open Connection
conn.Open();

       //Creating Adapter to Represent  SQL Command To Be Excuted
       //Fill DataSet and Update DataBase

SqlDataAdapter adapter = newSqlDataAdapter(@"Select  CustomerID, CompanyName from
       Customers",conn);


//Automatically generates single-table commands that are used to reconcile changes        //made
to a
//DataSet with the associated SQL Server databaser
SqlCommandBuilder builder = newSqlCommandBuilder(adapter);//Wrapping


//Creating DataSet to Fill it by SqlDataAdapter
DataSet ds = newDataSet();

//Excuting SQL Query and fill DataSet(Table name in the DataSet is  "Cusomers") By        //the
Result of Query .
adapter.Fill(ds, "Customers");//the Name of the Source Table to use for Table Mapping

//Number of Rows Befor Add Record
Console.WriteLine("Number of Rows Befor Add Record is :{0} ",
       ds.Tables["Customers"].Rows.Count);
```

```
//Creating DataRow from Customers Table
DataRow Row = ds.Tables["Customers"].NewRow();

// Filling it
Row["CustomerID"] = "112";

Row["CompanyName"] = "MU_nizar & Alyan & Hammod";

//Add Row to Customers Table
ds.Tables["Customers"].Rows.Add(Row);

//Number of Rows After Add Record
Console.WriteLine("Number of Rows After Add Record is :{0} ",
       ds.Tables["Customers"].Rows.Count);

adapter.Update(ds, "Customers");

conn.Close();
}
```

شرح الكود:

يختلف هذا الكود عن سابقه في أنّنا نقوم بإنشاء مرجع Reference من الصنف DataRows .في الحقيقة أن الصنف DataRow ليس له باني (لا يمكن إنشاء كائن منه) في الحقيقة يوجد باني ولكنه Protected وذلك لأن الصنف DataRow يمثل سطراً بشكل مجرد ولكن ما هي بنية هذا السطر (عدد الأعمدة وأنماطها)؟هذا ما يعطينا إياه الجدول DataTable حيث يمكن إنشاء سطر من هذا الجدول بواسطة الطريقة ()NewRow وإمساكه عن طريق Reference من الصنف DataRow.

ثم قمنا بتهيئة هذا السطر بالقيم وبعد ذلك قمنا بإضافة هذا السطر إلى الجدول Customers وذلك بعد الوصول إليه من خلال الخاصية Tables من الDataSet ومن ثم تحديد اسم الجدول (Customers) وبعد ذلك اختيار الخاصية Rows ثم من خلال الطريقة Add نضيف هذا السطر إلى الجدول.

ملاحظة: نلاحظ أننا لم نهيئ جميع الحقول (الأعمدة) عند تهيئة السطر وذلك لأننا قمنا بجلب العمودين CustomerID,CompanyName من خلال استعلام SQL وأي محاولة لتهيئةعمود غير هذه الأعمدة فأنه سيولد خطأ بالنسبة للشيفرة.

لنرى الخرج الآن:

حذف سجل من جدول معين ضمن قاعدة المعطيات

سنقوم في هذا المثال بالبحث عن السجل الذي قمنا بإنشائه قبل قليل وحذفه من قاعدة المعطيات.

```
staticvoid Main(string[] args)
{
	//1: Connection To DB
SqlConnection conn =newSqlConnection(@"Data
	Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\moammad\Desktop\Northwind && pubs
	DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User
	Instance=True");


//2: Open Connection
conn.Open();


//Creating Adapter to Represent  SQL Command To Be Excuted  , Fill DataSet and           Update
DataBase
SqlDataAdapter adapter = newSqlDataAdapter(@"Select  CustomerID, CompanyName from
	Customers",conn);


//Automatically generates single-table commands that are used to reconcile changes       //made
to a DataSet with the associated SQL Server databaser
SqlCommandBuilder builder = newSqlCommandBuilder(adapter);//Wrapping


//Creating DataSet to Fill it by SqlDataAdapter
DataSet ds = newDataSet();

//Excuting SQL Query and fill DataSet(Table name in the DataSet is  "Cusomers") By
	the Result of Query .
adapter.Fill(ds, "Customers");//the Name of the Source Table to use for Table Mapping
//Number of Rows Befor Add Record
Console.WriteLine("Number of Rows Befor Add Record is :{0} ",
	ds.Tables["Customers"].Rows.Count);

//key is the set of Columns
DataColumn[] key = newDataColumn[1];

//Selectin the Key
key[0] = ds.Tables["Customers"].Columns["CustomerID"];

//Add Primary Key
ds.Tables["Customers"].PrimaryKey = key;

DataRow findrow=  ds.Tables["Customers"].Rows.Find("112");

if(findrow!=null)
	//this statement Does not Delete findrow from DataSet
	//but Changes the Row State into deleted
findrow.Delete();

adapter.Update(ds, "Customers");

//Number of Rows After Add Record
```

```
Console.WriteLine("Number of Rows After Add Record is :{0} ",
       ds.Tables["Customers"].Rows.Count);


conn.Close();
}
```

شرح الكود:

في هذا المثال سنقوم بحذف سجل الزبون ذو الرقم 112 من الجدولCustomer.

نعلم أن كل سجل ضمن جدول يجب تمييزه عن غيره من السجلات بواسطة مفتاح، هذا المفتاح يتكون من مجموعة من الأعمدة لذلك قمنا بإنشاء مصفوفة من الأعمدة اسمها Key وهي تُمثل المفتاح، ثم قمنا بتحديد هذا المفتاح وهو العمود CustomerID ثم قمنا بتحديد المفتاح الأولي للجدول Customers وذلك عن طريق الخاصية Primary Key (نلاحظ أنها تأخذ مصفوفة من الأعمدة تُمثل المفتاح الأولي). كل ما قمنا به قبل قليل هو من أجل استدعاء الطريقة Find الخاصة بالمجموعة Rows (Collection) الموجودة ضمن الجدول Customers، تأخذ هذه الطريقة المفتاح الذي يميز السجل الذي نود البحث عنه ويرد مرجع على الصنف DataRow.

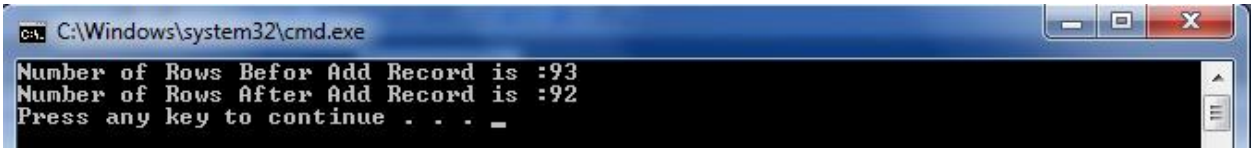
ثم قمنا باستدعاء الطريقة Delete الخاصة بالصنف DataRow، تقوم هذه الطريقة بتغيير الخاصية RowState الخاصة بهذا الكائن (DataRow) إلى Deleted ولا تقوم بحذف السجل من مجموعة المعطيات DataSet إلا بعد أن نقوم باستدعاء الطريقة Update، حيث تقوم هذه الطريقة بالمرور على عناصر مجموعة الكائنات Rows في كائن DataSet وعبر قراءة الخاصية RowState لكل سطر يقوم بتحديث السجلات في قاعدة المعطيات الفيزيائية.

في حال لم نقم بتعريف مفتاح وربطه بالجدول فأن الطريقة Find سترمي استثناء من نوع MissingPrimaryKeyException.

ملاحظة: تقوم الطريقة AcceptChanges للكائن DataSet بتأكيد (Commit) جميع التغيرات الحاصلة على ال DataSet لذلك إذا قمنا باستدعاء هذه الطريقة قبل الطريقة Update فأنه لن يتم تحديث السطر مع السطر الموافق له في قاعدة المعطيات الفيزيائية وذلك لأن الطريقة AcceptChanges تقوم بحذف جميع الأسطر التي لها الخاصية RowState تساوي Deleted من ال DataSet.

لهذا السبب ينصح المطورون بعدم استدعاء المنهج (الطريقة) AcceptChanges قبل استدعاء المنهج Update إذا أردنا تطبيق التغيرات على قاعدة المعطيات الفيزيائية.

لنرى الخرج الآن:



الوصول إلى عدة جداول عبرالكائن DataSet

إنّ أحد أهم المزايا التي يقدمها لنا نموذج ADO.NET هي الإمكانيات التي يوفرها لنا كائن DataSet حيث يسمح لنا بالتعامل مع عدة جداول في وقت واحد وتعريف العلاقات بين هذه الجداول وكل هذا ضمن كائن DataSet وحيد.
يستخدم الكائن DataRelation لوصف العلاقات بين الجداول في DataSet فلكل DataSet مجموعة (Collection) باسم Relations تحوي على كائنات من نوع DataRelation وتمكننا من تعريف وقراءة الروابط بين كائنات DataTable في المجموعة (Collection) على شكل خاصية اسمها Tables.
سنتكلم عن العلاقة بين جدول الزبائن (Customers) وجدول الطلبيات Orders (علاقة واحد إلى كثير) حيث أن الزبون الواحد يستطيع أن يطلب أكثر من طلبية (نعلم أيضاً أنه يجب أن يكون هناك عمود مشترك بين الجدولين- في مثالنا هو رقم الزبون-).

التعامل مع العلاقات بين الجداول باستخدام DataRlation

لاستخدام العلاقات بين الجدولين علينا أن ننتقل من سجل في أحد الجدولين إلى السجلات المرتبطة بذلك السجل في الجدول الأخر.تُسمى هذه العملية بالإبحار Navigation.
تتألف هذه العملية عادة من عبور من سجل أب في الجدول الأول (Customers مثلاً) إلى سجلات الأبناء المرتبطة معه في الجدول الأخر(Ordersمثلاً) وهذا ما سنوضحه من خلال الشيفرة التالية:

```
Static void Main(string[] args)
{
       //1: Connection To DB
SqlConnection conn =newSqlConnection(@"Data
       Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\moammad\Desktop\Northwind && pubs
       DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User
       Instance=True");


//2: Open Connection
conn.Open();

       //Creating Adapter to Represent  SQL Command To Be Excuted  , Fill DataSet and
       //Update DataBase
SqlDataAdapter custadapter = newSqlDataAdapter(@"Select  * from Customers",conn);

SqlDataAdapter ordadapter = newSqlDataAdapter(@"Select * from Orders", conn);

//Creating DataSet to Fill it by SqlDataAdapter
DataSet ds = newDataSet();

//Excuting SQL Query and fill DataSet(Table name in the DataSet is  "Cusomers") By
       the Result of Query .
custadapter.Fill(ds, "Customers");//the Name of the Source Table to use for Table Mapping


ordadapter.Fill(ds, "Orders");


//Creating DataRelation **
```

```
//CustOrders is the Name of Rlation , Second Argument is the Parent Column ,Third  Argument is
the Child Column
DataRelation custOrdRela =
       ds.Relations.Add("CustOrders",ds.Tables["Customers"].Columns["CustomerID"]
       ,ds.Tables["Orders"].Columns["CustomerID"]);


// طباعة معلومات كل زبون مع الطلبيات التي طلبها
foreach (DataRow CustRow in ds.Tables["Customers"].Rows)
{
Console.WriteLine("Customer ID: " + CustRow["CustomerID"] + "Name: " +
       CustRow["CompanyName"]);

foreach (DataRow OrderRow in CustRow.GetChildRows("CustOrders"))

Console.WriteLine("Order Id: " + OrderRow["OrderID"]);
}

conn.Close();
}
```

شرح الكود:

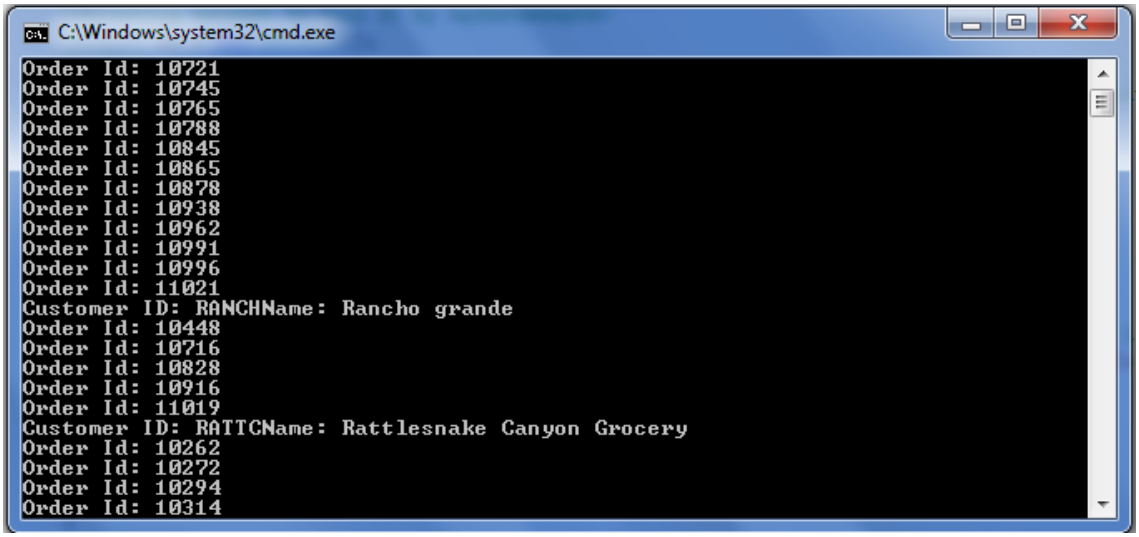
في هذه المرة قمنا بملىء Fill لكائن DataSet بجدولين هما جدول الزبائن وجدول الطلبيات، ثم قمنا بإنشاء كائن DataRelation وربطه بكائن DataSet من خلال الخاصية Relations الخاصة بكائن DataSet ، وبواسطة الطريقة Add والتي تأخذ ثلاث بارمترات هي:اسم العلاقة والعمودالأب Primary Key والعمود الابن Foreign Key.

ثم قمنا بالمرور عبر السجلات الموجودة في جدول الزبائن Customers باستخدام حلقة Foreach الأولى ومن أجل كل زبون (سجل ضمن جدول الزبائن) قمنا بطباعة جميع السجلات الأبناء المرتبطة بهِ في جدول الطلبياتOrders وذلك من خلال استدعاء الطريقة GetChildRows() الخاصة بالسجل الأب والتي تأخذ اسم العلاقة و ترد مجموعة (Collection) من كائنات DataRow يمكن طباعتها ضمن حلقة Foreach الثانية.

ملاحظة:

أحياناً نحتاج إلى الحصول على السجل الأب من خلال أحد السجلات الأبناء يمكن ذلك بواسطة الطريقة GetParentRow والتي تأخذ اسم العلاقة وترد السجل الأب وبعد ذلك يمكن التعامل معه كأي سجل عادي.

لنرى الخرج الآن:



دعم XML في ADO.NET

يتركز دعم XML في ADO.NET من خلال كائن DataSet حيث أنه يتضمن عدداً من الطرق من أهمها :

1- WriteXml: والذي يكتب كافة محتويات الDataSet ضمن ملف XML.

2- ReadXml: والذي يقوم بتحميل مستند XML ضمن كائن DataSet (ضمن الذاكرة) ليتم معالجتها في الذاكرة.

التعامل مع القيود Constraint

يملك كل جدول مجموعة من القيود تُعرّف القوانين الموضوعة على قيم الأعمدة تسمى هذه القوانين بشروط التكامل، وهي مجموعة من الشروط التي نقوم بتعريفها ضمن الجدول من أجل الحفاظ على صيغة نطامية للمعلومات. ولها ثلاثة أنواع:

1- تكامل وحدات المعطيات، مثلاً لا يمكن فتح حساب مصرفي لزبون دون معرفة عنوانه.

2- التكامل المرجعي، مثلاً: لا يمكن إجراء عمليات مصرفية على حساب قبل فتح حساب (تكامل سجل مع سجل أخر).

3- شروط التكامل المُعرّفة من قبل المستخدم، مثلاً : قيمة الراتب أكبر تماماً من الصفر.

مثلاً : يمكن أن نجعل عمود البريد الإلكتروني يأخذ قيماً فريدة Unique أو أن نجعل عموداً ما مفتاح أولي Primary Key أو Foreign Key أو أن يكون هذا العمود يقبل أو لا يقبل Null (قيمة خالية تعبر عن عدم وجود معلومة) أو غيرها من الشروط الأخرى.

ملاحظة: نفترض أن القارئ لديه معرفة جيدة عن تصميم قاعدة معطيات من المستوى المفاهيمي Conceptual Level إلى المستوى المنطقي، وحتى إنشاء الجداول والقيود عليها (شروط التكامل).

نعلم من خلال الأمثلة السابقة أن لكل كائن DataTable مجموعة من العلاقات من خلال الخاصية Relations، يملك كائن DataSet الخاصية EnforceConstraints والتي تأخذ قيمتين True,False في حال قمنا بوضع القيمة

False فأنه سيتجاهل كافة المخالفات لشروط التكامل (أي إذا كان العمود لا يقبل Null و وضعنا فيه Null وكانت قيمة هذه الخاصية False فأنه سيتجاهل مخالفة شروط التكامل)، وفي حال قمنا بمخالفة شروط التكامل وكانت قيمة الخاصية Enforce Constraints تساوي True عندئذٍ سيتولد استثناء من نوع ConstraintException.

لنرى المثال التالي:

```
private static void DemonstrateEnforceConstraints()
{
    // Create a DataSet with one table, one column and
    // a UniqueConstraint.
    DataSet dataSet = new DataSet("dataSet");
    DataTable table = new DataTable("table");
    DataColumn column = new DataColumn("col1");
    // A UniqueConstraint is added when the Unique
    // property is true.
    column.Unique = true;
    table.Columns.Add(column);
    dataSet.Tables.Add(table);
    Console.WriteLine("constraints.count: " +
    table.Constraints.Count);
      //**
      dataSet.EnforceConstraints = false;
  // عملية مخالفة شروط التكامل
    DataRow row;
    for (int i = 0; i < 5; i++)
    {
        row = table.NewRow();
        row["col1"] = 1;
        table.Rows.Add(row);
    }
```

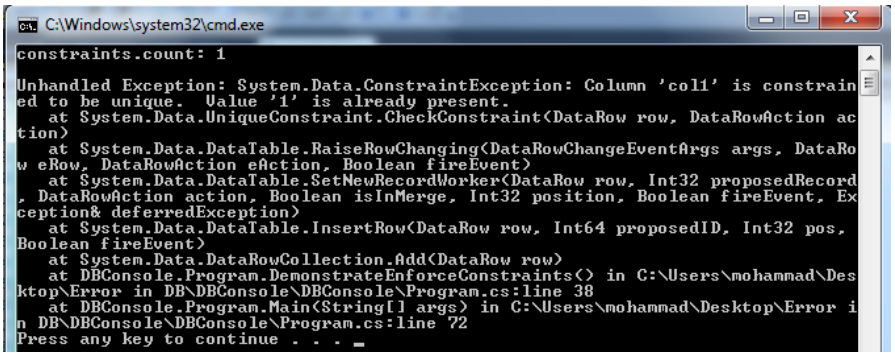
```
        table.AcceptChanges();
    }
    static void Main(string[] args)
    {
        DemonstrateEnforceConstraints();
    }
```

شرح الكود:

قمنا في البداية بإنشاء كائن مجموعة معطيات ثم قمنا بإنشاء جدول ثم ثمنا بإنشاء عمود ووضعنا عليه شرط unique (قيم العمود يجب أن لا تتكرر)، وفي حال قمنا بتكرار قيم هذا العمود يمكن أن نتجاهل مخالفة هذا الشرط عن طريق الخاصية EnforceConstraints الخاصة بكائن مجموعة المعطيات DataSet وعن طريق إسناد القيمة false له فأنه سيتجاهل شرط التكامل هذا (يجب وضع قيمة هذه الخاصية قبل أن نقوم بخرق شرط التكامل هذا)، وفي حال خالفنا شروط التكامل (القيود) وذلك من خلال وضع قيمة هذه الخاصية تساوي true فأنه سيتم رمي استثناء من نوع ConstraintException.

لنرى الخرج الآن في حال وضعنا للسطر ** القيمة false:

ملاحظة : تأخذ الخاصية EnforceConstraints القيمة True افتراضياً لذلك إذا أردنا مخالفة شروط التكامل فأنه يجب أن نضع السطر **(القيمة False) قبل عملية المخالفة وإلا (في حال وضع السطر ** بعد عملية المخالفة) فأنه سيتم رمي استثناء كما سنرى في الصورة التالية:



التعامل مع المناقلات Transactions

في الحقيقة مفهوم المناقلات ظهر مع وجود العمليات المصرفية حيث كنا بحاجة لتحقيق تزامن بين شخصين يريدان أن يجريا عمليات مصرفية على نفس الحساب.
المناقلة :هي مجموعة من التعليمات البرمجية والتي تحقق الخاصية ACID:

1- **A**tomicity: وتعني أن مجموعة التعليمات هي عبارة عن كتلة واحدة فإما أن يتم تنفيذها كاملة (تنفيذ جميع التعليمات) أو أن لا يتم تنفيذ أياً من هذه التعليمات.
2- **C**oherence: وتعني الترابط المنطقي أو الموائمة، أي أن العلاقات المنطقية بين المعطيات تبقى دلالتها صحيحة بعد نجاح المناقلة أو فشلها.
3- **I**solation: لا تتعلق نتيجة المناقلة بنتيجة مناقلة أخرى تتم على التوازي (أي أن إدارة التنافس على الموارد تقع على عاتق المبرمج).
4- **D**urability: ضمان استمرار التعديلات الناتجة عن تنفيذ المناقلة إلى ما بعد انتهائها وعدم العودة نهائياً للحالة السابقة (No RoolBack)، أي أن نتيجة المناقلة لا يمكن أن تتغير بعد انتهائها.

ملاحظة هامة: إن الخاصة الأولى يؤمنها لنا DBMS ولكن الخواص الثلاثة الأخيرة هي مسؤولية المبرمج.
لنرى المثال التالي: حيث أننا ننفذ مجموعة من التعليمات فإما أن تُنفذ جميعها أو أن لا تُنفذ ولا واحدة منها:

```
static void Main(string[] args)
{

        SqlConnection connection=new SqlConnection(@"Data
       Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\mohammad\Desktop\Northwind && pubs
       DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True");

            connection.Open();

             SqlCommand command = connection.CreateCommand();

             SqlTransaction transaction;

             // Start a local transaction.

             transaction = connection.BeginTransaction("SampleTransaction");

             // Must assign both transaction object and connection

             // to Command object for a pending local transaction

             command.Connection = connection;

             command.Transaction = transaction;

             try

             {

                 //First Statement
```

```
        command.CommandText =
                                "Insert into Customers (CustomerID, CompanyName) VALUES
            ('1', 'Alyan1')";

      // Execute DML Statement and Return the Number of Affected Rows
        command.ExecuteNonQuery();
        //Second statment
        command.CommandText =
                           "Insert into Customers (CustomerID, CompanyName) VALUES ('2',
      'Gnoom1')";

      // Execute DML Statement and Return the Number of Affected Rows
        command.ExecuteNonQuery();
        // Attempt to commit the transaction.
        transaction.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Commit Exception Type: {0}", ex.GetType());
        Console.WriteLine("  Message: {0}", ex.Message);

        // Attempt to roll back the transaction.
        try
        {
            transaction.Rollback();
        }
        catch (Exception ex2)
        {
```

```
            //This catch block will handle any errors that may have occurred
              // on the server that would cause the rollback to fail, such as
              // a closed connection.
                Console.WriteLine("Rollback Exception Type: {0}",
        ex2.GetType());
                Console.WriteLine("  Message: {0}", ex2.Message);
            }
        }
}
```

شرح الكود:

في البداية قمنا بإنشاء كائن اتصال ثم قمنا ثم قمنا بإنشاء أمر Command خاص بالاتصال الذي قمنا بإنشائه، ثم قمنا بإنشاء كائن مناقلة باستخدام الطريقة BeginTransaction، الخاصة بكائن الاتصال والتي تعيد مرجع Reference على كائن SqlTransaction والتي تبدأ عملية المناقلة وبما أننا نريد أن ننفذ مجموعة من الأوامر على شكل مناقلة فأنه يجب أن نحدد للكائن Command كائن الاتصال الذي سوف يرتبط به بالإضافة إلى كائن المناقلة الذي سوف يتحكم بالعمليات ويديرها.

ثم قمنا بتحديد أول تعليمة وهي تعليمة DML (insert,Update,Delete) وهذه العمليات لا ترد بيانات بطبيعتها وإنما ترد عدد الأسطر التي تأثرت بالتعليمة لذلك استخدمنا الطريقة ExecuteNonQuery وهي تنفيذ لعملية DML وليس استعلام عن البيانات (لهذا السبب سميت NoneQuery)، ونفس الأمر من أجل التعليمة الثانية ويمكننا إضافة أي عدد من التعليمات، ثم قمنا بعملية تثبيت Commit والتي تعني تثبيت نتيجة عمليات المناقلة.

في حال حدوث استثناء Exception قبل انتهاء المناقلة (مثلاً التعليمة الثانية خاطئة) عندئذ سيتم إمساك الاستثناء عن طريق الكتلة Catch، ومن ثم القيام استدعاء الطريقة RoolBack() والتي تعود لبداية المناقلة وبالتالي التراجع عن كافة العمليات التي قمنا بها (لا يتم تنفيذ التعليمة الأولى) أو يمكن التراجع إلى نقطة محددة من المناقلة قمنا بإنشائها تُسمى SavePoint، سنرى ذلك من خلال المثال التالي:

```
static void Main(string[] args)

{

        SqlConnection connection=new SqlConnection(@"Data
        Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\mohammad\Desktop\Northwind && pubs
        DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True");

            connection.Open();

             SqlCommand command = connection.CreateCommand();

             SqlTransaction transaction;

             // Start a local transaction.

             transaction = connection.BeginTransaction("SampleTransaction");

             // Must assign both transaction object and connection

             // to Command object for a pending local transaction

             command.Connection = connection;

             command.Transaction = transaction;

             try

             {

                 //First Statement

                 command.CommandText =

                     "Insert into Customers (CustomerID, CompanyName) VALUES ('1'

                                  , 'Alyan1')";

                 command.ExecuteNonQuery();

                 //Second statment

            command.CommandText ="Insert into Customers (CustomerID, CompanyName) VALUES ('2',
'Gnoom1')";

                 command.ExecuteNonQuery();


               // حفظ نقطة مرجعية يمكن التراجع إليها بدل العودة إلى بداية المناقلة

               transaction.Save("s1");


                //Third statement Generate Exception
```

```
        command.CommandText =
"Insert into Customers (CustomerID, CompanyName4) VALUES ('3', 'Hammood')";
        command.ExecuteNonQuery();

        // Attempt to commit the transaction.
        transaction.Commit();
        Console.WriteLine("All records are written to database.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Commit Exception Type: {0}", ex.GetType());
        Console.WriteLine("  Message: {0}", ex.Message);
        // Attempt to roll back the transaction.
        try
        {
             //RoolBack To SavePoint
            transaction.Rollback("s1");
             // تثبيت المناقلة وبالتالي تنفيذ أول تعليمتين
             transaction.Commit();
        }
        catch (Exception ex2)
        {
        // This catch block will handle any errors that may have occurred
         // on the server that would cause the rollback to fail, such as
            // a closed connection.
          Console.WriteLine("Rollback Exception Type: {0}", ex2.GetType());
           Console.WriteLine("  Message: {0}", ex2.Message);
        }
```

شرح الكود:
نفس المثال السابق ولكننا قمنا بتنفيذ أول تعليمتين ثم قمنا بحفظ نقطة استعادة مرجعية باسم "s1" باستخدام الطريقة save("s1") الخاصة بكائن المناقلة، والآن عند تنفيذ التعليمة الثالثة سيحدث خطأ وسيرمي استثناء لأن اسم العمود غير موجود بالجدول وسننتقل إلى الكتلة Catch حيث سنقوم بعملية تراجع RoolBack إلى النقطة "s1" ومن ثم نقوم بتثبيت المناقلة عن طريق التعليمة Commit، ومن دون هذه التعليمة Commit لا يتم تثبيت المناقلة (وبالتالي لا يتم تنفيذ أول تعليمتين).
في حالة مثالنا عند حدوث خطأ في التعليمة الثالثة ننتقل إلى الكتلة Catch حيث يتم التراجع إلى النقطة المرجعية "s1" والتي أُنشئت بعد التعليمة الثانية وبالتالي عند تنفيذ التعليمة ;transaction.Rollback("s1") سيتم العودة إلى نهاية التعليمة الثانية وعند تثبيت المناقلة بواسطة التعليمة transaction.Commit() تكون التعليمة الأولى والثانية تمَ تنفيذهما بنجاح.
من المهم جداً معرفة أهمية المناقلات، وهي مُستخدمة في كثير من الأماكن وقد حلت بعض مشاكل التنافس على الموارد لهذا السبب يجب الاهتمام بها.

أحداث ADO.NET

الأحداث هي الآلية التي تستخدمها الكائنات لإرسال المعلومات الهامة إليك وإذا اخترت عدم وضع شيفرة لهذه الأحداث فإنك قد تضيع فرصة مهمة للاستماع إلى بعض التنبيهات المهمة.
يمكنك في ADO.NET العمل مع الأحداث التي تصدرها كائنات الاتصال Connection و DataAdapter و DataSet بجميع عناصرها (DataTable وDataColumn و DataRow).
أن العمل مع البيانات هو عملية دقيقة وقد يحدث العديد من المشاكل، لذلك عليك أن تكون حذراً لتجنب الحصول على مشاكل إضافية، مثلاً: قد يتسبب الاتصال بمخزن البيانات في بعض الأحيان في توليد خطأ (بسبب عطل في الشبكة أو عطل HardWare)، لذلك يقدم لنا الكائن DataAdapter الحدث FillError والذي يتم قدحه Fire عند حدوث خطأ في ملئ مجموعة البيانات DataSet يمكننا في هذه الحالة من معالجة الخطأ عندما يحدث.
إنّ إضافة بعض الشيفرة البرمجية إلى هذا الحدث لمعالجة الخطأ أو لإعادة تنفيذ التابع Fill هو فكرة عظيمة إذا لم تكن متأكداً من أن مصدر البيانات متاحاً لك في أي وقت.
يمكننا إضافة الأسطر التالية إلى الشيفرة الخاصة بنا:

```
DataSet ds = new DataSet();

SqlDataAdapter adapter = new SqlDataAdapter();

adapter.FillError += new FillErrorEventHandler(adapter_FillError)
```

وتتم المعالجة ضمن الطريقة adapter_FillError كما يلي:

```
static void adapter_FillError(object sender, FillErrorEventArgs e)

 {

            //sender is the Object that Fire the Event

            ((SqlDataAdapter)sender).Fill(ds, "Table1");

            //e is information about Error

}
```

تُقدم مجموعة البيانات DataSet ومكوناتها العديد من الأحداث المفيدة منها مثلاً عندما يتغير محتوى السطر يتم قدح حدث أو عندما يتم حذف السطر ونفس الأمر مع الأعمدة يوجد أحداث خاصة بالتعديل والحذف وهنا نشاهد نسختين من الأحداث أحدهما ينتهي ب "ing"، مثل الحدث ds.Tables["t1"].RowChanging ويعني أنه عندما نقوم بتغيير محتوى سطر مثلاً فأنه قبل تغيير محتوى السطر يتم قدح الحدث واستدعاء التابع الخاص به ومعالجة هذه التغيرات (أي أن هذه الأحداث التي تنتهي ب "ing" تحصل قبل أن يتم حدوث الفعل)، على عكس ذلك يوجد أحداث تنتهي ب "ed" هذه الأحداث تحصل (تُقدَح) بعد الانتهاء من الفعل مثل ds.Tables["t1"].RowChanged، يتم قدح هذا الحدث بعد أن يتم تغيير محتوى السطر.
والخلاصة هي أنه إذا أردت اختبار قيمة ما ومنع حدوث التغيير أو التعديل عليك استخدام الأحداث التي تنتهي ب "ing"، أما إذا أردت التعامل مع التعديلات بعد أن يتم إجراؤها فعليك استخدام الأحداث التي تنتهي ب "ed".

الإجراءات المخزنة Stored Procedures

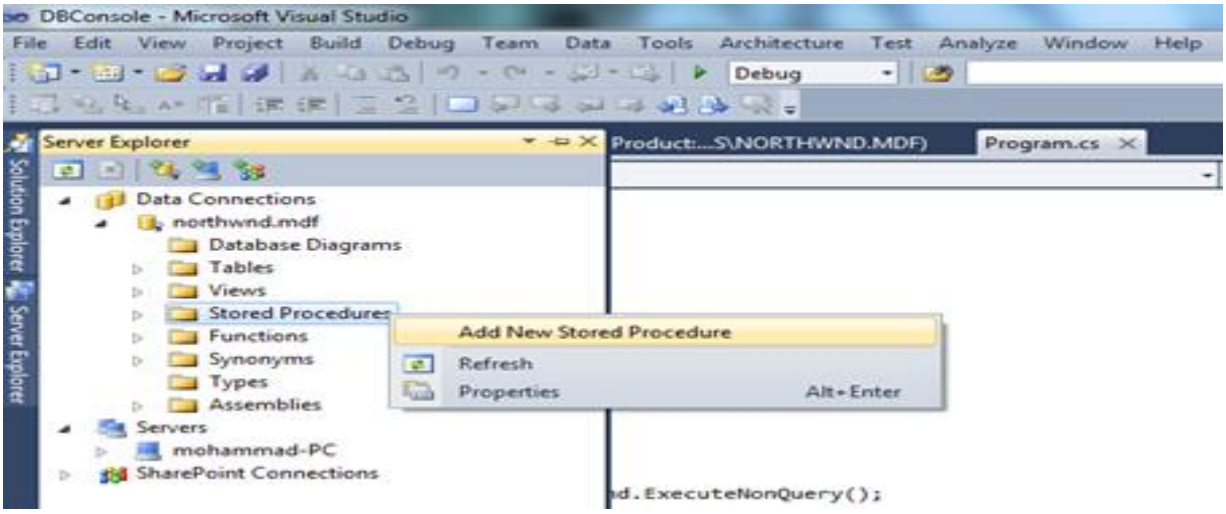
الإجراء المُخزن Stored Procedure هو قسم من الشيفرة يتم تخزينه ضمن قاعدة المعطيات نفسها أي تعمل في جانب المخدّم Server -Side Processing.
يمكن لتطبيقات تعمل خارج قاعدة المعطيات قدح إجراء مخزن باستخدام مقدار بسيط من الشيفرة والاتصالات الشبكية،يمكن إنشاء إجراء مخزن يقوم بإضافة أو حذف أو تعديل سجل موظف في جدول الموظفين، في حال قمنا بعملية حذف باستخدام إجراء مخزن فأننا نحتاج إلى رقم تعريف الموظف. من دون الإجراءات المخزنة عليك تنفيذ أمر Delete من SQL والذي قد يحتاج إلى أكثر من معلومة يتم إرسالها عبر الشبكة. وبما أنه موجود على المخدّم Server فأن أي شخص يستطيع استدعائه بدلاً من إعادة كتابته من جديد.
بالإضافة لمحدودية اتصالات الشبكة، فأن تنفيذ الإجراءات المخزنة هو أكثر فاعلية من أوامر SQL وذلك لعدة أسباب منها:

1- أن عبارة SQL تُرسل إلى مخدم قاعدة المعطيات ليقوم بترجمتها ومن ثم تنفيذها، بينما يتم حفظ الإجراءات المخزنة في حالة مترجمة Compiled جاهزة للتنفيذ لحظة تبليغها.
2- كلما كبرت وتعقدت مهام قاعدة بياناتك، كلما زادت الفوائد التي ستحصل عليها من استخدام الإجراءات المخزنة.

يمكن أن نحتاج إلى توليد استعلام ديناميكياً يمكن القيام بذلك في حال كانت عبارات SQL الخاصة بك مستقرة إلى حد ما أو تحوي بعض الوسائط المتغيرة والتي يمكن تمريرها إلى الإجراء المخزن.
يمكن أن تعطي الإجراءات المخزنة خرج (بيانات إلى المُستدعي) ويمكن أن لا تعطي أي خرج مثل حذف سجل موظف، كما يمكن تمرير بارامترات إلى الإجراء المخزن.

كتابة إجراء مُخزن في قاعدة البيانات

يمكن كتابة إجراء مُخزن ضمن قاعدة المعطيات وذلك من خلال الأمر الذي تشرحه الصورة التالية:



ثم نكتب هذا الإجراء المُخزن والذي يقوم بإضافة منتج إلى جدول المنتجات ويأخذ متحولات تُمثل قيم جميع الأعمدة كبارمترات ماعدا رقم المنتج لأنه يتم توليده تلقائياً (Auto Generated) ويرد رقم المنتج.
لنرى هذا الإجراء المخزن وهو مكتوب بلغة PL/SQL.
التعليقات كافية لفهم المحتوى.

```
CREATE PROCEDURE  dbo.AddProduct

--@  is Mandatory With the Parameters. all this is Parameters for

--Stored Procedure

(

  @MyProductName nvarchar(40),

  @MySupplierID int,

  @MyCategoryID int,

  @MyQuantityPerUnit nvarchar(20),

  @MyUnitPrice money,

  @MyUnitsInStock smallint,

  @MyUnitsOnOrder smallint,

  @MyReorderLevel smallint,

  @MyDiscontinued bit

)

AS

DECLARE @ProductID int


  -- insert a row into the Products table

  INSERT INTO Products (

    ProductName, SupplierID, CategoryID, QuantityPerUnit,

    UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel,

    Discontinued

  ) VALUES (
```

```
    @MyProductName, @MySupplierID, @MyCategoryID, @MyQuantityPerUnit,
    @MyUnitPrice, @MyUnitsInStock, @MyUnitsOnOrder, @MyReorderLevel,
    @MyDiscontinued
  )
  -- use the @@IDENTITY function to get the last inserted
  -- identity value, which in this case is the ProductID of
  -- the new row in the Products table
  SET @ProductID = @@IDENTITY
  -- return the ProductID
  RETURN @ProductID
```

يُمكننا تنفيذ هذه الإجراء المُخزن من داخل SQL Server كما يلي:

```
DECLARE @MyProductID int
EXECUTE @MyProductID = AddProduct 'Widget', 1, 1, '1 Per box', 5.99, 10, 5, 5, 1
PRINT @MyProductID
```

كما يمكن تعريف متحولات وتمريرها للإجراء بدلاً من الثوابت:

```
--تعريف متحولات
DECLARE @MyProductID int
DECLARE @MyProductName nvarchar(40)
DECLARE @MySupplierID int
DECLARE @MyCategoryID int
DECLARE @MyQuantityPerUnit nvarchar(20)
DECLARE @MyUnitPrice money
DECLARE @MyUnitsInStock smallint
DECLARE @MyUnitsOnOrder smallint
DECLARE @MyReorderLevel smallint
DECLARE @MyDiscontinued bit
```

```
-- تحديد قيم المتحولات

SET @MyProductName = 'Wheel'

SET @MySupplierID = 2

SET @MyCategoryID = 1

SET @MyQuantityPerUnit = '4 per box'

SET @MyUnitPrice = 99.99

SET @MyUnitsInStock = 10

SET @MyUnitsOnOrder = 5

SET @MyReorderLevel = 5

SET @MyDiscontinued = 0

-- EXECUTE استدعاءالإجراء المُخزن من خلال الأمر

EXECUTE @MyProductID = AddProduct @MyProductName,

  @MySupplierID, @MyCategoryID, @MyQuantityPerUnit,

  @MyUnitPrice, @MyUnitsInStock, @MyUnitsOnOrder,

  @MyReorderLevel, @MyDiscontinued


-- طباعة النتيجة

PRINT @MyProductID
```

استخدام الإجراءات المُخزنة في ADO.NET

- يمكن تنفيذ إجراء مُخزن في قاعدة المعطيات باستخدام كائن Command.
- تحتاج بعض الإجراءات المخزنة إلى بارامترات وبعضها لا يحتاج ذلك.
- سنقوم الآن بتنفيذ إجراء مُخزن يقوم بإضافة سطر إلى جدول المنتجات Products (ضمن قاعدة المعطيات الشهيرة NorthWind) .

```
static void Main(string[] args)

{

      //1: Connection To DB

      SqlConnection conn = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\mohammad\Desktop\Northwind && pubs
DataBases\northwnd.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True");

     //2

     SqlCommand commmand = new SqlCommand("AddProduct", conn);

     //3: this Command is Calling Stored Procedure

      commmand.CommandType = CommandType.StoredProcedure;

     //4: Open Connection

       conn.Open();

     //5: if the Stored Procedure has a parameters you Must be  Add Paramerters

     SqlParameter MyProductName = commmand.Parameters.Add("@MyProductName"

              , SqlDbType.NVarChar);

        MyProductName.Value = "Computer";

     SqlParameter MySupplierID = commmand.Parameters.Add("@MySupplierID", SqlDbType.Int);

     MySupplierID.Value = 1;

    SqlParameter MyCategoryID = commmand.Parameters.Add("@MyCategoryID", SqlDbType.Int);

     MyCategoryID.Value = 2;

    SqlParameter MyQuantityPerUnit = commmand.Parameters.Add("@MyQuantityPerUnit"

              , SqlDbType.NVarChar);

    MyQuantityPerUnit.Value = "units";

   SqlParameter MyUnitPrice = commmand.Parameters.Add("@MyUnitPrice", SqlDbType.Money);

    MyUnitPrice.Value = 150;
```

```
  SqlParameter MyUnitsInStock = commmand.Parameters.Add("@MyUnitsInStock"
      , SqlDbType.SmallInt);
   MyUnitsInStock.Value = 22;
   SqlParameter MyUnitsOnOrder = commmand.Parameters.Add("@MyUnitsOnOrder"
                    , SqlDbType.SmallInt);
   MyUnitsOnOrder.Value = 22;
  SqlParameter MyReorderLevel = commmand.Parameters.Add("@MyReorderLevel"
              , SqlDbType.SmallInt);
  MyReorderLevel.Value = 22;


  SqlParameter MyDiscontinued = commmand.Parameters.Add("@MyDiscontinued"
              , SqlDbType.Bit);
    MyDiscontinued.Value = 0;
   //Execute Stored Procedure
     int record = commmand.ExecuteNonQuery();
}
```

شرح الكود:

في هذا المثال قمنا بإنشاء كائن اتصال ومن ثم قمنا بإنشاء كائن أمر وحددنا له كائن الاتصال الذي سوف يرتبط معه بالإضافة إلى اسم الإجراء المخزن، ثم حددنا نوع الأمر الذي سوف ينفذه كائن الأمر Command وهو Stored Procedure، ثم قمنا بفتح الاتصال مع قاعدة المعطيات وبما أن للإجراء المخزن عدد من البارمترات فيجب أن نمرر كل هذه البارمترات إلى الإجراء المخزن عن طريق كائن الأمر حيث يجب توخي الدقة والحذر في التعامل مع الأنماط (يجب أن يكون لدى الشخص الذي سوف يستدعي الإجراء معرفة مُسبقة عن أنواع البارمترات الخاصة بالأعمدة)، وفي حال لم يكن للإجراء المخزن بارمترات فلا نضيف أي بارمترات.

ثم بعد أن نقوم بإنشاء البارمتر عن طريق الخاصية Parameters الخاصة بكائن الأمر وباستخدام الطريقة Add نحدد اسم البارمتر (يجب أن يكون له نفس اسم البارمتر في الإجراء المُخزن) و نوعه من خلال الأنماط الخاصة ب SQL Server.

بعد ذلك قمنا بتنفيذ (استدعاء) الإجراء من خلال الكائن Command وباستخدام الطريقة ExecuteNonQuery قمنا بتنفيذ الإجراء.تقوم هذه الطريقة بإعادة عدد الأسطر التي تأثرت عند تنفيذ العملية وبما أننا نضيف سطر فأنه سيرد لنا رقم 1 دوماً.

في الحقيقة يمكن استخدام الطريقة ExecuteReader ولكن لن ترد أي أسطر لأنها عملية (insert) وفي حال كان الإجراء المُخزن يرد مجموعة من الأسطر فأننا ننفذ الطريقة ExecuteReader وباستخدام حلقة While نحصل على كامل النتائج (كما في الأمثلة السابقة).
يُمكننا مشاهدة السطر الذي قمنا بإضافته من خلال تحديد جدول المنتجات Product من خلال ال Server Explorer، ثم من خلال الزر الأيمن للفأرة نختار الأمر Show Table Data.

القادح Trigger

القادح هو مجموعة من التعليمات المُخزنة داخل في قاعدة المعطيات التي يُنفذها نظام إدارة قواعد المعطيات DBMS آلياً عند حدوث حدث معين (insert,update,delete,...) على جدول معين ضمن قاعدة المعطيات.

- تُسمى عملية تنفيذ القادح "Firing".
- يتم تنفيذ القادح "Firing" قبل Before أو بعد After أو عوضاً عن Instead of حدث معين (أحداث القادح هي عمليات DML أو DDL).
- يكون القادح ذو أهمية كبيرة في حال كنا نريد مراقبة أي عمليات أو أي تغييرات تتم على جدول معين والقيام بعمليات قبل أو بعد أو عوضاً هذا الحدث.

مثلاً: يمكننا كتابة قادح على جدول المنتجات، حيث يتم تنفيذ هذا القادح بعد عملية إضافة سطر في جدول المنتجات يمكن أن نكتب في جدول تدقيق المنتجات ProductAudit وهو جدول موجود لدينا يحوي جميع العمليات التي تمت على جدول المنتجات Products من إضافة أو حذف أو تعديل لقيم أحد الأعمدة.

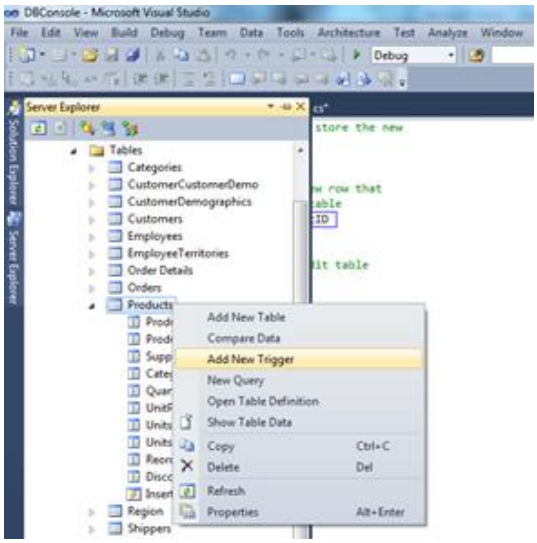
الآن لنرى القادح التالي والذي سنقوم بتعريفه على جدول المنتجات Products، والذي يقوم بإضافة سطر إلى جدول ProductAudit والذي يحوي على توصيف لجميع العمليات التي تمت على الجدول Products وذلك بعد كل عملية إضافة أو حذف أو تعديل لقيمة العمود UnitPrice. لنبدأ الآن بأول قادح:
يقوم هذا القادح واسمه InsertProductTrigger بإضافة سجل إلى الجدول ProductAudit (يحوي رقم السجل الذي تم إضافته إلى جدول المنتجات)، وذلك بعد إضافة سجل في جدول المنتجات Products.
قبل أن نشاهد القادح علينا أن نقوم بإنشاء الجدول ProductAudit إلى قاعدة بيانات Northwind كما يلي:

```
/*
  this statements creates a table that is used to
  store the results of triggers that audit modifications
  to the Products table
*/
USE Northwind
CREATE TABLE ProductAudit (
  ID int IDENTITY(1, 1) PRIMARY KEY,
  Action nvarchar(100) NOT NULL,
  PerformedBy nvarchar(15) NOT NULL DEFAULT User,
  TookPlace datetime NOT NULL DEFAULT GetDate()
)
```

- تقوم التعليمة IDENTITY(1, 1) بتوليد المفتاح الأولي Primary Key للعمود ID بشكل أوتوماتيكي

(تلقائي) .قيم العمود تبدأ ب 1 وتزداد في كل مرة بمقدار 1 بعد كل عملية إضافة Insert على جدول المنتجات.

- العمود Action يحوي توصيف للعملية التي قمنا بها على جدول المنتجات Products (إضافة أو حذف أو تعديل على العمود UnitPrice في جدول المنتجات).
- الأعمدة الباقية تأخذ قيمة يتم تحديدها افتراضياً في حال كانت قيمة العمود تساوي Null.
- الآن سنقوم بكتابة القادح InsertProductTrigger ويتم من خلال اختيار الأمر كما في الصورة التالية:



ثم نقوم بإضافة الشيفرة التالية:

```
/*
  InsertProductTrigger Trigger creates a trigger that fires
  after an INSERT statement is performed on the
  Products table
*/

Create TRIGGER  InsertProductTrigger
ON Products
AFTER INSERT
AS
  -- don't return the number of rows affected
  -- That it  Improves Performance
  SET NOCOUNT ON

  -- declare an int variable to store the new
  -- ProductID
  DECLARE @NewProductID int

  -- get the ProductID of the new row that
  -- was added to the Products table
  SELECT @NewProductID = ProductID
  FROM inserted
```

```
  -- add a row to the ProductAudit table
  INSERT INTO ProductAudit (
    Action
  ) VALUES (
    'Product added with ProductID of ' +
      CONVERT(nvarchar, @NewProductID)
  )
```

الآن وبعد إضافة Insert سجل في جدول المنتجات Products وذلك عن طريقة تعليمة sql أو عن طريق تنفيذ الإجراء المخزن AddProduct الذي قمنا بشرحه في المثال السابق، وبعد After (كما حددنا في تعريف القادح) تنفيذ الإجراء المخزن AddProduct (أي تعليمةInsert) يتم إضافة سجل في الجدول ProductAudit.
يُمكننا مشاهدة السطر الذي قمنا بإضافته من خلال تحديد جدول المنتجات ProductAudit من خلال ال Server Explorer، ثم من خلال الزر الأيمن للفأرة نختار الأمر Show Table Data.

ملاحظات هامة:

- يمكننا تنفيذ القادح قبل عملية الإضافة على جدول المنتجات وذلك بأن نضع Before بدلاً من After أو يمكن تنفيذ القادح بدلاً من عملية الإضافة وذلك بأن نضع Instead Of في تعريف القادح.
- كل ما سنقوم به في التمارين القادمة هو أن نحدد أن القادح سيقوم بعملية إضافة سجل إلى الجدول ProductAudit بعد عملية الحذف Delete (هي حدث القادح كما ذكرنا سابقا) أو بعد عملية تعديل عمود UnitPrice في جدول المنتجات.

القادح الخاص بعملية الحذف

نضيفه بنفس الطريقة السابقة ونسميه DeleteProductTrigger، كل ما قمنا به هنا هو تغيير الحدث الخاص بالقادح وهو أنه يجب أن ينفذ القادح بعد عملية الحذف AFTER DELETE.

```
/*
  DeleteProductTrigger creates a trigger that fires
  after a DELETE statement is performed on the
  Products table
*/
Crate TRIGGER DeleteProductTrigger
ON Products
AFTER DELETE
AS

  -- don't return the number of rows affected
  SET NOCOUNT ON

  -- declare an int variable to store the
  -- ProductID
  DECLARE @NewProductID int

  -- get the ProductID of the row that
  -- was removed from the Products table
  SELECT @NewProductID = ProductID
```

```
  FROM deleted

  -- add a row to the ProductAudit table
  INSERT INTO ProductAudit (
    Action
  ) VALUES (
    'Product #' +
      CONVERT(nvarchar, @NewProductID) +
      ' was removed'
  )
```

الآن وبعد حذف سجل من جدول المنتجات، يتم إضافة سجل إلى الجدول ProductAudit يحوي توصيف لعملية الحذف التي قمنا بها، كما ويمكن مشاهدة السجل التي قام القادح بإضافته إلى الجدول ProductAudit كما تكلمنا سابقاً.

القادح الخاص بعملية تعديل عمود "UnitPrice"

نضيفه بنفس الطريقة السابقة ونسميه UpdateUnitPriceProductTrigger، يتم تنفيذ هذا القادح بعد أن يتم تعديل قيمة العمود UnitPrice في جدول المنتجات ويجب أن تكون نسبة التخفيض أكبر من 25% حتى يتم إضافة سجل في الجدول ProductAudit.

```
/*
  UpdateUnitPriceProductTrigger creates a trigger
  that fires after an UPDATE statement is performed on the
  the UnitPrice column of the Products table.
  If the reduction of the unit price of a product is
  greater than 25% then a row is added to the ProductAudit table
  to audit the change.
*/

ALTER TRIGGER UpdateUnitPriceProductTrigger
ON Products
AFTER UPDATE
AS

  -- don't return the number of rows affected
  SET NOCOUNT ON

  -- only run the code if the UnitPrice column
  -- was modified
IF UPDATE(UnitPrice)
BEGIN

  -- declare an int variable to store the
  -- ProductID
  DECLARE @MyProductID int

  -- declare two money variables to store the
  -- old unit price and the new unit price
  DECLARE @OldUnitPrice money
  DECLARE @NewUnitPrice money
```

```
-- declare a float variable to store the price
-- reduction percentage
DECLARE @PriceReductionPercentage float

-- get the ProductID of the row that
-- was modified from the inserted table
SELECT @MyProductID = ProductID
FROM inserted

-- get the old unit price from the deleted table
SELECT @OldUnitPrice = UnitPrice
FROM deleted
WHERE ProductID = @MyProductID

-- get the new unit price from the inserted table
SELECT @NewUnitPrice = UnitPrice
FROM inserted

-- calculate the price reduction percentage
SET @PriceReductionPercentage =
  ((@OldUnitPrice -@NewUnitPrice) / @OldUnitPrice) * 100

-- if the price reduction percentage is greater than 25%
-- then audit the change by adding a row to the PriceAudit table
IF (@PriceReductionPercentage > 25)
BEGIN
  -- add a row to the ProductAudit table
  INSERT INTO ProductAudit (
    Action
  ) VALUES (
    'UnitPrice of ProductID #' +
      CONVERT(nvarchar, @MyProductID) +
    ' was reduced by ' +
      CONVERT(nvarchar, @PriceReductionPercentage) +
      '%'
    )
  END
END
```

الآن نُعدل قيمة العمود UnitPrice وبشرط أن تكون نسبة التخفيض أكبر من 25%، وعند ذلك يتم إضافة سجل إلى الجدول ProductAudit يوصف عملية التعديل.

الفصل السابع

نشر التطبيقات

Deployment

مقدمة

يُعتبر نشر التطبيقات Deployment أحد أهم المزايا العظيمة التي يقدمها NET Framewrork.، و تعني تثبيت التطبيقات Installationعلى أنظمة المستخدمين، يعتبر نشر التطبيقات جزءاً من دورة حياة المنتج Software life Cycle وله مخططات خاصة به وهي مُقدمة من UML.

سابقاً كان يمكن نشر التطبيقات باستخدام التعليمة xcopy التي يقدمها DOS، فالمجمعات البسيطة (Assemblies ويقصد بها exe,dll) تحوي عدداً من الملفات، ولم نعد بحاجة لحفظ إعدادات المجمعة ضمن مسجل النظام registry (كما كان سابقاً مع مكونات COM).وبالتالي فإن عملية نسخ ملفات التطبيق إلى القرص الصلب كافية لتثبيته على جهاز المستخدم.

إذا سبق وقمت بتصميم تطبيقاتك من قبل، فأنك ستدرك أهمية هذه الميزة التي يقدمها NET. لمبرمجي التطبيقات. خصوصاً أن عملية نشرالتطبيقات سابقاً كانت تتطلب وضع عناصر التحكم وملفات الربط الديناميكي والمكونات في مسجل النظام وهذا يولد الكثير من المشاكل المتعلقة بالإصدارات والتوافقية وغيرها من المشاكل، هذه المشاكل غير موجودة في التطبيقات المكتوبة بلغات ضمن منصة NET..

لذلك قدمت شركة Microsoft البرنامج Windows Installer لتسهل على المبرمج نشر تطبيقاته بعد تصميمها باستخدام هذا البرنامج المعياري.

ما هي عملية نشر التطبيقات Deployment

نقصد بعملية النشر Deployment هي تثبيت وإعداد التطبيقات على أنظمة الهدف.

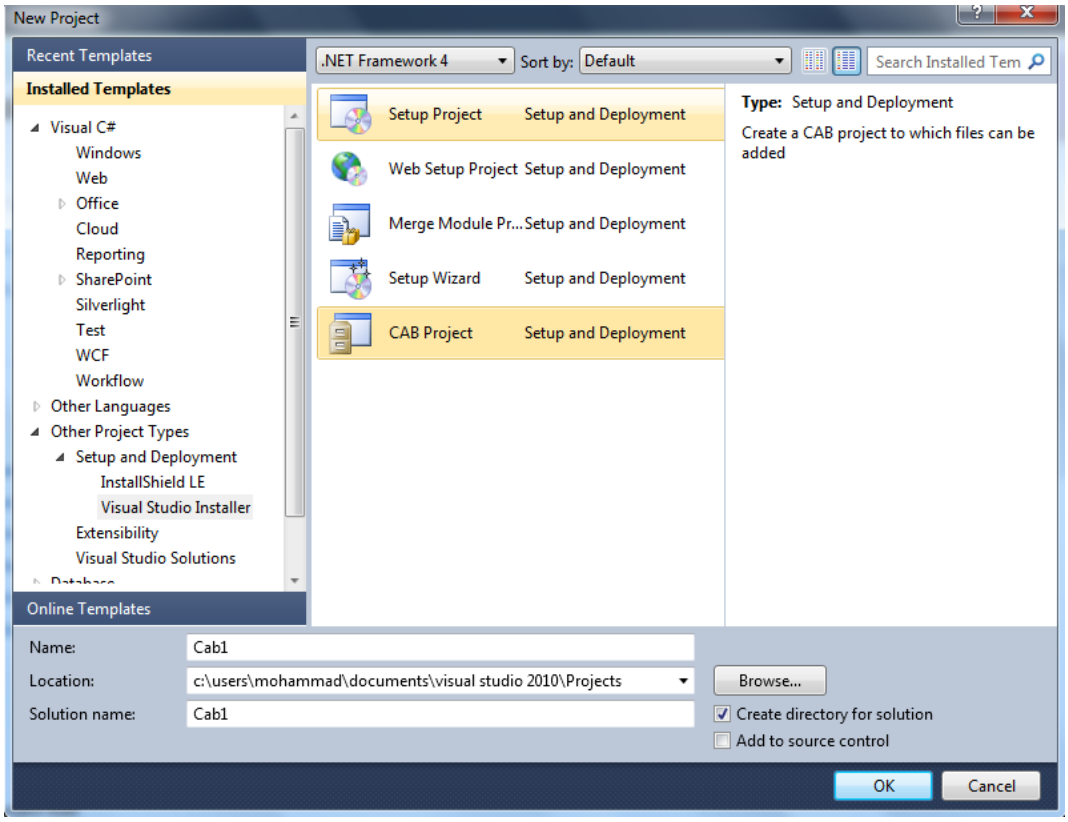
تُعَرّفْ نموذج عملية التطوير في إطار عمل حلول شركة Microsoft دورة الحياة لعملية التطوير وفقاً لأربعة أطوار (مراحل) Phases:

1- **طور الرؤية Envisioning:** حيث يتم في هذا الطور (أو المرحلة) تحديد معالم التطبيق بوضوح.
2- **طور التخطيط Planning :** يُستخدم لتحليل وتصميم الحل.
3- **طور التطوير Deployment:** يمثل أكبر هذه المراحل (الأطوار).
4- **طور الإستقرار Stabilizing:** والذي يتم فيه إصلاح المشاكل والأخطاء ومنها يتم إرسال الإصدارات التجريبية والنهائية Release إلى الزبون.

وبناءً على هذا النموذج يتم بناء التطبيقات.

أنواع مشاريع النشر

نختار Add New Project في Visual Studio.NET من خلال القائمة File ثم نرى الشكل التالي:



يمكننا إنشاءعدة أنواع من المشاريع منها :

1- **القالب Cab Project :** ويمكننا باستخدامه إنشاء ملفات حافظة (Cabinet)، يمكن أن تُستخدم هذه الملفات الحافظة لدمج عدد من المجمعات ضمن ملف واحد وضغط هذا الملف.
2- **القالب Setup Project**: وهو القالب الذي سنستخدمه هنا، يُستخدم هذا النوع من المشاريع لأنشاء حزمة Windows Installer وبالتالي فهو القالب المناسب لنشر تطبيقات Windows (وهو محور حديثنا في هذا الفصل).
3- **المعالج Setup Wizard:** وهو عبارة عن برنامج مساعد لإختيار أحد القوالب السابقة لإنشاء تثبيت تطبيق أو لإنشاء حزمة موزعة.

سنتلكلم عن Setup Project لإنشاء مشاريع Windows Installer.

لكن يجب ان نعرف ما هو Windows Installer في البداية.

ما هو Windows Installer؟

هو عبارة عن خدمة تتولى عمليات تثبيت Installation و تحديث Update وإصلاح repair و حذف Remove التطبيقات على مختلف أنظمة Windows، وهي جزء من النظام Windows وتختلف إصداراتها بإختلاف إصدارات Windows.

Windows Installer يتعقب تنصيب التطبيقات ويترك أثراً لها في قاعدة معطيات خاصة به وعند إزالة تثبيت التطبيق Uninstall يمكن بسهولة تتبع وحذف الإعدادات الموجودة في مسجل النظام registry والتي تكون قد أُضيفت أثناء تثبيت التطبيق، بالإضافة إلى حذف الملفات الخاصة بالتطبيق والموجودة على القرص الصلب مع جميع الملفات المرتبطة بالتطبيبق و الموجودة في سطح المكتب وقائمة أبدأ، في حال كان أحد الملفات التي سيقوم Installer Windows بحذفها مرتبطة بتطبيق أخر عندها لا يتم حذف هذا الملف وبالتالي لا تتوقف هذه التطبيقات عن العمل.

قاعدة المعطيات الخاصة Windows Installer تُمكنه من القيام بعملية إصلاح repair للأخطاء في حال تلف إعدادات مسجل النظام registry أو في حال تلف أو حذف ملف dll يمكن القيام بعملية إصلاح، ويتم ذلك بالإستعانه بقاعدة البيانات الخاصة به.

إن عملية تطوير التطبيقات باستخدام Visual Studio .NET تعطينا القدرة على إنشاء حزم تثبيت لتطبيقات Windows وهذا ما سنتكلم عنه في الفقرات القادمة.

إصطلاحات Window Installer

علينا أن نتعامل مع عدد من المصطلحات والمفاهيم عند العمل مع Windows Installer، مثل: الحزم Packages والمزايا Features والمكونات Components.

تُمثل الحزمة قاعدة بيانات MSI إختصاراً ل Microsoft Installer، أما الميزة فهي ما سيعرض للمستخدم من إمكانيات التطبيق ويمكن أن تتألف الميزة من عدة مزايا أو مكونات للتطبيق. أما المكوّن فهو إمكانيات جزء من التطبيق من وجهة نظر المطور. لقد تم التفريق بين المزايا والمكونات وذلك لأن المكون الواحد يمكن أن يتضمن أكثر من ميزة ولكن الميزة لا توجد ضمن أكثر من ميزة.

مميزات Windows Installer

يقدم لنا Windows Installer العديد من المزايا منها:

1- يمكن تثبيت المزايا ويمكن الإعلان عنها أو إزالة تثبيتها، بالإضافة إلى إمكانية تخصيص عملية التثبيت وبالتالي يمكننا تثبيت بعض المزايا بشكل منفصل.
2- إذا تم عطب جزء من التطبيق أو كله فسيقوم بإصلاح نفسه ذاتياً باستخدام ميزة الإصلاح الذاتي لحزم Windows Installer.
3- إمكانية الرجوع للحالة السابقة (Rollback) إذا أخفقت عملية التثبيت، فإذا أخفقت عملية التثبيت سيعاد كل شيء كما كان في السابق وذلك يتضمن أن تعديل على مفاتيح مسجل النظام أو أيه ملفات.
4- إمكانية إزالة التطبيق بالكامل، ويتضمن ذلك أي معلومات مسجلة في سجل النظام تم وضعها أثناء تثبيت التطبيق أو أيه ملفات للتطبيق.

إنشاء حزمة تثبيت لبرنامج قمنا بإنشائه في Visual Studio.NET

سنقوم الآن بشرح لآلية نشر تطبيق NET. ولكن قبل ذلك علينا أن نستعرض عملية تخطيط الثبيت أولاً:

عملية تخطيط التثبيت:

قبل أن نبدأ ببناء برنامج التثبيت، سنخطط لما سنضعه في هذا البرنامج، لذلك يوجد بعض الأسئلة المطروحة أولاً:

ما هي الملفات التي نحتاجها للتطبيق؟ بالطبع إن الملف التنفيذي وبعض المجمعات هي تلك الملفات التي يجب أن تتواجد ضمن حزمة التثبيت، وإذا تطلب الأمر يمكننا تضمين ملفات مساعدة مثل ملفات Readme.txt أو ملفات الإعدادات أو ملف قاعدة المعطيات مثلاً. وهذا يقودناإلى ضرورة معرفة جميع الملفات المطلوبة.

ما هي المجلدات التي يجب أن نستخدمها؟ يجب أن تثبت ملفات التطبيق ضمن مجلد ما في المجلد Program Files. يجب أن نسمح للمستخدم أن يحدد مكان مجلد التطبيق الذي سوف نقوم بتنصيبه إذا لم يرغب في المسار الإفتراضي الذي نفرضه نحن.

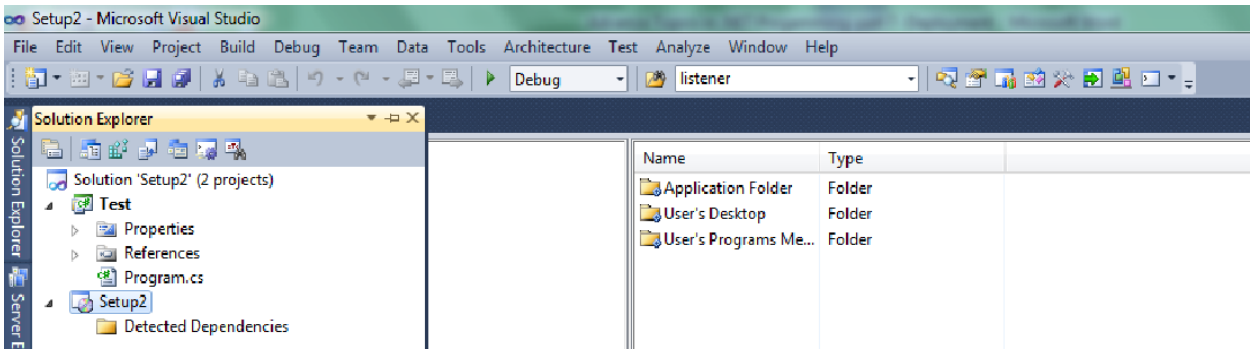
كيف سيصل المستخدم للتطبيق؟ يمكننا وضع إختصار للملف التنفيذي ضمن قائمة"ابدأ" أو وضع أيقونة على سطح المكتب.

ما هو وسط التوزيع؟ هل نود وضع حزمة التثبيت على CD أو على Hard Disk أو على الشبكة.

ماذا سنسأل المستخدم ؟ يجب أن نحدد فيما إذا كان على المستخدم قبول نص ترخيص التطبيق License وعرض ملف القراءة Readme، وسؤاله عن المسار الذي سيتم تثبيت التطبيق ضمنه؟ وبعض الخيارات الأخرى لتثبيت التطبيق.

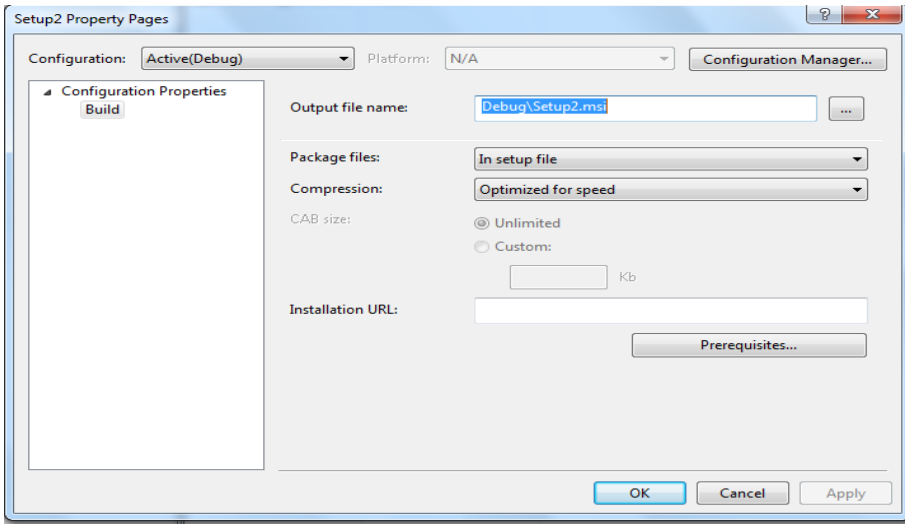
إنشاء مشروع Windows Installer

نقوم بإضافة Setup Project، سثم نقوم بإضافة المشروع الذي نود نشره (الذي نريد أن نقوم بتثبيته) وذلك من خلال الخيار Add Existing Project (يظهر عن طريق النقر بزر الفأرة الأيمن على Solution Explorer).
وبعد ذلك نرى الشكل التالي:



خصائص المشروع :

يجب أن نُعرف في البداية الملفات التي ستثبت، وعلينا ضبط خصائص المشروع لذلك نختار من القائمة Project الخيار Properties فيظهر لنا الشكل التالي:



يمكننا من خلال الخيار Package Files معرفة كيفية توضع الملفات ضمن حزمة التثبيت كما يمكن ضبط بعض الخصائص مثل اسم ومكان ملف الخرج (Setup2.msi، كما نرى في الصورة السابقة)، كما يمكن تحديد نمط تنفيذ المشروع Debug (نمط التنقيح وهو بطيئ) أو Release (نمط الإطلاق وهو سريع).

لمربع التحرير والسرد Package Files ثلاث قيم هي:

1- **الخيار As Losose uncompressed files:** ويعني أن جميع الملفات والبرامج ستحفظ كما هي دون ضعطها.
2- **الخيار In Setup File:** يعني أن جميع الملفات ستدمج وتضغط ضمن ملف MSI.
3- **الخيار Cabinet files:** حيث ستحفظ جميع الملفات بهيئة مضغوطة ضمن ملفات CAB ويمكننا ضبط أحجام هذه الملفات كما نريد.

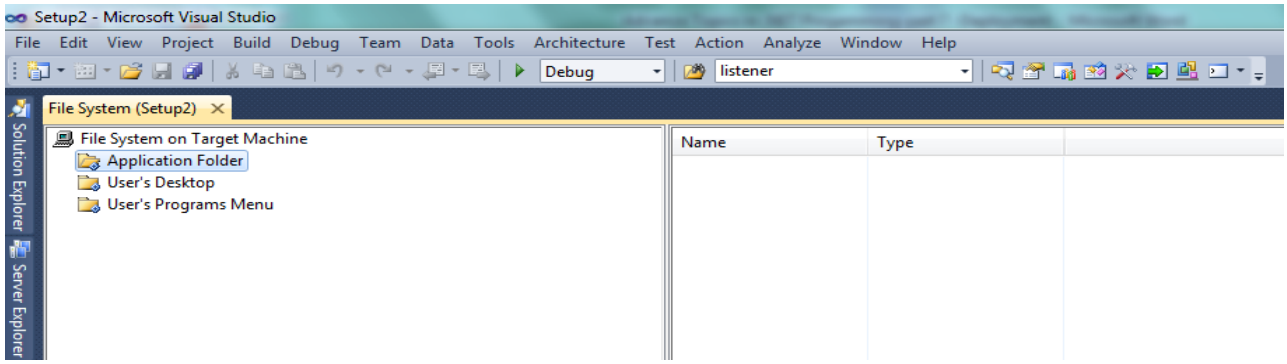
ثم نقوم بضبط خصائص المشروع وذلك من خلال نافذة خصائص المشروع Properties Windows، يمكن أن نضع اسم الكاتب Author، مثلاً: محمد نزار غنوم شويش ويمكن أن نضع توصيف للمشروع من خلال الخاصية Description بالإضافة إلى بعض الخصائص الهامة والتي يمكن الإطلاع عليها.

محررات الإعدادات Setup Editors

يتضمن مشروع الإعداد Setup Project ستة محررات، يمكننا تحديد المحرر من خلال القائمة View ثم القائمة الفرعية Editors ثم نختار أحد المحررات وهي كما يلي:

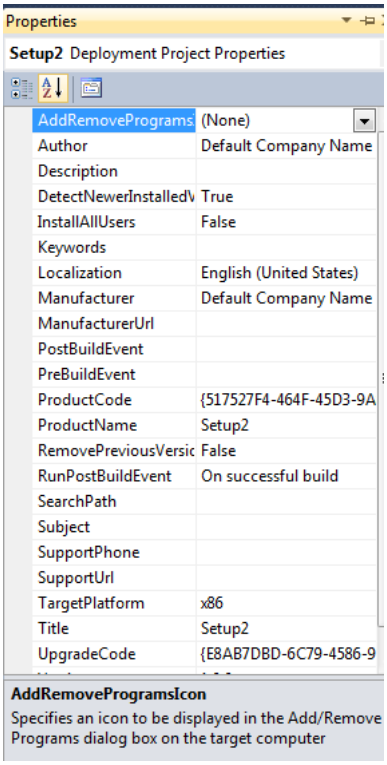
1- محرر نظام الملفات File system

يمكننا بواسطته إضافة الملفات إلى حزمة الثبيت وضبط مواقعها على جهاز الزبون، سنرى الشكل التالي عند إختيار محرر نظام الملفات:



1- يمثل المجلد Application Folder مجلد التطبيق وهو الذي سيحوي على الملفات التنفيذية للتطبيق ومكتبات الشيفرة وقواعد البيانات وحتى الشيفرة المصدرية (إذا أحببنا وضعها)، بالإضافة إلى ملفات المصادر مثل الصور والملفات النصية. ويُعرّف موقع هذا المجلد وفق الصيغة التالية:

```
[Program Files Folder] [Manufacturer]
[ProductName]
```



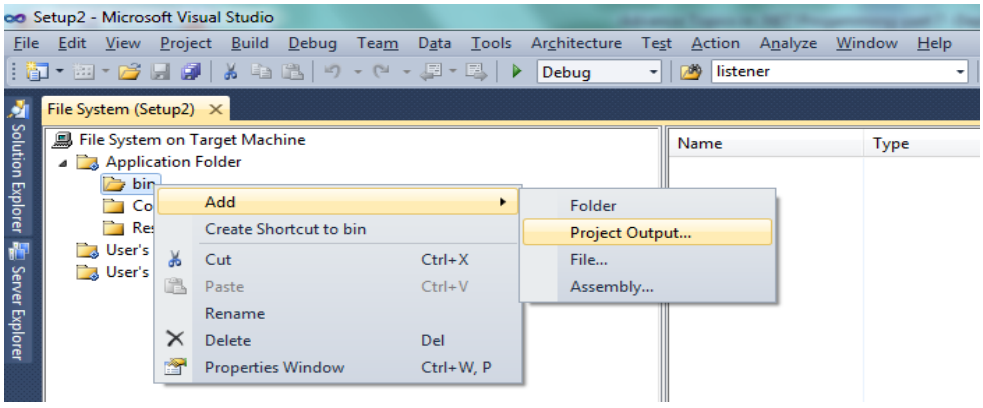
الخصائص ProductCode و ProductName يمكن ضبطها من خصائص المشروع كما كما في الشكل التالي:

2- بينما يمثل المجلد Global Assembly Cache Folder وهو يحوي على المجمعات المشتركة والتي بين عدة تطبيقات والموقعة بأسم قوي.
3- إذا أردت أن تضع أيقونة على سطح المكتب فأنك ستستخدم المجلد user Desktop.
4- كما يمكن وضع إختصار لتطبيق ضمن قائمة ابدأ وذلك من خلال المجلد user Program menu.

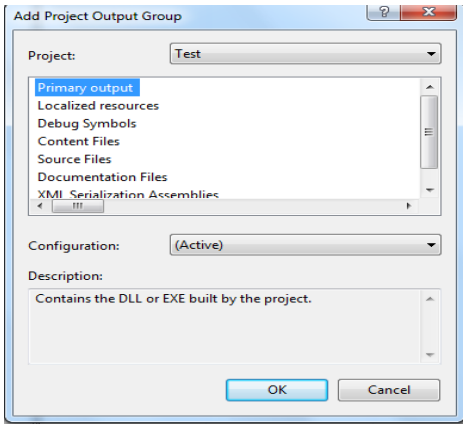
في البداية سنقوم بإضافة الملفات التنفيذية للمشروع وذلك في المجلد Application Folder ولكن قبل ذلك يمكننا أن ننشئ مجلدات فرعية ضمنه على سبيل المثال:

1- **المجلد bin** والذي يحوي على الملف التنفيذي بالإضافة إلى المجمعات الخاصة به كما يمكن وضع قاعدة معطيات ضمنه أيضاً.
2- المجلد Code والذي يحوي على الشيفرة المصدرية للمشروع.
3- المجلد Resources والذي يحوي مصادر المشروع مثل صورة أيقونه الملف التنفيذي وملفات التراخيص وملفات الReadme.
4- في البداية نقوم بإنشاء المجلدات الثلاثة السابقة ضمن المجلد Application Folder من خلال النقر بالزر الأيمن على المجلد Application Folder و من ثم نختار ADD → Folder

ثم نقوم بإضافة خرج المشروع test إلى التطبيق كما يلي:



ثم نحدد ما الذي نريد إضافته من خلال النافذة التالية:



1- **Primary output:** يحوي ملفات dll, exe للمشروع الحالي test، وبذلك نكون قد أضفنا خرج المشرع إلى المجلد bin.
2- يمكن إضافة ال **source Code** من خلال اختيار **source files** أو يمكن إضافة أمور أخرى كما هو معروض مثل المحتوى وملفات التوثيق.
3- **هذه الملفات** التي قمنا بإضافتها مثل خرج المشروع test يمكن ضبط خصائصها أيضاً، حيث يوجد العديد من الخصائص الهامة وهي كمايلي:

الخاصية	الوصف
Condition	تستخدم هذه الخاصية لتحديد فيما إذا كنا نود تثبيت هذا الملف أم لا
Exclude	عندما تأخذ هذه الخاصية القيمة true هذا يعني أن الملف لن يتم وضعه في حزمة التثبيت .
Permanent	عندما تأخذ هذه الخاصية القيمة true هذا يعني أن الملف سيبقى على جهاز الزبون حتى بعد إزالة التطبيق
Readonly	عندما تأخذ هذه الخاصية القيمة true هذا يعني أن الملف قابل للقراءة على جهاز الزبون
Vital	تشير هذه الخاصية إلى أن هذا الملف أساسي لتثبيت التطبيق وفي حال الإخفاق في تثبيت هذا الملف ستخفق عملية التثبيت كلياً.

- الآن سنقوم بإضافة الشيفرة المصدرية إلى المجلد Code كما تكلمنا سابقاً.
- ثم نقوم بإضافة بعض الملفات النصية مثل ملفات التراخيص وملفات المساعدة و صروة أيقونة التطبيق إلى المجلد Resources، وذلك من خلال النقر بالزر الأيمن على هذا المجلد ومن ثم نختار Add→file.
- ثم نقوم بإنشاء إختصار لخرج التطبيق وذلك من خلال تحديد المجلد bin ثم ننقر بالزر الأيمن على خرج التطبيق Primary output ونختار Create Shortcut to Primary output test (Active)، ثم نقوم بنسخ هذا الملف copy ثم نقوم بلصقه paste في مجلد user Disktop وثم نقوم بلصقة مرة أخرى في المجلد user Program menu.

2- محرر أنواع الملفات File Types

هناك بعض التطبيقات المشهورة مثل word والتي تقم بربط أنواع محدد من الملفات مع تطبيقها، فالملفات ذات اللاحقة doc. يتم فتحها تلقائياً عند النقر عليها مرتين متتاليتين ضمن برنامج word، مثلاً: في حال قمنا ببناء محرر نصوص يجب أن يقوم بفتح الملفات التي أنشأناها في هذا التطبيق لذلك يجب أن نعطي هذه الملفات اسم ولاحقة حتى يتم فتحها بواسطة برنامجنا عند النقر عليها لمرتين متتاليتين.

في البداية نقوم بإظهار محرر أنواع الملفات من القائمة view →Editors→file Types ثم نقوم بإضافة file type، ويكون له مجموعة من الخصائص هي كما يلي:

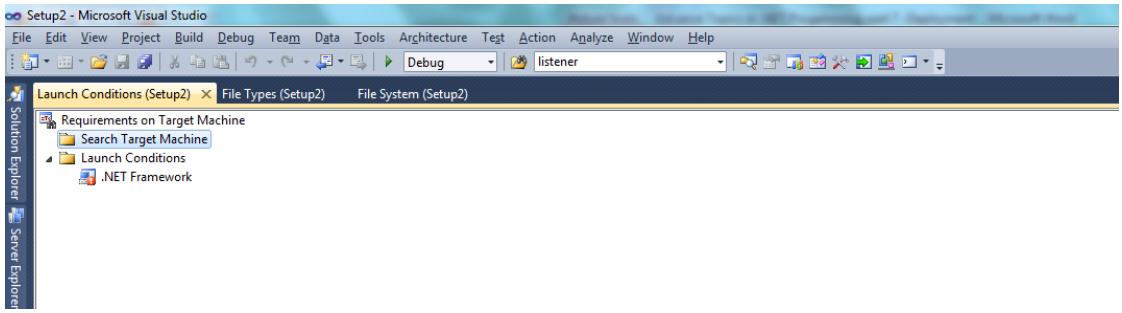
الخاصية	الوصف
Name	يجب أن نضع اسم مناسب يدل على نوع الملف .
Command	تُمثل هذه الخاصية الملف التنفيذي الذي سينفذ عندما يقوم مستخدم بفتح ملف من هذا النوع
Description	توصيف هذا النوع من الملفات
Extensions	سنحدد هنا إمتداد الملف الذي سيتم تسجيله مع تطبيقنا .حيث سيتم تسجيل ذلك في مكان ما من مسجل النظام .
Icon	هنا سيتم تحديد الأيقونة التي ستأخذها الملفات التي من هذا النوع –والتي من المفترض أن تكون موجودة في أحد المجلدات التي تكلمنا عنها سابقاً- عادةً يجب أن تكون في مجلد Resources

إنشاءالأحداث

بعد إنشاء أنواع الملفات في محرر أنواع الملفات يمكننا إضافة أحداث Actions، إن الحدث الذي سيضاف تلقائياً هو Open ويمثل حدث فتح التطبيق وفتح الملف ضمنه، ولكن يمكننا إضافة أحداث أخرى مثل Printأو أي حدث كان يمكن لبرنامجنا أن يقوم به مع الملفات من هذه الأنواع وعند تعريف كل حدث علينا تحديد قيم الخاصتين Arguments ="%1" والتي تعني أن اسم الملف سيتم تمريره كبارمتر للتطبيق والخاصية verb=open مثلاً أو يمكن أن تكون لها قيمة print/ إذا كان تطبيقنا يدعم هذه الكلمة.

3-محرر منفذ الشروط Launch Condition

يمكننا من خلال هذا المحرر تخصيص بعض المتطلبات التي يجب توفرها في جهاز الزبون قبل البدء بعملية التثبيت، نقوم بعرضه فيظهر لنا التالي:



نلاحظ وجود قسمين رئيسين يتم من خلالهما تخصيص المتطلبات:

1- **القسم Search Machine:** يتم من خلاله تخصيص عمليات بحث في ملفات محددة أو مفاتيح محددة في سجل النظام.
2- **القسم Launch Conditions:** يعرف رسائل الخطأ التي ستظهر عند إخفاق عمليات البحث المعرّفة في القسم الأول.

ملاحظة: سابقاً كان يوجد خيار للبحث عن مكتبات NET Framework. في جهاز الزبون، وذلك قبل أن تصبح NET Framework. جزءاً من النظام Windows، وفي حال عدم توافره فأنه سيتم تحميله من الأنترنت. يمكننا رؤية خصائص هذا الشرط الخاص بتوافر NET Framework. على حاسب الزبون فنجد مجموعة من الخصائص مثل: الرسالة التي سيصدرها في حال عدم وجود NET Framework. على حاسب الزبون.

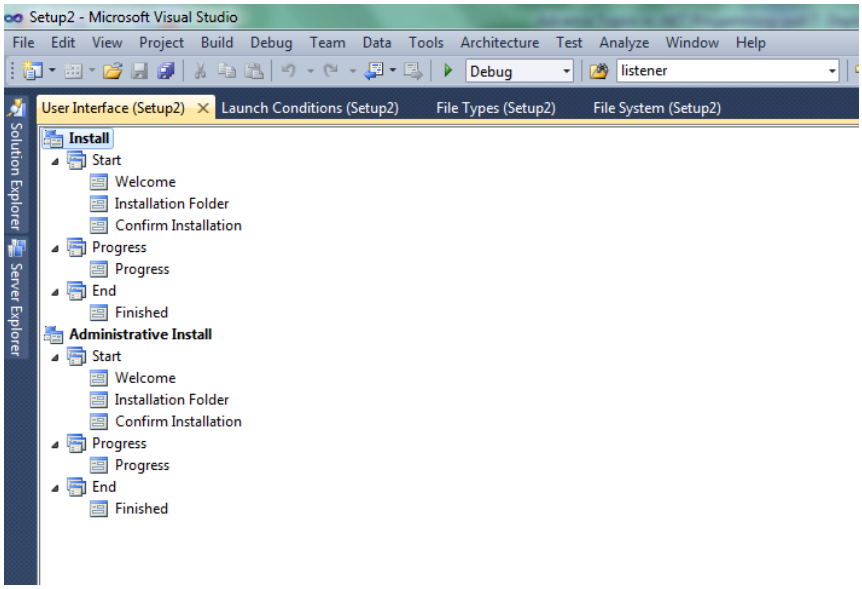
4- محرر واجهة المستخدم user interface

يمكننا من خلال هذا المحرر تعريف صناديق الحوار التي ستعرض للمستخدم لضبط عملية التنصيب، مثل: نص إتفاقية الترخيص ومكان التنصيب بالإضافة إلى معلومات أخرى.

تعرض هذه الواجهة للمستخدم صناديق الحوار المولدة تلقائياً وفقاً لتسلسل عرضها أثناء عملية التثبيت وفقاً لنمطين :

1- **نمط التثبيت :** عملية التثبيت النموذجية للتطبيق على جهاز الزبون.
2- **ونمط التثبيت الإداري:** تثبيت صورة للتطبيق Image على عقدة شبكية مشتركة Network Share، يمكن عندئذٍ تثبيت التطبيق من الشبكة.

لنرى الآن صناديق الحوار الإفتراضية.



صنايق الحوار الإفتراضية

نلاحظ من خلال صناديق الحوار أن أن هناك ثلاث تسلسلات تظهر وفقها صناديق الحوار، لنلق نظرة على صناديق الحوار الإفتراضية:

1- يستخدم صندوق الحوار Welcome لعرض رسالة ترحيبية، يمكن تغيير خصائص هذا الصندوق مثل تغيير رسالة الترحيب أو حتى الصورة الموجودة ضمنه.
2- الصندوق الثاني هو installation Folder: حيث يتم تحديد مكان تثبيت التطبيق، إذا كنت تود إضافة صناديق حوار أخرى لبرنامج التثبيت فعليك أن تضيفها قبل صندوق الحوار هذا.
3- صندوق حوار تأكيد عملية التنصيب Confirm Installation.
4- يستخدم صندوق الحوار Progress لعرض المرحلة التي وصلت إليها عملية التثبيت حالياً.
5- صندوق الحوار Finish والذي يدل على إنتهاء تثبيت التطبيق.

ملاحظة: لكل صندوق حوار مجموعة من الخصائص، مثلاً : صندوق الحوار Welcome له خصائص مثل الرسالة الترحيبية و الصورة التي ستظر في الشاشة الترحيبية بالإضافة إلى نص حقوق الملكية.

صناديق حوار إضافية

يوفر لنا Visual studio .NET مجموعة من صناديق الحوار الجاهزة لكي نضيفها تسلسل ظهور صناديق الحوار، يمكننا إضافة صناديق حوار وذلك بالنقر بالزر الأيمن للفأرة على التسلسل Start ثم نختار Add Dialog فيظهر لنا مجموعة من صناديق الحوار نذكر منها:

1- صندوق الحوار Register user: يستخدم لعرض رسالة تفيد بأنه إذا نقر على زر Register Now سيتم تسجيله كمستخدم للتطبيق.
2- صندوق الحوار License Agreement: يعرض رسالة ترخيص للمستخدم يمكن تحديد الملف الذي يحوي نص الترخيص من خلال الخاصية License File.

بناءالمشروع

الآن سنقوم ببناء المشروع وذلك من خلال القائمة Build نختار Build Solution فينتج لدينا ملف التنصيب، وهو موجود في مجلد Debug الخاص بالمشروع (في حال كنا نعمل في نمط التنقيح).

ومن ثم يمكننا تنصيب البرنامج، كما يمكننا ثتبيت أو إلغاء تثبيت التطبيق عن طريق النقر بالزر الأيمن على اسم المشروع Setup Project في ال Solution Explorer ومن ثم نختار Install أو uninstall.

الفصل الثامن

تعدد النياسب

Multithreading

مقدمة

يعتبر موضوع تعددية النياسب (MutltiThreading) من أهم المواضيع المُستخدمة في التطبيقات الشبكية وفي النظم المؤسساتية (Enterprise Applciations) و حتى في التطبيقات العادية، وسنجد أنه لا يوجد تطبيقات تعمل من دون أن تستثمر إمكانية تعدد النياسب التي يتيحها لنا نظام التشغيل.

سنتكلم في البداية عن مفاهيم أساسية في نظم التشغيل ثم سننتقل إلى كتابة أمثلة عملية وحقيقية، ستكون هذه الأمثلة بمثابة صقل لكافة الأفكار والمفاهيم التي سنتكلم عنها.

مفاهيم أساسية في نظم التشغيل

عندما نقوم بتشغيل أي برنامج يحوي شيفرة تنفيذية أو عند ترجمة (Compile) ملف يحوي شيفرة مصدرية ثم تشغيل البرنامج الناتج والذي يحوي شيفرة تنفيذية فإن نظام التشغيل يقوم بتحميل هذه البرنامج إلى الذاكرة ويضعه قيد التفيذ (يصبح اسم البرنامج هنا مهمة process)، يقوم نظام التشغيل بحجز 4 مناطق للمهمة في الذاكرة هي كما يلي:

1- **منطقة للكود Code Segment :** تحوي الشيفرة التنفيذية للبرنامج الذي نريد تنفيذه.
2- **منطقة معطيات Data Segment:** تحوي معطيات البرنامج الذي نريد تنفيذه، مثل المتحولات العامة والمتحولات الستاتيكية.
3- **منطقة المكدس Stack segment:** تحوي بارمترات الاستدعاءات للتوابع المختلفة، بالإضافة إلى المتحولات المحلية (local variables) والقيم المُعادة من إستدعاءات التوابع.
4- **منطقة الكومة Heap Segment:** تحوي معاملات الحجز الديناميكي (الأغراض الناتجة عن العملية New)، بالإضافة إلى كتلة تحكم بالمهمة (Process Control Bolck) إختصاراً PCB، تحوي هذه الكتلة معلومات شاملة عن عن المهمة مثل حالة المهمة ورقم المهمة وغيرها من المعلومات الهامة جداً، بالإضافة إلى عناوين المناطق الذاكرية الثلاثة الأولى التي تكلمنا عنها (منطقة الكود والمكدس والمعطيات) وبالإضافة إلى معلومات عن الدخل/خرج (I/O).

يمكننا يتعريف المهمة (Porcess) بأنها برنامج قيد التنفيذ، حيث يقوم نظام التشغيل بحجز المناطق الذاكرية الأربعة التي ذكرناها سابقاً للأسباب التالية:

1- عادةً يتم تحميل أكثر من برنامج إلى الذاكرة ويتم تنفيذها واحداً تلوا الأخر، لأن التعامل مع الذاكرة الرئيسية RAM أسرع من التعامل مع القرص الصلب (Hard Disk) بمئة ألف مرة تقريباً.

2- البرنامج التفيذي موجود على القرص الصلب بشكل ساكن وعند تنفيذه يصبح مهمة (Porcess)، و يحجز له نظام التشغيل المناطق الأربعة التي تكلمنا عنها، وبالتالي نريد أن نقلل من عدد مرات النفاذ للقرص الصلب.

ويقوم نظام التشغيل بإعطاء شريحة زمنية (Time slice) لكل مهمة (Process)، وعند إنتهاء هذه الشريحة يقوم المعالج بتنفيذ مهمة أخرى من رتل الأنتظار (يتم حفظ سياق المهمة الحالية في الذاكرة ثم يتم تحميل سياق المهمة

الجديدة إلى المعالج)، يقوم المعالج CPU بإختيار أحدى المهام المنتظرة حتى يتم تنفيذها من رتل الإنتظار وفق خوارزمية جدولة معينة[1].

عملية تحميل عدة برامج برامج إلى الذاكرة وتنفيذ واحد منها في لحظة زمينة معينة تُسمى MultiProgramming (كل ما سبق هو بإفتراض أنه لدينا معالج واحد فقط).

تعرفنا على تفاصيل المهمة (Process)، فما هو النيسب (Thread)؟

في الحقيقة قبل تعريف النيسب (Thread) يجب أن نعلم أن كل مهمة (Process) تحوي Thread واحد على الأقل وتسمى عندها single-Threaded ، يمكننا إعتبار المهمة (Process) بمثابة حاوية (Container) لمجموعة من النياسب (Threads)، بشكل تفصيلي،عندما نقوم بتشغيل برنامج بلغة C# أو C++ فأن البرنامج يحوي لتابع ()main وهذا التابع هو نيسب (Thread) منفصل.

تعريف النيسب (Thread): هو عبارة عن مسلك برمجي مستقل في المكدس (Stack) ويشترك مع غيره من النياسب في الكومة (Heap)[2]، بعبارة أخرى: هو كتلة من التعليمات التي نريد تنفيذها على التوازي مع غيرها من التعليمات الأخرى.

بشكل فعلي كل نيسب هو عبارة عن Stack بالإضافة إلى مجموعة من السجلات الخاصة بما فيها عداد البرنامج (PC).

ذكرنا سابقاً أن نظام التشغيل يُخصص لكل مهمة (Process) شريحة زمنية لكي يتم تنفيذها وفقاً لخوارزمية جدولة معينة، هذه الشريحة الزمنية المخصصة لهذه المهمة يتم توزيعها على عدد النياسب الموجودة في هذه المهمة وفقاً لخوازمية جدولة معينة أيضاً. الجدير بالذكر أن التبديل بين النياسب ضمن المهمة يتم بسرعة كبيرة جداً وهذا يعطي إنطباع بأن التنفيذ يتم على التوازي.

الجدير بالذكر أيضاً أن إدارة المهام (Processes Management) هي من مهمة نظام التشغيل بالإضافة إلى قضايا التزامن وإدارة التضاربات والتشارك على الموارد.أما إدارة النياسب (Threads Management) فهي من مهمة المبرمج لذلك سنتكلم إداة النياسب في هذا الفصل.

جميع الأفكار المتناولة في هذا الفصل مستوحاة من لغة البرمجة C، الرابط التالي يحوي شرح تفصيلي للنياسب في لغة البرمجة C ضمن نظام التشغيل unix:

https://computing.llnl.gov/tutorials/pthreads/#PthreadsAPI

[1] للمزيد يمكن الإطلاع على الرابط التالي : (http://en.wikipedia.org/wiki/Scheduling_(computing

[2] للمزيد يمكن الإطلاع على الرابط التالي : https://www.cs.rutgers.edu/~pxk/416/notes/05-threads.html

لماذا نحتاج إلى تعدد النياسب Multithreading

في الحقيقة معظم التطبيقات التي نستعملها نستخدم أكثر من نيسب وذلك للأسباب التالية:

1- **التنفيذ في الخلفية (Executing in Background):** لنفرض أننا نريد تنفيذ أمر طباعة مثلاً عندها لكي لا ينتظر المستخدم أمر الطباعة حتى يتم، نجعل أمر الطباعة في نيسب منفصل ونشغله ونكمل عملنا بشكل طبيعي.

2- **إنجاز مهمة اكثر من مرة لتسريع العمل:** مثلاً : تنفيذ برنامج لمعالجة صورة معينة يمكن تقسيم العمل على أربعة نياسب وبالتالي تسريع العمل مع إستغلال أمثل للموارد لكن لن نحصل على ما نريده بشكل أمثلي من خلال معالج واحد وإنما نحتاج إلى أكثر من معالج (الاستغلال الأمثلة للموارد المتاحة).

3- **تعدد النياسب هو الطريقة الطبيعية لبناء نوع محدد من البرامج:** مثلاً: Web Server يقوم بالتنصت"listening" على منفذ "Port" محدد منتظراً أن يتصل به الزبائن (Clients). إذاً كيف سيستطيع معالجة طلب الزبون الأول بينما يجب عليه أن يبقى منتظراً لطلب زبون جديد قادم ؟ يكمن الحل في إنشاء نيسب لمعالجة طلب كل زبون متصل لذلك سيكون تخديم ستة عملاء (زبائن) لن يكون مختلفاً عن معالجة زبون واحد لأننا سننشئ ستة نياسب متطابقة.

كيف نقوم بإنشاء تطبيقات متعددة النياسب Multithreaded

علمنا إلى الآن أن النيسب (Thread) هو مجموعة من التعليمات التي يتم تنفيذها على التوازي مع بقية البرنامج وهذه التعليمات تُمثل Method يتم تمرير اسم هذا ال Method إلى مفوض (أو مندوب يُسمى Delegate) وهذا المندوب يتم تسليمه إلى غرض من الصف Thread .

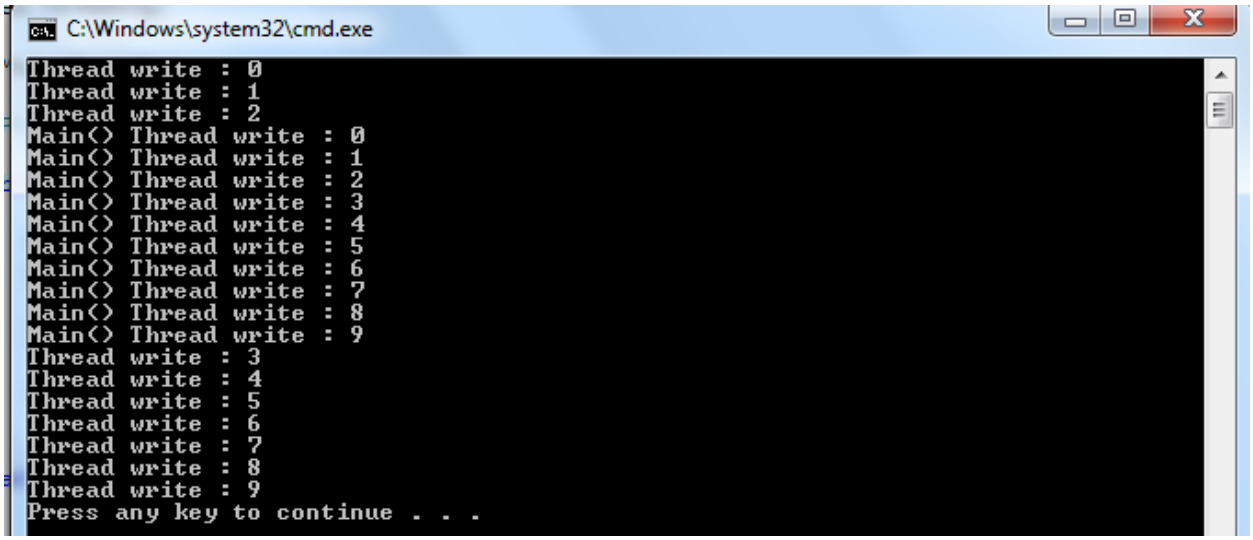
لنرى البرنامج التالي:

```
using System;
using System.Threading;

namespace Multithreading
{
    class Program
    {
        public static void process()
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("Thread write : {0}", i.ToString());
            }

            //Console.WriteLine();
        }
```

```
        static void Main(string[] args)
        {

            Thread t1 = new Thread(new ThreadStart(process));

            t1.Start();


            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("Main() Thread write : {0}", i.ToString());
            }
            //Console.WriteLine();
        }
    }
}
```

شرح الكود:

في البداية قمنا باستخدام فضاء الأسماء System.Threading، كل ما يلزمنا لتشغيل منهج كائن (Method) على أنه نيسب (Thread) هو أن نقوم بإنشاء غرض من الصف Thread وهذا الغرض يأخذ كبارمتر كائن مفوض (Delegate اسمها ThreadStart) وهذا المفوض هو عبارة عن مؤشر إلى المنهج (أو التابع) الذي نريد تنفيذه ك نيسب وبعد إنشاء النيسب نقوم باستدعاء المنهج Start() للكائن t1 وذلك يؤدي إلى تغيير حالة الغرض t1 إلى نيسب قيد التشغيل ضمن المهمة الحالية (Process). وبالتالي لدينا الآن نيسبان (Two Threads) الأول هو المنهج Main() والثاني هو t1 حيث يتم تنفيذ كل منهما على التوازي ولنرى الخرج الآن.



في الحقيقة النتيجة تختلف عند التنفيذ لمرات مختلفة وحتى أن الخرج يتخلف من حاسب يحوي معالج وحيد إلى حاسب يحوي معالجين أو أكثر.

ملاحظة:

Delegate: هي مفوض عن مجموعة من التوابع التي تحمل نفس التوقيع، حيث يمكننا تعريف توقيع معين يخضع له مجموعة من ال Methods، وعن طريق كائن من هذا النمط (Delegate) يمكننا تنفيذ Method واحد أو أكثر عن طريق وضعها ضمن رتل (Queue) ثم تنفيذها واحداً تلوا الأخر (تشبه هذه الفكرة فكرة المؤشر إلى تابع في لغة C/C++).

في الحقيقة هذه التقنية مستخدمة في الأحداث وفي النياسب.

لنرى المثال التالي:

```
namespace MultiThreading
{
    public delegate void mydelegate(int a);

    class Program
    {

        public static void delegate_fun1(int a)
        {

            Console.WriteLine(" fun1 " + a.ToString());
        }

        public static void delegate_fun2(int a)
        {

            Console.WriteLine(" fun2 " + a.ToString());

        }

        static void Main(string[] args)
        {

            // delegate  ; Pointer to function
            // delegate  declaration in namespace
            mydelegate d1 = new mydelegate(delegate_fun1);

            d1(10); // calling delegate_fun1(10);

            d1(20);//  Calling delegate_fun1(20);

            d1 += delegate_fun2;//Add method to method Queue ; that means add delegatefun2

            d1(30);//Calling delegate_fun1(30); then Calling delegate_fun2(30); because it is
queue
            d1 = delegate_fun2; // clear Queue and Add delegate_fun2 to Queue

            d1(50);


            //anonymous method
```

```
            d1 += delegate(int n)
            {
                Console.WriteLine("anonymous method {0}", n);
            };

            d1(70);

        }
    }
}
```

شرح الكود:

قمنا في البداية بتعريف مفوض (Delegate) عن مجموعة من التوابع (أو المناهج) التي تحمل توقيع معين (يضعه المبرمج كما يريد) باسم mydelegate.

ثم قمنا بإنشاء كائن من نمط mydelegate وقمنا بإعطائه اسم Method (delegate_fun1) والتي لها نفس التوقيع التي تفرضه ال mydelegate التي قمنا بتعريفها، ثم قمنا باستدعاء التابع عن طريق كائن المفوضd1 (والذي يمثل مؤشر إلى رتل من التوبع التي قمنا بإضافتها والتي تحمل نفس التوقيع المطابق لتعريف المفوض mydelegate).

وبالتالي التعليمة التالية: `d1(10);` ستقوم باستدعاء المنهج delegate_fun1 من أجل القيمة 10.

و من ثم نقوم باستدعاء نفس المنهج السابق(delegate_fun1) من أجل القيمة 20.

بعد ذلك نقوم بإضافة المنهج الثاني (delegate_fun2) إلى رتل التوابع التي يؤشر عليهاالمفوض d1(كل من المنهجين المُضافين يحملان نفس التوقيع)، وذلك عن طريق التعليمة التالية : `d1 += delegate_fun2;`

وبالتالي الآن عند إستدعاء المناهج الموجودة في الرتل عند القيمة 30 بالشكل: `d1(30);` سيؤدي ذلك لإستدعاء المنهجين delegate_fun1 و delegate_fun2 من أجل القيمة 30.

بعد ذلك قمنا باسناد(=d1) قيمة جديدة للمفوض d1 (أو للمؤشر إلى رتل التوابع)، وهذه القيمة هي قيمة المنهج delegate_fun2 وبالتالي عند استدعاء قيمة المفوض d1(50) سيؤدي إلى استدعاء المنهج delegate_fun2 عند القيمة 50.

الجدير بالذكر أن المفوض يمكن استخدامه مع المناهج التي ليس لها اسم (anonymous method) والقسم الأخير من الكود يوضح هذه الفكرة، حيث أننا قمنا بإضافة منهج من دون اسم، ولكن له نفس التوقيع الذي يفرضهmydelegate.

لنرى الخرج الآن:

إيقاف نيسبThread Terminatation

تعرفنا إلى الآن كيف ننشئ نسيب ونشغله عن طريق المنهج ()Start، ولكن كيف يتم إيقاف النيسب ومتى يتنهي عمله؟

في الحقيقة ينتهي عمل النيسب بشكل طبيعي عند انتهاء كود المنهج (Method) الذي قمنا بربطه مع غرض من الصف Thread لكي يتم تنفيذه ك نيسب.

أما كيفية إنهائه بشكل قسري فإن ذلك يتم بواسطة المنهج (Method) ()Abort للصف Thread والذي يقوم برمي إستثناء من النمط ThreadAbortException، وبعد ذلك تبدأ المهمة (Process) المسؤولة عن هذا النيسب بإنهاء النيسب الذي قام برمي الإستثناء السابق.

ملاحظات:

- من غير المحبذ أستخدام المنهج Abort لأنه يقوم بإيقاف عمل النيسب فوراً أثناء عملة وهذا قد يسبب بعض المشاكل الخطيرة مثل إيقاف نيسب يقوم بعملية كتابة على ملف!، أو يقوم بتحديث معطيات ضمن ضمن قاعدة معطيات فأن ذلك يؤدي إلى تخريب البيانات!.
- يُنصح عادة باستخدام متحول منطقي (Boolean) يتم تفحصه داخل النيسب للتأكد من إنتهاءعمله وذلك لكي نعطي للنيسب فرصة لكي يقوم بعمليات التنظيف (Release) قبل الإنتهاء، مثل إغلاق الملفات المفتوحة بالإضافة إلى إغلاق الإتصال مع قاعدة المعطيات.

النياسب التي تعمل في الخلفية Background Threads

- في الحقيقة إن نظام التشغيل يستخدم مجموعة من المهام (Processes) والتي تعمل في الخلفية (Background) بأولوية أقل من باقي المهام، وتسمى في نظام ويندوز بأسم (Windows Services) "خدمات ويندوز"، أما في نظام Linux/Unix فتُسمى daemons. هذه المهام (Processes) لا تظهر للمستخدم ويمكن أن تعمل بشكل تلقائي مثل المهمة explorer.exe المسؤولة عن إظهار النوافذ في نظام ويندوز، أو بشكل يدوي مثل OracleServer والتي هي مسؤلة عن جعل الحاسب يعمل كمخدم قواعد معطيات.

- الفكرة ذاتها يمكن تطبيقها على النياسب حيث يمكننا أن نجعل نيسب معين يعمل في الخلفية (Background) ، وهذه النياسب مناسبة جداً من أجل إنجاز المهام التي لا يتوجب حدوثها فوراً كما أن المستخدم لن يحتاج إلى إنتظارها، لذلك يتم تشغيل هذه النياسب التي تعمل في الخلفية بأفضلية أقل من النياسب العادية والتي تسمى بالنياسب الأمامية.

- كما يمكن أن نجعل نيسب ما يعمل في الخلفية (Background) عن طريق الخاصية IsBackground والتي يتم ضبطها بالقيمة true.

- عادةً المهام (Processes) أو النياسب (Threads) التي تعمل في الخلفية (Processes) لا تستطيع التخاطب مع عناصر GUI (الواجهات).
- تنتهي النياسب الخلفية (Background) عند إنتهاء أخر نيسب أمامي (forground).

Thread Class

يمثل الصنف System.Threading.Thread نيسب في مستوى نظام التشغيل.

أهم الخصائص الموجود في الصنف Thread.

الخاصية	الوصف
CurrentThread	تعيد مرجع (Reference) إلى النيسب المنفذ حالياً .
IsAlive	تعيد true إذا كان النيسب قد بدأ ولم ينتهي بعد.
ISBackground	تعيد true إذا كان النيسب يعمل في الخلفية.
Name	تعيد أو تحدد (set or get) اسم النيسب.
Priority	تعيد أو تحدد (set or get) أفضلية النيسب.
ThreadState	تعيد أو تحدد (set or get) حالة النيسب.

أهم المناهج الموجود في الصنف Thread.

الطريقة (method)	الوصف
Start	البدأ بتنفيذ بتنفيذ النيسب.
Abort	تنهي تنفيذ النيسب بشكل فوري.
Interrupt	تقاطع النيسب الذي يكون في حالة waitSleepJoin
Join	إنتظار نيسب معين (النيسب المُستدعي لهذا المنهج) حتى ينتهي تنفيذ هذا النيسب.
Suspend	إرجاء عمل النيسب لفترة غير محدودة (pause-إيقاف مؤقت-)
Resume	تبطل عمل التابع السابق (Suspend)
Sleep	تنوم النيسب لفترة محددة.

حالات النيسب Thread States

لكل نيسب حالة أو عدة حالات وهي موجودة ضمن التعداد (Enum) المُسمى ThreadState، يمكن للنيسب أن يكون له حالة واحدة أو أكثر مثل الحالة Aborted. الجدول التالي يعرض قيم هذا التعداد.

القيمة	الوصف
Running	النيسب يعمل الآن
StopRequested	النيسب قد استقبل اشارة إيقاف(Abort)
SuspendRequested	النيسب قد استقبل اشارة إيقاف مؤقت(pause)
Background	النيسب يعمل في الخلفية
Unstarted	النيسب تم إنشاؤه ولكن لم يتم تنفيذ المنهج ()Start
Stopped	النيسب تم إيقافه بشكل نهائي(النيسب مات)
WaitSleepJoin	النيسب متوقف ضن استدعاء wait أو Sleep أو Join.
Suspended	النيسب متوقف بشكل مؤقت
AbortRequested	يتنقل النيسب إلى هذه الحالة عندما نكون قد استدعينا المنهج Abort للنيسب. ولكن هذا النيسب لم يستقبل بعد إشارة الإيقاف والتي يتم التعبير عنها بالإستثناء (ThreadAbortException) والذي ينهي عمل هذا النيسب.
Aborted	نفس الحالة السابقة(AbortRequested) و النيسب قد توقف(مات) ولكن حاته لم تتغير إلى الحالة Stopped.

Thread Priority

- لكل نيسب أفضلية (أولوية) بالنسبة لغيره من النياسب التي تشترك معه في نفس المهمة (Process).

- يتم إنشاء جميع النياسب بأفضلية Normal.

- يجب الحذر عند تحديد الأولويات لأن هذه الأولويات يستعملها نظام التشغيل من أجل تحديد من هو النيسب الذي سيتم تنفيذه الآن، لذلك لا تتوقع دائماً الحصول على النتائج المطلوبة عند تحديد أولويات النياسب عن طريق التعداد (enum) المسمى ThreadPriority والتي يعرضها الجدول التالي.

القيمة	الوصف
Highest	النيسب له الأفضلية العليا
AboveNormal	النيسب له أفضلية أعلى من النيسب العادي
Normal	النيسب له أفضلية عادية
BelowNormal	النيسب له أفضلية أدنى من النيسب العادي
Lowest	النيسب له الأفضلية الدنيا

التنافس على الموارد Concurrency

على الرغم من أن التطبيقات متعددة النياسب (Multithreaded Application) ذات فوائد عظيمة جداً إذا تسمح بالقيام بأكثر من عمل بالوقت نفسه، إلا أن إدارة النياسب ليس أمراً سهلاً (بل هو معقد قليلاً) بالإضافة إلى وجود خطورة في بعض المواقف، مثلاً: إذا حاول نيسبان العمل على غرض مشترك في الذاكرة فإن النتائج ستكون خاطئة أو عندما يقوم نيسبان بالكتابة إلى ملف مشترك في نفس الوقت فأن خطأ ما سوف يحدث. لذلك تستعمل NET. الآليات (أو الطرق) التي يستخدمها نظام التشغيل للمزامنة بين المهام (Processes)، لكي يستخدمها المبرمج من أجل المزامنة بين النياسب (Threads).

ملاحظة هامة: إدارة المهام (Processes Management) من تزامن وحل مشاكل التنافس على الموارد هي مسؤولية نظام التشغيل، أما إدارة النياسب (Threads Management) فهي من مهمة المبرمج، لهذه الأسباب سنرى مفهومين هما:

1- **المقاطع الحرجة (Critical Sections):** وهي مقطع من الكود مشترك بين نيسبين (Two Threads) أو أكثر، وعند دخول نيسبين إلى مقطع الحرج فذلك يؤدي إلى خطأ في النتائج أو خطأ في التنفيذ، لذلك نلجأ إلى استخدام القفل Lock.
2- **القفل(Lock):** وهي طريقة لتحقيق دخول نيسب واحد إلى المقطع الحرج، وعند تنفيذ كامل الكود ضمن المقطع الحرج يقوم هذا النيسب (الذي دخل للتو و وضع قفل على المقطع الحرج) بتحرير القفل قبل الخروج من المقطع الحرج ليستعمله نيسب أخر. ذلك من أجل الحصول على الخرج الصحيح الذي نريده.

ملاحظات:

1- يجب علينا الحذر عند استعمال القفل بحيث نقفل أصغر جزء من الكود (الشيفرة) التي نحتاج إلى حمايته ضمن البرنامج.

2- هذه الأفكار بالإضافة إلى جميع الأفكار التي سنشرحها في هذا الفصل لاحقاً هي موجودة ومطبقة من قبل نظام التشغيل ولكن على نطاق المهام (Processes) وسنراها على نطاق النياسب.

استخدام المقاطع الحرجة والقفل

تقدم لغة C# حلاً واضحاً للمقاطع الحرجة وذلك عن طريق العبارة Lock. تؤكد هذه العبارة على أن كتلة الشيفرة التي تليها والموضوعة ضمن قوسين { } ستُنفذ من قبل نيسب واحد على الأكثر في لحظة ما.

لنرى مقطع الشيفرة التالي:

```
Static int [] arr=new int [10];

 Static int  index++;

      if (index>=10)
          return;

      Console.WriteLine(arr[index]);
```

- تقوم هذه الشيفرة بزيادة الدليل (index) في كل مرة ثم نقوم بفحص الشرط قبل الولوج إلى المصفوفة، لنفرض أن هذه الشيفرة ينفذها مئة نيسب أو أكثر من ذلك (مع العلم أن كلاً من المتحول index والمصفوفة هي Static (مشتركة بين الأغراض).

- كما قلنا سابقاً يقوم المعالج بتنفيذ جزء من شيفرة كل نيسب، ولنفرض أننا وصلنا إلى القيمة Index=9 ولنفرض أن النيسب قد وصل إلى تعليمة الطباعة ثم فقد التحكم وانتقل التحكم إلى نيسب أخر والذي بدورة قام بزيادة قيمة المتحول index (أصبحت قيمته تساوي 10) ثم قام بتفحص الشرط فوجده محقق لذلك ينفذ النيسب التعليمة return، بعد ذلك لنفرض أنه عاد التحكم إلى النيسب الأول (الذي يقف عند تعليمة الطباعة) ثم يقوم بالحصول على القيمة arr[index] عندها نكون قد خرجنا خارج حدود المصفوفة وعندها يرمي البرنامج استثناء وينهار البرنامج.

- في الحقيقة إن تنفيذ هذا البرنامج على حاسب يحوي معالجاً واحداً قد يولد المشكلة السابقة الذكر ولكن تنفيذ البرنامج على حاسب يحوي عدة معالجات عندها لن يحدث هذا الخطأ (لأنه ضمن الشريحة الزمنية المخصصة للنيسب يقوم بتنفيذ كامل المقطع الحرج، في حال كان المقطع الحرج كبير نوعاً ما عندها قد تحدث المشكلة التي تكلمنا عنها، وهذا ما سنراه لاحقاً).

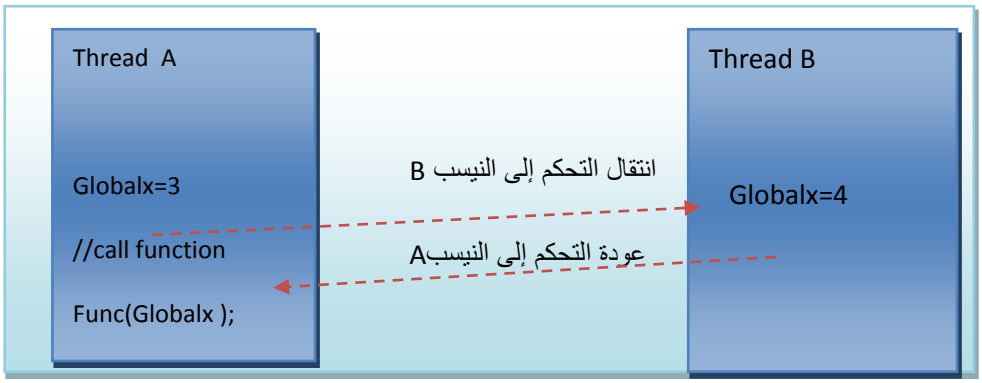
و هذه الشيفرة كاملة:

```
using System;
using System.Threading;

namespace MultiThreading
{
    class Program
    {
        static int index = 0;
        static int [] arr=new int [10];

        Program()
        {
            for (int i = 0; i < arr.Length; i++)
            {
                arr[i] = i;
            }
        }

        public   void process()
        {
               index++;

                if (index >=10)
                    return;

                Console.WriteLine(arr[index]);
        }
        static void Main(string[] args)
        {

            Program p = new Program();

            Thread[] arr = new Thread[10];

            for (int i = 0; i<10; i++)
            {
                Thread t1 = new Thread(new ThreadStart(p.process));

                arr[i] = t1;
            }

            for (int i = 0; i < 10; i++)
            {
                arr[i].Start();
            }
        }
    }
}
```

التزامن بين النياسب Threads Synchronization

نحتاج إلى المزامنة بين النياسب لسببين هما:استخدام موارد مشتركة والتواقت (Concurrency).

كل نيسب له مكدس خاص به ومجموعة مسجلات خاصة به (المكدس يحوي بارمترات الإستدعاءات بالإضافة المتحولات المحلية)، أما المتحولات العامة (Static Variables) فهي مشتركة بالنسبة لكل النياسب (تنتمي إلى المهمة (Porcess) ككل)، لذلك يمكن الوصول إليها من قبل جميع النياسب الموجودة ضمن المهمة (Porcess) وذلك يعطي فرصة لظهور مشاكل حقيقية صعبة الحل.لنرى الشكل التالي والذي يوضح المشكلة:



في الشكل السابق لدينا مهمة (Porcess) تحوي نيسبين A , B وكل منهما يستطيع الوصول إلى المتحول العام.

في البداية يضع A القيمة 3 في المتحول العام Globalx في هذه اللحظة يتنقل التحكم من النيسب A إلى النيسب B (طبعاً يتم حفظ سياق النيسب A والذي هو عبارة عن المكدس الخاص به بالإضافة إلى مجموعة مسجلات أحد هذه المسجلات لمعرفة التعليمة التالية التي نريد تنفيذها عند العودة للتنفيذ مرة أخرى)، والآن النيسب B يقوم بتغيير قيمة المتحول العام Globalx إلى القيمة 4 وعندما يعود التنفيذ إلى النيسب A يقوم باستدعاء التابع Func دون الانتباه إلى أن المعطيات قد تغيرت وبالتالي لدينا خطأ.

في الحقيقة من الصعب جداً أكتشاف هذا النوع من الأخطاء لأنها تعتمد أصلاً على التوقيت. أي أننا إذا نفذنا البرامج مرة أخرى قد لا تظهر لنا المشكلة التي كنا نتكلم عنها (لم يحدث تبديل بالسياق بين النيسبين A , B).

لا تقتصر المشاركة على التشارك بالمتحولات العامة إنما يمكن أن تظهر مع أي مورد مشترك مثل الملفات وقواعد المعطيات.

السبب **الثاني** للمزامنة **هو التواقت بين النياسب**: إذا كان من المهم أن ينتظر نيسب ما وصول نيسب أخر إلى نقطة معينة بالعمل (مثلاً النيسب الأول يحضر بيانات والثاني ينتظر أكتمال هذه البيانات).

تحقيق التزامن Synchronization Implementation

- سنعرض مسألة ولوج أكثر من نيسب على حساب مصرفي والقيام بعمليات سحب مال من الرصيد.
- عندما يكون لدينا مناقلة معينة (Transcation) في طور التنفيذ فأنه يجب على هذه المناقلة أن تتم بالكامل قبل الشروع في مناقلة أخرى وهذا ما سنقوم به من خلال التزامن بين النياسب، وإلا فأن النتائج ستكون خاطئة (يمكن أن يصبح الرصيد ضمن الحساب ذو قيمة سالبة !).

سنقوم بعرض المثال مرة من دون تزامن و مرة مع تزامن.

```
class Account
{

    int balance;
    bool Dolock;

    Random r = new Random();

    public Account(int initial,bool Dolock)
    {
        balance = initial;

        this.Dolock = Dolock;
    }

    int Withdraw(int amount,bool Dolock)
    {

        // This condition will never be true unless the lock statement
        // is commented out:
        if (balance < 0)
        {
            throw new Exception("Negative Balance");
        }

        // Comment out the next line to see the effect of leaving out
        // the lock keyword:
        if (Dolock)
        {
            lock (this)
            {
                if (balance >= amount)
                {
                    Console.WriteLine("Balance before Withdrawal :  " + balance);
                    Console.WriteLine("Amount to Withdraw        : -" + amount);
                    balance = balance - amount;
                    Console.WriteLine("Balance after Withdrawal  :  " + balance);
                    return amount;
                }
                else
                {
                    return 0; // transaction rejected
                }
            }
        }
        else
```

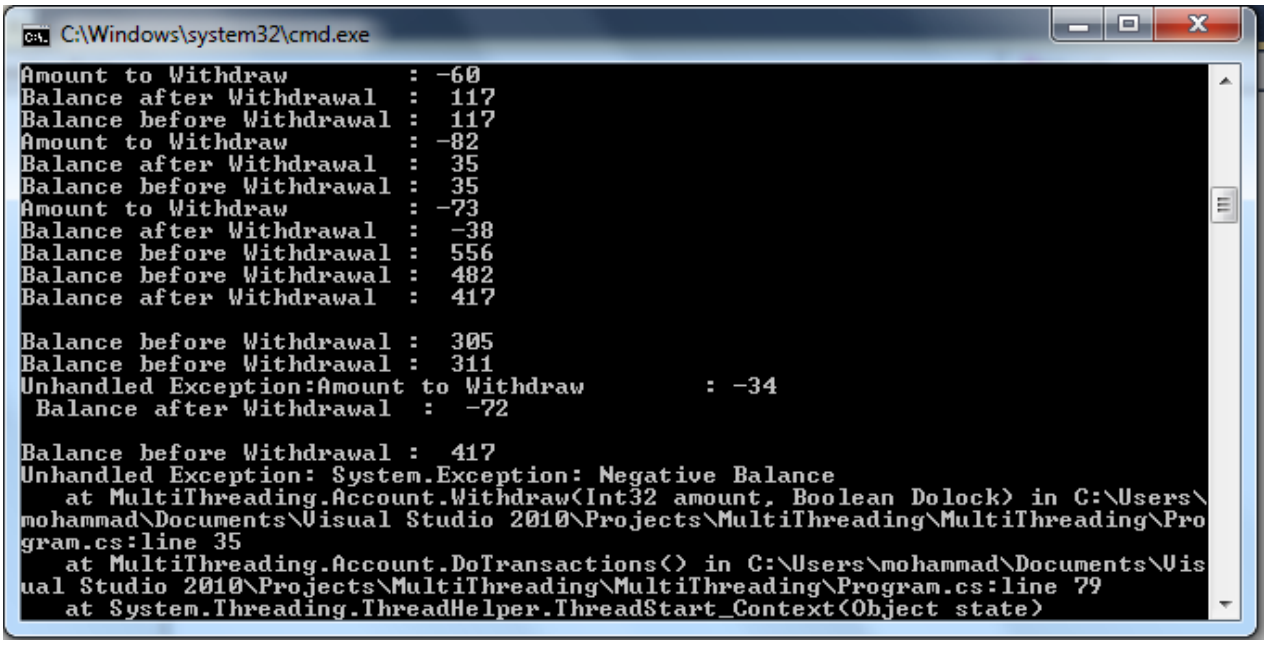
```
        {
            if (balance >= amount)
            {
                Console.WriteLine("Balance before Withdrawal :  " + balance);
                Console.WriteLine("Amount to Withdraw        : -" + amount);
                balance = balance - amount;
                Console.WriteLine("Balance after Withdrawal  :  " + balance);
                return amount;
            }
            else
            {
                return 0; // transaction rejected
            }
        }
    }

    public void DoTransactions()
    {
        for (int i = 0; i < 100; i++)
        {
            Withdraw(r.Next(1, 100),Dolock);
        }
    }
}

class Test
{
    static void Main()
    {
        Thread[] threads = new Thread[10];

        Account acc = new Account(1000,true);

        for (int i = 0; i < 10; i++)
        {
            Thread t = new Thread(new ThreadStart(acc.DoTransactions));
            threads[i] = t;
        }
        for (int i = 0; i < 10; i++)
        {
            threads[i].Start();
        }
    }
}
```

شرح الكود:

- في البداية لدينا الصف Account والذي يحوي على قيمة الحساب بالإضافة إلى متحول منطقي للدلالة على استعمال قفل (Lock) أم لا ويتم تحديد هذه القيم عند إنشاء الغرض (في الباني).
- في الMain() قمنا بإنشاء حساب ثم أنشأنا 10 نياسب وكل هذه النياسب تتنافس على نفس الغرض (الحساب المصرفي)، ثم قمنا بتشغيل هذه النياسب العشرة وهنا نميز حالتين:

1- عند إنشاء الحساب لا نريد أن نجعل النياسب متزامنة في عملها لذلك نمرر للباني القيمة false بالإضافة إلى قيمة الرصيد الإبتدائية. في هذه الحالة تتداخل النتائج ولانعرف ماذا يحصل لأن المناقلات لا تتم بشكل كامل وإنما بشكل جزئي لذلك يصبح الرصيد سالب مما يؤدي إلى رمي استثناء وبالتالي إيقاف البرنامج (في الكود نفحص الرصيد إذا كان سالب فأننا نقوم برمي استثناء لإيقاف النيسب).

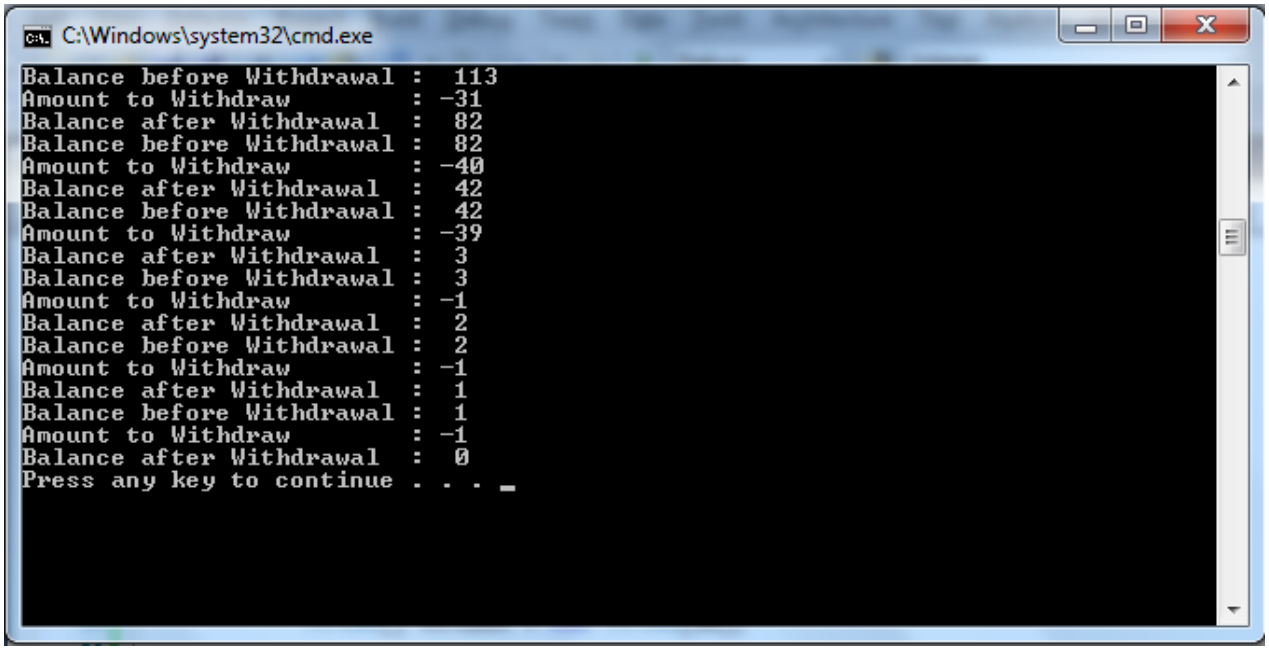
لنرى الخرج الآن:



نلاحظ ظهور استثناءات كثيرة (الخرج خاطئ).

2- أما في حال قمنا بوضع قيمة true (أي نريد استعمل قفل على الغرض من أجل المزامنة بين النياسب) وذلك عند إنشاء الحساب فأنه في لحظة ما لايمكن لنيسبين أن يدخلا إلى المقطع الحرج.
فقط نيسب واحد في لحظة ما يدخل إلى المقطع الحرج (يضع قفل على الغرض وبقية النياسب تكون في حالة انتظار). و بعد أن ينتهي هذ النيسب من تنفيذ كامل المقطع الحرج(والذي يمثل مناقلة كاملة) يقوم بتحرير القفل، ثم يقوم نظام التشغيل بإختيار أحد النياسب التي في حالة إنتظار للدخول في المقطع الحرج. وثم يقوم النيسب المختار يقوم بكل ما قام به النيسب السابق (إقفال الغرض عند الدخول إلى المقطع الحرج وتحرير القفل عند الخروج من المقطع الحرج) وهكذا نحصل على خرج صحيح.

لنرى الخرج الآن.



نلاحظ عدم وجود أخطاء وهذا هو المطلوب.

أصناف التزامن Synchronization Classes

يحتوي فضاء الأسماء System.Threading العديد من الأصناف التي يمكن أن نستخدمها لتحقيق التزامن، الفقرات التالية ستوضح هذه الأصناف.

الصف Interlocked

يحتوي هذه الصنف على مناهج (method) آمنة (Safe) لتقديم العمليات الآمنة على المتحولات المشتركة بين عدة نياسب.

هذه المناهج هي ذرية (atomic) أي أن المنهج هو عبارة عن مناقلة (Transaction) فإما أن تتم بشكل كامل دون مقاطعة من المعالج أو أن لا تتم أبداً .

من هذه المناهج :

1- **Increment :** تزيد قيمة متحول.
2- **Decrement** : تنقص قيمة متحول.
3- **Exchange:** يضع قيمة في متحول ويعيد القيمة الأصلية (القديمة) للمتحول.
4- **CompareExchange:** تقارن بين قيمتين (Comparand, Destination) وإذا كانتا متساويتين تضع قيمة متحول أخر (Value) في المتحول (Destsination).

- جميع هذا المناهج (methods) هي مناهج ذرية أي لا يمكن أن تقاطع من المعالج عند التبديل بين النياسب (أي تنفذ كتعليمة واحدة).
- في الحقيقة إن عملية الزيادة (Increment) والإنقاص (Increment) لا تُنفذ بشكل ذري (أي كتعليمة واحدة)على معظم المعالجات، بل تحتاج إلى ثلاثة مراحل هي كالتالي:
  1- تحميل قيمة المتحول من الذاكرة إلى أحد مسجلات المعالج.
  2- زيادة أو إنقاص القيمة.
  3- تخزين القيمة الجديدة للمتحول في الذاكرة الرئيسية.
- إذا لم نستخدم هذه المناهج (methods) السابقة (الخاصة بالزيادة والإنقاص) فإن النيسب الأول الذي يقوم بالزيادة أو الإنقاص للمتحول المشترك يمكن أن يُقاطع من قبل وحدة المعالجة في أول مرحلتين من أجل أن يقوم نيسب ثاني بعملية الزيادة أو الإنقاص لنفس المتحول المشترك، لنفرض أن هذا النيسب قام بالعمليات الثلاثة السابقة(1- تحميل 2-زيادة أو إنقاص 3- تخزين). وعندما يعود النيسب الأول لإكمال عمله يقوم بعملية التخزين في المتحول المشترك (يكتب فوق القيمة التي زادها أو أنقصها النيسب الثاني) وبالتالي لدينا معطيات خاطئة.

لهذا السبب نحن بحاجة إلى هذا الصف (Interlocked) والذي يؤمن لنا عمليات على شكل مناقلات (أي تُنفذ كتعليمة واحدة ولا يمكن مقاطعتها).

لنرى الآن الكود التالي والذي يستخدم الصنف Interlocked.

```
using System;
using System.Threading;

namespace InterlockedExchange_Example
{
   class MyInterlockedExchangeExampleClass
   {
        //0 for false, 1 for true.
        private static int usingResource = 0;

         private  const int numThreadIterations = 5;

         private const int numThreads = 4;

         static void Main()
         {
                Thread myThread;

                Random rnd = new Random();

                for (int i = 0; i < numThreads; i++)
                {
                    myThread = new Thread(new ThreadStart(MyThreadProc));

                    myThread.Name = String.Format("Thread {0} :", i + 1);

                    //Wait a random amount of time before starting next thread.
                    Thread.Sleep(rnd.Next(0, 1000));

                    myThread.Start();
                }
          }

            private static void MyThreadProc()
            {
                // numThreadIterations is  5
                for (int i = 0; i < numThreadIterations; i++)
                {
                    UseResource();

                    //Wait 1 second before next attempt.
                    Thread.Sleep(1000);
                }
            }

            static bool UseResource()
            {
                //0 indicates that the method is not in use.
               //Exchange method Return The Original value of usingResource(before Assigning
              // 1 to usingResource variable)
                if (0 == Interlocked.Exchange(ref usingResource, 1))
                {
                    Console.WriteLine("{0} acquired the lock", Thread.CurrentThread.Name);

                   //** : Code to access a resource that is not thread safe would go here.

                    //Simulate some work
                    Thread.Sleep(500);
```

```
                Console.WriteLine("{0} Release lock", Thread.CurrentThread.Name);

                //Release the lock
                Interlocked.Exchange(ref usingResource, 0);
                return true;
            }
            else
            {
             Console.WriteLine("  {0} was denied the lock, Thread.CurrentThread.Name);

                return false;
            }
        }
    }
}
```

شرح الكود:

نقوم في هذا البرنامج بتحقيق التزامن بين مجموعة من النياسب والتي تنفذ كل منها مجموعة من التعليمات، حيث يقوم كل نيسب باستعمال المورد المشترك والذي هو عبارة عن متحول Static حيث نقوم بتغيير قيمته بين ال 0 وال 1، حيث تدل القيمة 0 على أن المورد غير مستخدم والقيمة 1 تدل على أن المورد مستخدم، يقوم كل نيسب باستلام القفل ثم يقوم بعمله داخل المقطع الحرج، وهنا نضع المعالجة التي نريدها على المورد المشترك وقد أشرنا إلى ذلك ضمن الكود عن طريق وضع تعليق **، ثم قمنا بتنويم النيسب لمدة معينة لمحاكاة لعملية المعالجة، وبعد أن ينتهي هذا النيسب يقوم بتحرير القفل (Release Lock) عن طريق المنهج Exchange الخاص بالصف Interlocked.

لنلقي نظرة على المنهج UseResource والذي ينفذه كل نيسب من النياسب الأربعة ضمن التطبيق السابق ضمن حلقة لخمس مرات، حيث يقوم كل نيسب بالدخول إلى المقطع الحرج عن طريق التعليمة:

```
if (0 == Interlocked.Exchange(ref usingResource, 1))
```

حيث تقوم هذه التعليمة باسناد القيمة 1 إلى المتحول المشترك usingResource وذلك عن طريق المنهج Exchange والذي يعيد قيمة المتحول القديمة قبل التعديل، فإذا كانت تساوي الصفر عندها يكون المورد غير مستخدم لذلك يتم الدخول في المنطقة الحرجة.

أما بقية النياسب فتقوم بتنفيذ الشرط السابق ولكن المنهج Exchange برد القيمة 1 (القيمة القديمة للمتحول المشترك)، وهذا يدل على أن المورد مستخدم الآن لذلك لا يتم الدخول إلى هذا الشرط، ثم يقوم النيسب الذي لم يستطع أن يحصل على المورد المشترك بطباعة جملة كمايلي:

```
Console.WriteLine(" {0} was denied the lock", Thread.CurrentThread.Name);
```

ثم يردالمنهج usingResource قيمة false.

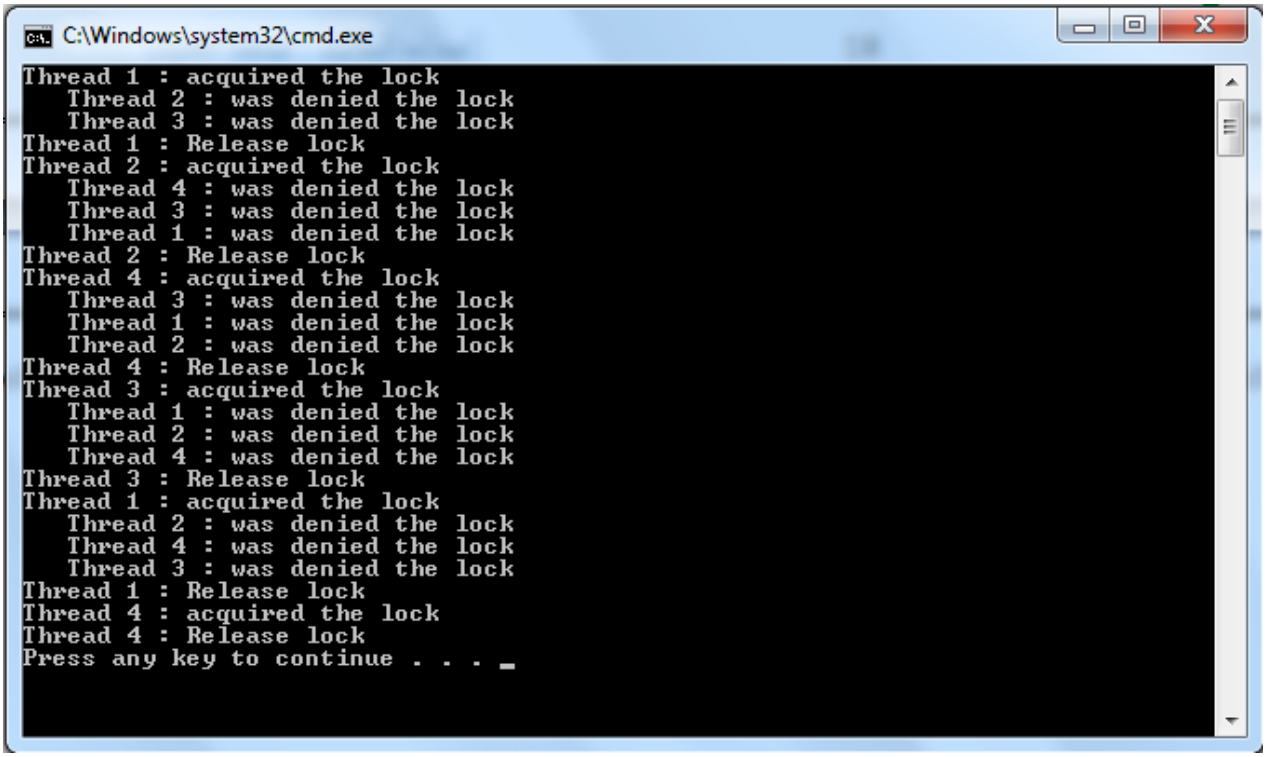
وهكذ من أجل بقية النياسب (تفشل في الدخول إلى المقطع الحرج).

طبعاً النيسب الذي قام بالدخول إلى المقطع الحرج عندما يقوم بالخروج فأنه يحرر القفل عن طريق التعليمة:

```
Interlocked.Exchange(ref usingResource, 0);
```

وهكذا يعود النيسب الذي خرج من المقطع الحرج لتنفيذ المنهج usingResource مرة أخرة (لأنه يُستدعى ضمن حلقة لخمس مرات)، فإما أن يدخل إلى المقطع الحرج أو أن يفشل ويطبع عبارة فشل ويرد قيمة false.

لنرى الخرج الآن، حيث يظهر لنا ترتيب دخول النياسب إلى المنطقة الحرجة والخروج منها والنياسب التي فشلت في الدخول إلى المنطقة الحرجة.



مسألة برمجة تفرعية Parallel Programming

المثال التالي يعرض لنا نيسبين يقومان بعملية جمع سلسة من الإعداد. مثلاً، نريد جمع الأعداد من 1 إلى 1,000,000 نقوم بتوزيع العمل على نيسبين (Two Threads) كل منهما يقوم بجمع الأعداد من 1 حتى 500,000 من دون تزامن (أي لا ينتظر أحد منهما الأخر).

أرجو قراءة التعليقات فهي هامة جداً.

```
using System;
using System.Threading;

namespace MultiThreading
{
    public class ThreadSafe
    {
        // Field totalValue contains a running total that can be updated
        // by multiple threads. It must be protected from unsynchronized
        // access.
        private double totalValue = 0.0;

        // The Total property returns the running total.
        public double Total { get { return totalValue; } }

        // AddToTotal safely adds a value to the running total.
        public double AddToTotal(double addend)
        {
            double initialValue, computedValue;

            do
            {
                // Save the current running total in a local variable.
                initialValue = totalValue;

                // Add the new value to the running total.
                computedValue = initialValue + addend;

                // CompareExchange compares totalValue to initialValue. If
                // they are not equal, then another thread has updated the
                // running total since this loop started. CompareExchange
                // does not update totalValue. CompareExchange returns the
                // contents of totalValue, which do not equal initialValue,
                // so the loop executes again.
            }
            while (initialValue != Interlocked.CompareExchange(ref totalValue,
                computedValue, initialValue));

            // If no other thread updated the running total, then
            // totalValue and initialValue are equal when CompareExchange
            // compares them, and computedValue is stored in totalValue.
            // CompareExchange returns the value that was in totalValue
            // before the update, which is equal to initialValue, so the
            // loop ends.

            // The function returns computedValue, not totalValue, because
            // totalValue could be changed by another thread between
```

```
            // the time the loop ends and the function returns.
            return computedValue;
        }
    }


    public class Test
    {
        // Create an instance of the ThreadSafe class to test.
        private static ThreadSafe ts = new ThreadSafe();

        private static double control;

        private static ManualResetEvent mre = new ManualResetEvent(false);

        public static void Main()
        {
            // Create two threads, name them, and start them. The
            // thread will block on mre.

            Thread t1 = new Thread(TestThread);
            t1.Start();

            Thread t2 = new Thread(TestThread);
            t2.Start();

            // Now let the threads begin adding numbers to  the total.
            mre.Set(); // Notify All Waiting Threads

            // Waiting The Main Thread until all the threads are done(t1,t2).
            t1.Join();
            t2.Join();

           Console.WriteLine("Thread safe: {0}  Ordinary Double: {1}", ts.Total, control);
        }


        private static void TestThread()
        {

           //Wait until the signal.
            mre.WaitOne(); //Wait Current Thread until The mre call set method

            for (int i = 1; i <= 10; i++)
            {
                // Add to the running total in the ThreadSafe instance, and
                // to an ordinary double.
                double testValue =i;

                control += testValue;//ordinary Add

                ts.AddToTotal(testValue); // Thread safe add
            }
        }
    }

}
```

شرح الكود:

نريد في هذا المثال أن يقوم بجمع الأعداد من 1 حتى 20 ونريد تقسيم العمل على نيسبين كل منهما يقوم بجمع الأعداد من 1 حتى 10 والنتيجة يتم تخزينها ضمن غرض ts من الصنف ThreadSafe وتحديداً ضمن المتحول totalValue، وفي النهاية نقوم بطباعة قيمة الجمع العادية(المخزنة ضمن المتحول Control) مع قيمة الجمع عن طريق نيسبين.

قمنا بإنشاء نيسبين و شغلنا كل منهما ثم قام كل من النيسبين بتنفيذ المنهج TestThread وفي بدايته استدعاء للمنهج WaitOne الخاص بالصف ManualResetEvent، يقوم هذا المنهج بتعليق عمل النيسب الحالي (حالة إنتظار) إلى أن يستقبل هذا النيسب إشارة متابعة، وهذه الإشارة يتم إرسالها عن طريق المنهج set الخاص بالصف ManualResetEvent (وذلك ضمن الـ main Thread - يجب أن لا ننسى أن ال main method هو نيسب-) والذي يعيد كل النياسب التي في حالة إنتظار إلى العمل مجدداً.

ثم قمنا في داخل ال main Thread بإستدعاء المنهج Join لكل من النيسبين t2،t1 والذي يقوم بتعليق عمل النيسب الحالي(main Thread) إلى أن ينتهي كل من النيسبين t1, t2 من عملهما (لكي لا يقوم ال main Thread بتنفيذ عملية الطباعة الأخيرة لنتائج لم يكتمل حسابها بعد). إلى الآن نكون قد وضّحنا سيناريو تنفيذ الكود. ولكن كيف يعمل كل من النيسبين t1, t2 ؟

في الحقيقة كل من النيسبين t2 , t1 يقوم بتمرير قيمة العداد(i) إلى المنهج AddToTotal الخاص بالغرض ts ليقوم بجمعها إلى المتحول totalValue الخاص بالغرض ts، وضمن هذا المنهج نقوم في البداية بتخزين القيمة الحالية للمتحول totalValue ضمن متحول أخر اسمة initialValue ثم تتم عملية الجمع ويتم تخزين نتيجة الجمع ضمن متحول أخر اسمه computedValue.

ثم يقوم النيسب عن طريق المنهج CompareExchange الخاص بالصف Interlocked بمقارنة القيمة الحالية للمتحول totalValue مع القيمة القديمة له (والمخزنة ضمن المتحول initialValue) فإذا كانت القيم متساوية فإنه هذا المنهج (CompareExchange) يخزن قيمة المتحول computedValue في المتحول totalValue (تثبيت عملية الجمع) و يعيد قيمة المتحول totalValue الأصلية (قبل التعديل والتي تساوي قيمة المتحولinitialValue) ولذلك يخرج من الحلقة (Do - while)، وإلا إذا لم تكن القيم متساوية عندها يكون النيسب الأخر قد قام بتعديل قيمة المتحول totalValue لذلك يجب إعادة عملية الجمع (نعود في الحلقة مرة ثانية وذلك لكي نحصل على نتائج صحيحة)، عملياً يعيد المنهج CompareExchange قيمة totalValue والتي لا تساوي قيمة المتحول initialValue لذلك يعود النيسب بالحلقة مرة أخرى. لنرى الخرج الآن : (مجموع القيم من 1 إلى 20 هو 110).

تنويه:

لنفرض أننا نريد الآن أن نقوم بجمع الأعداد من 1 حتى 50 مليون ونريد أن نرى كم يتحتاج المعالج إلى زمن لتنفيذ البرنامج في حال نيسب واحد يعمل، وفي حالة تقسيم العمل على 10 نياسب (كل نيسب يقوم بجمع الأعداد من 1 حتى 500 ألف) لنلاحظ الفائدة التي حققناها من تعدد النياسب في الوقت الذي نحتاجه لتنفيذ البرامج.

في الحقيقة الفائدة تكون أكبر في المعالجات متعددة النوى(مثلاً : Core 2 Duo).

كل ما علينا تغييره عن الكود السابق هو فقط تغيير ال main فقط:

يمكنكم مشاهدة النتائج بأنفسكم.

```
public static void Main()
{
    // Create two threads, name them, and start them. The
    // thread will block on mre.

    Thread [] arr =new Thread[10];

    for (int i = 0; i < arr.Length; i++)
    {
        Thread t1 = new Thread(TestThread);
        arr[i] = t1;
    }

    for (int i = 0; i < arr.Length; i++)
    {
        arr[i].Start();
    }


    // Now let the threads begin adding numbers to
    // the total.
    mre.Set();


    // Waiting The Main Thread until all the threads are done.
    for (int i = 0; i < arr.Length; i++)
    {
        arr[i].Join();
    }

    Console.WriteLine("Thread safe: {0}  Ordinary Double: {1}", ts.Total, control);
}
```

Semaphore Class

السيمافور هو عبارة عن آليه لتحقيق التزامن بين المهام (Processes) في مستوى نظام التشغيل، يمكن أيضاً استعماله لتحقيق التزامن بين النياسب.

يمكن تعريف السيمافور أيضاً بأنه: "عبارة عن رقم يُمثل عدد محدود من النياسب التي تستطيع الحصول على مورد مشترك" أو (حوض (Pool) من الموارد التي يتم التنافس عليها من قِبَل أكثر من نيسب).

أي أن السيمافور يستخدم من أجل التحكم في الوصول إلى حوض (Pool) من الموارد المشتركة.

يمكن لأي نيسب أن يحاول الحصول على وحدة من وحدات الموارد المشتركة ضمن الحوض (الحوض يحوي عدد محدود من الموارد).

ذكرنا سابقاً أن السيمافور هو عبارة عن عدد، هذا العدد يتم إنقاصه في كل مرة يقوم فيها نيسب معين باستخدام وحدة من الواحدات التي يؤمنها السيمافور، في نفس الوقت هذا العدد (الذي يُمثل السيمافور) يتم زيادته عندما يقوم نيسب معين بتحرير الوحدة التي استخدمها من حوض الموارد(تُسمى هذه العملية بتحرير السيمافور وهي عبارة عن منهج اسمه ()Release –فعلياً تُمثل زيادة قيمة السيمافور بمقدار 1 لكي يتم السماح لنياسب أخرى باستعمال هذه الوحدة من حوض الموارد-)، وعندما تصبح قيمة السيمافور تساوي الصفر عندها أي نيسب يقوم بطلب وحدة من واحدات المنبع سيقوم بالإنتظار على هذا السيمافور وذلك عن طريق المنهج ()Waitone وعندما يقوم أحد النياسب التي تستخدم السيمافور بتحرير السيمافور ()Relesase عندها يقوم نظام التشغيل بإختيار أحد النياسب من رتل الإنتظار والذي يحوي النياسب التي تنتظر على هذا السيمافور.

مثال:

لنفرض أن لدينا طابعة وهذه الطابعة تستطيع أن تعالج ثلاثة طلبات على الأكثر، عند ذلك يقوم نظام التشغيل بتخصيص سيمافور خاص بهذه الطابعة وتكون قيمته تساوي 3. لنفرض أن أربعة نياسب قاموا بطلب الحصول على مورد من هذا السيمافور عندئذٍ، ثلاثة نياسب سوف تستخدم موارد الطابعة و النيسب الرابع وكل النياسب التي تريد الحصول على مورد من هذا السيمافور سوف تنتظر على السيمافور(لأن قيمة السيمافور الآن هي 0 –أي أن المورد مستخدم من قِبَل ثلاثة نياسب-) إلى أن يقوم أحد النياسب الثلاثة الأولى بتحرير السيمافور (زيادة قيمة السيمافور بمقدار 1)، وعندها يقوم نظام التشغيل بإختيار أحد النياسب من رتل الإنتظار ليستخدم المورد الذي تم تحريره وتبقى النياسب المتبقية في حالة إنتظار إلى حين تحرير أحد الموارد من حوض الموارد.

ملاحظة 1: قيمة السيمافور هي عدد أكبر أو يساوي الصفر ويعبر عن عدد النياسب التي تستطيع أن تحصل على المورد مباشرة و من دون إنتظار.

ملاحظة 2: في الحقيقة يمكن لنيسب أن يقوم بطلب الحصول على مورد من سيمافور أكثر من مرة، مثلاً يمكن لنيسب معين أن يطلب الحصول على مورد من سيمافور ثلاثة مرات وعندها نقوم باستدعاء المنهج ()WaitOne ثلاثة مرات، وعندما نقوم بتحرير السيمافور يجب استدعاء المنهج ()Release ثلاثة مرات أيضاً أو يمكن أن نستدعي المنهج (3)Release والذي يأخذ كبارمتر عدد مرات طلب الحصول على مورد من السيمافور.

الجدير بالذكر أنه يوجد نوعين من السيمافورات:

1- named system semaphores: يتم إنشاؤه عن طريق وضع اسم للسيمافور ضمن الباني عند إنشاء غرض سيمافور (هذا الأسم معروف من قِبَل نظام التشغيل)، وهذا النوع مُستخدم من قِبَل نظام التشغيل لتحقيق التزامن بين المهام (Processes).

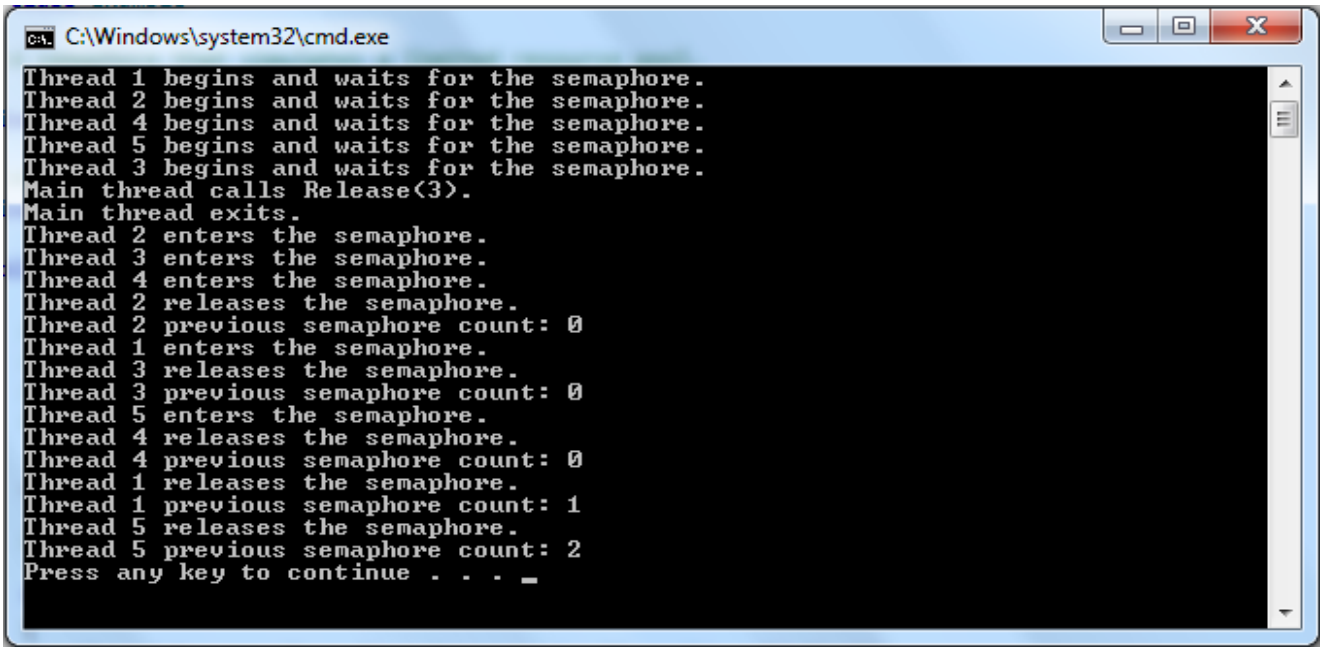
2- local semaphore: موجود على مستوى المهمة (Process) فقط ويستخدم من قِبَل أي نيسب ضمن المهمة وهو مستخدم لتحقيق التزامن بين النياسب ضمن المهمة. الآن لنرى الكود التالي.

أرجو قراءة التعليقات فهي هامة جداً.

```
using System.Threading;
using System.IO;

namespace MultiThreading
{

    public class Example
    {
        // A semaphore that simulates a limited resource pool.
        private static Semaphore _pool;

        public static void Main()
        {
            // Create a semaphore that can satisfy up to three
            // concurrent requests. Use an initial count of zero,
            // so that the entire semaphore count is initially
            // owned by the main program thread.

            _pool = new Semaphore(0, 3);

            // Create and start five numbered threads.

            for (int i = 1; i <= 5; i++)
            {
                Thread t = new Thread(new ParameterizedThreadStart(Worker));

                // Start the thread, passing the number
                t.Start(i);
            }

            // Wait for half a second, to allow all the
            // threads to start and to block on the semaphore.
            //
            Thread.Sleep(500);

            // The main thread starts out holding the entire
            // semaphore count. Calling Release(3) brings the
            // semaphore count back to its maximum value, and
            // allows the waiting threads to enter the semaphore,
            // up to three at  the same time.

            Console.WriteLine("Main thread calls Release(3).");
```

```
            _pool.Release(3); //Increment The initial count of Semaphore 3 times

            Console.WriteLine("Main thread exits.");
        }

        private static void Worker(object num)
        {
            // Each worker thread begins by requesting the
            // semaphore.
            Console.WriteLine("Thread {0} begins " +
                "and waits for the semaphore.", num);

            _pool.WaitOne();

            Console.WriteLine("Thread {0} enters the semaphore.", num);

            // The thread's "work" consists of sleeping for
            // about a second.

            Thread.Sleep(1000 );

            Console.WriteLine("Thread {0} releases the semaphore.", num);

            // _pool.Release():Exits the semaphore and returns the previous count.
            Console.WriteLine("Thread {0} previous semaphore count: {1}",
                num, _pool.Release());

        }
    }
}
```

لنرى الخرج الآن :



في البداية تكون قيمة السيمافور تساوي الصفر وبعد تشغيل خمسة نياسب وهذه النياسب في حالة إنتظار على السيمافور، يقوم ال main Thread بزيادة قيمة السيمافور بمقدار 3، وعندها يختار نظام التشغيل ثلاثة نياسب من أصل خمسة موجودين في رتل الإنتظار (هذه النياسب هي 2 و3و4) ويبقى 1و 5 في حالة إنتظار إلى حين أن يقوم أحد النياسب (2,3,4) بتحرير السيمافور، وفعلاً يقوم النيسب رقم 2 بتحرير السيمافور ويرد القيمة السابقة للسيمافور والتي تكون تساوي الصفر (في الحقيقة هي تساوي ال 1)، وبعد ذلك يختار نظام الشتغيل النيسب رقم 1 للدخول في السيمافور(استعمال مورد من الموارد الموجودة ضمن الحوض) وعلى هذا المنوال يكمل تنفيذ بقية البرنامج.

(Mutual Exclusion) Mutex Class

يعطي هذا الصف(Mutex) طريقة بسيطة للتزامن بحيث يسمح لنيسب واحد بالدخول إلى المقطع الحرج مع إستبعاد البقية (بقية النياسب تبقى في حالة إنتظار إلى أن يقوم هذا النيسب بتحرير ال Mutex).

يستعمل ال Mutex آلية (Lock - unlock).

يسمح كائن من الصف Mutex بالوصول الحصري لنيسب واحد فقط إلى مورد ما مع إسبتعاد البقية، هذا المبدأ يُسمى الإستبعاد المتبادل[1]، يسمح هذا الصف بتحقيق التزامن بين النياسب الموجودة في مهام (Processes أو تسمى Application Domains) مختلفة.

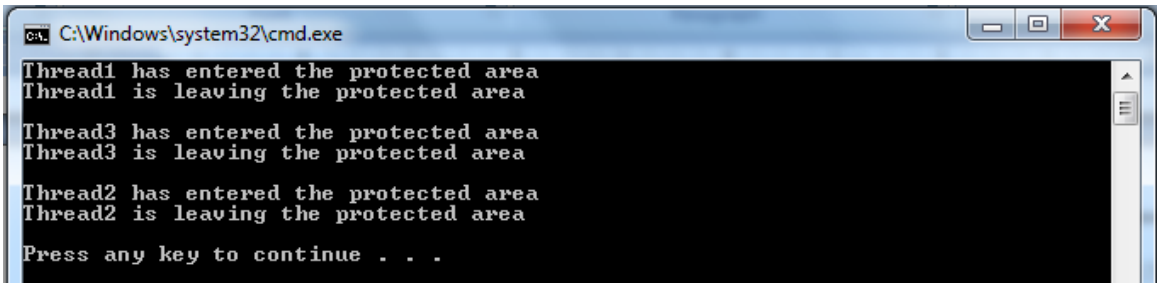
لنرى الكود التالي، أرجو قراءة التعليقات فهي هامة جداً.

```
using System;
using System.Threading;

namespace MultiThreading
{
    // This example shows how a Mutex is used to synchronize access
    // to a protected resource. Unlike Monitor, Mutex can be used with
    // WaitHandle.WaitAll and WaitAny, and can be passed across
    // AppDomain boundaries(Different Processes).

    class Test
    {
       // Create a new Mutex. The Main thread owns the Mutex
       // false means The Mutex is not Owned by calling Thread
       private static Mutex mut = new Mutex(true);

        private const int numIterations = 1;

        private const int numThreads = 3;


       static void Main()
        {
            // Create the threads that will use the protected resource.
            for (int i = 0; i < numThreads; i++)
            {
                Thread myThread = new Thread(new ThreadStart(MyThreadProc));
```

[1] للمزيد يمكن الإطلاع على الرابط التالي : http://en.wikipedia.org/wiki/Mutual_exclusion

```
            myThread.Name = String.Format("Thread{0}", i + 1);

            myThread.Start();
        }

        // ** Important Notice ** :
        // The main thread exits, but the application continues to
        // run until all foreground threads have exited.

        // Release the Mutex. That is owned by Main Thread(When we are Creating Mutex)
        mut.ReleaseMutex();

    }

    private static void MyThreadProc()
    {
        for (int i = 0; i < numIterations; i++)
        {
            UseResource();
        }
    }

    // This method represents a resource that must be synchronized
    // so that only one thread at a time can enter.
    private static void UseResource()
    {
     // Try to get Mutex , if Mutex is available
     // or
   //Wait until it is safe to enter(until Mutex is available),if Mutex is not available
        mut.WaitOne();

        Console.WriteLine("{0} has entered the protected area",
            Thread.CurrentThread.Name);

        // Place code to access non-reentrant resources here (Critical Section).

        // Simulate some work.
        Thread.Sleep(500);

        Console.WriteLine("{0} is leaving the protected area\r\n",
            Thread.CurrentThread.Name);

        // Release the Mutex.
        mut.ReleaseMutex();
    }
 }
}
```

لنرى الخرج الآن، والذي يوضح ترتيب حصول النياسب النياسب على ال Mutex وتحرير ال Mutex من قِبَلِها.



الإقفال المتبادل DeadLock

ليكن لدينا مقطع الشيفرة التالية.

```
private static Mutex mut = new Mutex();

public static void AcquireData()
 {
          mut.WaitOne();

          // الحصول على البيانات

          mut.ReleaseMutex();
 }

public static void UseData()
 {
          mut.WaitOne();

          //  استعمال البيانات

          mut.ReleaseMutex();
 }
```

مناقشة

- لنفرض أنه لدينا نيسبين الأول ينفذ المنهج AcquireData والثاني ينفذ المنهج UseData.

- في البداية يعمل النيسب الأول (ينفذ المنهج AcquireData) والذي يحاول الحصول على كائن المقطع الحصري (mut) عبر إستدعاء WaitOne()، فإذا كان المقطع الحصري متوفراً يعود الإستدعاء مباشرةً ويصبح النيسب الأول هو مالك المقطع الحصري (الكائن mut)، بعد ذلك يقوم التابع بقراءة البيانات الخاصة به. في هذه اللحظة إذا استدعى النيسب الثاني المنهج UseData فإنه سيتوقف عند الإستدعاء WaitOne() لإن المقطع الحصري غير متوفر الآن. عاجلاً أو آجلاً سينتهي تنفيذ المنهج AcquireData الخاص بالنيسب الأول ويحرر المقطع الحصري باستدعاء ReleaseMutex() وذلك يؤدي إلى عودة الإستدعاء إلى النيسب الثاني ضمن المنهج UseData وعندها يستطيع استخدام البيانات، وهكذا نكون قد حققنا التزامن بين نيسبين يريدان الوصول إلى موارد مشتركة(قد تكون متحولات أو أغراض أو حتى ملفات).

قد تبدو هذه العملية بسيطة لكن هناك العديد من المشاكل المعقدة التي تظهر عند مزامنة النياسب بهذه الطريقة، مثلاً: ماذا لو فتح useData ملفاً أو استخدم مورداً أخر يحتاجه AcquireData ؟

يُحتمل أن يقف النيسب الثاني الذي ينفذ UseData منتظراً AcquireData (الذي ينفذه النيسب الأول) حتى يحرر المقطع الحصري , لكن AcquireData يحتاج إلى الموارد التي يستخدمها UseData كي يتمكن من إتمام عمله وتحرير المقطع الحصري. وبالتالي نحن الآن في حلقة مفرغة (DeadLock) حيث يتوقف كل من النيسبين عن العمل ولا يتسطيعان التقدم .

في الحقيقة من الصعب تصميم تطبيقات متعددة النياسب بحيث لا تحدث فيها حلقات مفرغة (DeadLocks).

كيف يمكن أن تتجنب حدوث الإقفال المتبادل (DeadLock) في شيفرة برنامجك؟

الإقفال المتبادل (DeadLock) يحدث بين نيسبين (Two Threads) عندما كل من النيسبين بحاجة إلى معطيات من الأخر حتى يكمل عمله.

أي حين يكون النيسب الأول ضمن مقطع حرج يملك قفل على غرض معين وينتظر بعض المعلومات من النيسب الثاني والذي يحاول الدخول إلى المقطع الحرج الخاص به (بالنيسب الثاني والذي يحوي تعليمة إقفال على نفس الغرض الذي يقفل عليه النيسب الأول)، وبالتالي لن يستطيع النيسب الثاني من إنجاز عمله لأنه ينتظر النيسب الأول حتى يحرر القفل عن الغرض المشترك، و بالتالي يتوقف كلا النيسبي عن العمل.

هذه المشكلة تُعتبر من أعقد المشاكل التي واجهت مصمي نظم التشغيل وقد تم حلها على مستوى المهام (Processes)، و لكن حلها على مستوى النياسب (Threads) هي من مسؤولية المبرمج.

في الحقيقة تنتج هذه المشكلة (DeadLock) عن إستخدام الأقفال Locks وعن إستخدام ال Mutex أيضاً.

لنرى كيف يمكن أن توجد المشلكة وسنناقش أكثر من حل لها.

```
class Deadlock
{
    static List<int> elements = new List<int>();

    static void Thread1()
    {
        Thread.Sleep(1000);
        int[] items;
        lock (elements)
        {
            while (elements.Count < 3)
            {
                Thread.Sleep(1000);
            }

            items = elements.ToArray();// Copying The elements to items
        }
```

```
        Console.WriteLine("Thread 1 Add print elements of List");

        foreach (int  item in items)
        {
            Console.WriteLine("item {0}", item);
            Thread.Sleep(1000);
        }
    }

    static void Thread2()
    {
        Thread.Sleep(1500);

        lock (elements)
        {
            Console.WriteLine("Thread 2 Add element(30) to List");
            elements.Add(30);
        }
    }


    public static void Main()
    {
        elements.Add(10);
        elements.Add(20);

        Thread t1 = new Thread(Thread1);
        Thread t2 = new Thread(Thread2);

        t1.Start();
        t2.Start();
    }
}
```

شرح الكود:

كما نرى أن النيسب الأول هو الذي يعمل في البداية لإنه ينام لفترة أقل من الفترة التي ينامها النيسب الثاني، و بالتالي في البداية يعمل النيسب الأول فيقوم بإقفال الغرض المشترك (elements) ثم يدخل في حلقة وهذه الحلقة حتى تتحقق وينتهي تنفيذ النيسب الأول (مع تحرير القفل) يجب أن يقوم النيسب الثاني بإضافة عنصر إلى الغرض (elements).

لكن الغرض (elements) مقفول من قبل النيسب الأول والنيسب الثاني ينتظر النيسب الأول حتى يحرر القفل ليستطيع الدخول إلى المنطقة الحرجة وتنفيذ تعليمة الإضافة للقيمة 30 إلى الغرض (elements). وهكذا يبقى النيسب الأول في حالة إنتظار للنيسب الثاني بينما النيسب الثاني ينتظر النيسب الأول حتى يحرر المقطع الحرج وهذا هو ال DeadLock.

في الحقيقة الجزء من الكود الملون باللون الأحمر هو الذي تسبب بالمشلكة.

تنويه:

يمكن حل المشكلة السابقة بإن نجعل النيسب الثاني ينام لمدة أقل من النيسب الأول، ضمن المنهج Thread2 الخاص بالنيسب الثاني نجعل مدة النوم بمقدار 500 بدلاً من 1500 وبالتالي يعمل النيسب الثاني في البداية (يقوم بعملية الإضافة) ثم يعمل النيسب الأول و بذلك نحصل على نتائج صحيحة ولا يتم حدوث DeadLock.

و يكون الخرج كما يلي:

ولكن هذا الحل لا يفي بالغرض دائماً لأن هذه المشكلة هي مشكلة توقيت، لكننا نريد حلاً شافياً لهذه المشكلة.

الحل الشافي يكمن في استخدام آلية إقفال متقدمة تقدم لنا ميزة إضافية لم نشاهدها في التعليمة Lock أو الصف Mutex، هذه الميزة هي إنتظار مع إيقاظ (Watit/Notify) للنياسب، وهذا ما يزودنا بهالصف Monitor حيث نجعل النيسب الأول ينتظر (Wait) إلى حين أن تصبح المعلومات جاهزة من قبل النيسب الأخر الذي ينفذ شيفرته الخاصة ثم يقوم بإيقاظ النيسب الأول ليستطيع إكمال عمله دون أي توقف.

ملاحظة هامة:

إن التعليمة:

```
Thread.Sleep(1000);
```

تقوم بتنويم النيسب الحالي لمدة محدد (بالميلي ثانية)، ولكن النيسب النائم يحتفظ بالقفل معه خلال فترة النوم. هذه التعليمة تسبب إنتظار نشط (busy waiting)، على عكس التعليمة Monitor.Wait(elements) والتي لا تسبب إنتظار نشط[1].

أما التعليمة التالية:

```
Monitor.Wait(elements);
```

تقوم بتحرير القفل كي يتمكن نيسب أخر من استلام القفل والدخول في المقطع الحرج ليعمل على الغرض المشترك، ثم يقوم بإيقاظ النيسب النائم مع إيقاف عمل النيسب الحالي (حالة إنتظار) نيسب أخر حتى يرسل له إشارة إيقاظ (تتم عن طريق المنهج Monitor.Pulse(elements);).

[1] للمزيد يمكن الإطلاع على الرابط التالي : http://en.wikipedia.org/wiki/Busy_waiting

لنرى حل المشكلة السابقة باستعمال الصف Monitor.

أرجو قراءة التعليقات فهي هامة جداً.

```
class SolveDeadlockByMonitor
{
    static List<int> elements = new List<int>();

    static void Thread1()
    {
        Thread.Sleep(1000);
        int[] items;

        //Begin  of Critical Section
        Monitor.Enter(elements);// Thread Acquires Lock

            while (elements.Count < 3)
            {
           //Release Lock and Waiting until Thread 2 call Monitor.pulse(elements);
                Monitor.Wait(elements);
            }
            items = elements.ToArray();// Copying The elements to items

       //End of Critical Section
        Monitor.Exit(elements); //Thread Releas Lock on elements
        Console.WriteLine("Thread 1 Add print elements of List");
        foreach (int  item in items)
        {
            Console.WriteLine("item {0}", item);
            Thread.Sleep(1000);
        }
    }
    static void Thread2()
    {
        Thread.Sleep(1500);
        //Begin  of Critical Section
        Monitor.Enter(elements);// Thread Acquires Lock

            Console.WriteLine("Thread 2 Add element(30) to List");
            elements.Add(30);

            Monitor.Pulse(elements);//Release Lock and Notify Thread1

        //End of Critical Section
        Monitor.Exit(elements); //Thread Releas Lock on elements
    }

    public static void Main()
    {
        elements.Add(10);
        elements.Add(20);

        Thread t1 = new Thread(Thread1);
        Thread t2 = new Thread(Thread2);

        t1.Start();
        t2.Start();
```

```
    }
}
```

شرح الكود:

نلاحظ أن تعليمات التنويم (sleep) ضمن النيسبين لا تؤثر على عمل البرنامج أي لا يهم أي من النيسبين سيبدأ بالتنفيذ لأن الصف Monitor قدم لنا حلاً لمشلكة التوقيت عن طريق الآلية (Watit/Notify).

لنرى الخرج الآن .

Monitor Class

- يقدم الصنف Monitor آلية عامة لمزامنة وصول النياسب إلى الأغراض باستخدام الأقفال.

- لكل كائن (object) فيNET. قفل يتم الحصول عليه من قبل نيسب واحد، وبذلك يمنع القفل جميع النياسب التي تريد الوصول إلى الغرض والدخول في المقطع الحرج (تبقى النياسب في حالة إنتظار لكي يقوم النيسب الذي يملك القفل بتحريره).

الصف Monitor يمللك المزايا التالية:

1- مرتبطة بغرض ويتم الحصول على المراقب(Monitor) عند الطلب.

2- غير مقيد، أي يمكن استدعائه في أي مكان ضمن الكود.

3- لا يمكن إنشاء غرض من هذا الصنف.

المعلومات التالية هي محفوظة من أجل كل غرض متزامن (يتم الوصول إليه من أكثر من نيسب):

1- نيسب وحيد يملك القفل.

2- رتل جاهزية (Ready Queue): يحوي النياسب التي تنتظر أن يختارها المعالج لكي تحصل على القفل مع استبعاد بقية النياسب(التي تبقى في هذا الرتل إلى أن تحصل على القفل).

3- رتل إنتظار(Waiting Queue): يحوي النياسب التي في حالة إنتظار إشارة إيقاظ من نيسب أخر لتستعيد الحصول على القفل.

الجدول التالي يوضح أهم المناهج التي يقدمها الصف Monitor.

Method	الوصف
Enter	إستلام القفل من أجل غرض محدد ويمثل هذا المنهج بداية المقطع الحرج
Wait	تحرير القفل الخاص بغرض محدد والإنتظار حتى يصل للنيسب إشارة إيقاظ (Notify) لكي يستعيد القفل.
Palse/palseAll	إرسال إشارة (Signal) إلى نيسب أو أكثر ينتظرون في رتل الإنتظار ليستعيد نيسب واحد القفل ليكمل عمله بينما البقية ينتظرون في رتل الجاهزية.
Exit	تحرير القفل من أجل غرض محدد ويمثل هذا المنهج نهاية المقطع الحرج

الجدير بالذكر أن العبارة Lock في C# و SyncLock في VB يعبرعنهما المترجم (Compiler) بالتعليمتين Monitor.Enter و Monitor.Exit.

لنرى المثال التالي والذي يوضح لنا كيفية التخاطب بين النياسب:

لنفرض أن لدينا نيسبين، النيسب الأول يقوم بإنتاج معطيات(عبارة عن رقم يتم زيادته في كل مرة)، والنيسب الثاني يقوم باستهلاك المعطيات التي أنتجهاالنيسب الأول.بطريقة متزامنة حيث يقوم النيسب الأول بإنتاج معطيات وفي هذه الحالة يكون النيسب الثاني (المستهلك) في حالة إنتظار إشارة من المنتج الذي يقوم بإيقاظ المستهلك ليستهلك المعطيات التي أنتجها، يقوم المستهلك بإستهلاك المعطيات ومن ثم يوقظ المنتج والذي يكون بحالة إنتظار إستهلاك المعطيات من قبل المستهلك وهكذا دواليك.

ملاحظة:

- يمكن تشبيه الأمر بشخص يقوم بإنتاج المعطيات ويضعها في مكان لا يتسع سوى لمنتج واحد، وبعد ذلك يخبر المستهلك بإن يستهلك المعطيات (في هذه الحالة المنتج ينتظر إشارة من المستهلك ليعود إلى أنتاج المعطيات) وبعد أن ينتهي المستهلك من إستهلاك المعطيات يوقظ المنتج ليعود إلى إنتاج المعطيات (وطبعاً يصبح المستهلك في حالة إنتظار).

- يمكن لنيسب أن يقوم بعملية إنتظار (مع تحرير القفل) عن طريق التعليمة : Monitor.Wait(object).

- يمكن لنيسب أخر أن يقوم بإيقاظ النيسب الذي ينتظر إشارة عن طريق التعليمة Monitor.Pulse(object) أو إيقاظ جميع النياسب التي تنتظر تحرير القفل الخاص بغرض معين من قِبَل مالك القفل والذي ينفذ التعليمة التالية: Monitor.PulseAll(object).

تسمى هذه المسألة بمسألةالمنتج/مستهلك(Producer/Consumer) وتعتبر هذه المسألة من أهم المسائل الواقعية والمستخدمة في البرمجة التفرعية[1].

[1] للمزيد يمكن الإطلاع على الرابط التالي : http://cs.gmu.edu/cne/modules/ipc/aqua/producer.html

عادة يكون لدينا:

N producer and M Consumer

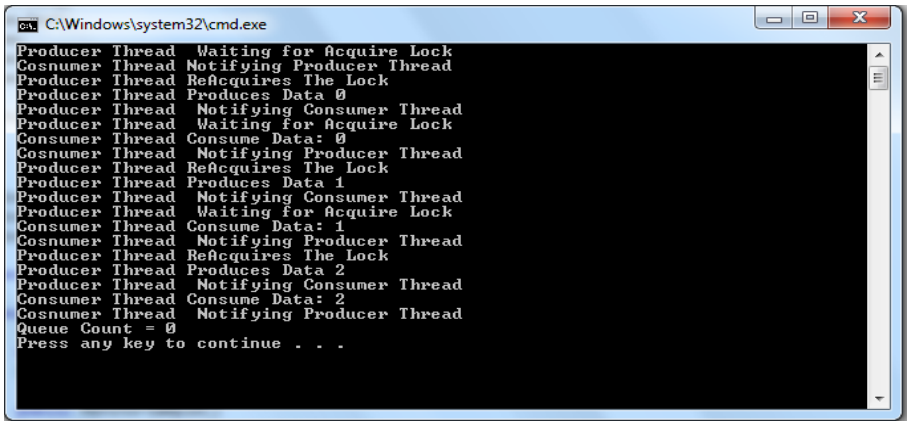
والآن يمكننا فهم الكود التالي، أرجو قراءة التعليقات فهي هامة جداً.

```
using System;
using System.Threading;
using System.Collections;

namespace MonitorCS1
{
    class MonitorSample
    {
        const int MAX_LOOP_TIME = 3;
        Queue m_smplQueue;

        public MonitorSample()
        {
            m_smplQueue = new Queue();
        }

        // Producer Thread
        public void FirstThread()
        {
            int counter = 0;

            lock (m_smplQueue)
            {
                while (counter < MAX_LOOP_TIME)
                {

                    Console.WriteLine("Producer Thread  Waiting for Acquire Lock");
                //Release Lock and  Waiting until Consumer notifying The Current Thread
                    Monitor.Wait(m_smplQueue);

                    Console.WriteLine("Producer Thread ReAcquires The Lock ");

                    //Producing Data
                    m_smplQueue.Enqueue(counter);//Push one element.

                    Console.WriteLine("Producer Thread Produces Data {0}", counter);

                    //Release Lock and Notifying Consumer Thread
                    Monitor.Pulse(m_smplQueue);

                    counter++;

                    Console.WriteLine("Producer Thread  Notifying Consumer Thread");

                }
            }
        }
```

```
        //Cosnumer Thread
      public void SecondThread()
      {
          lock (m_smplQueue)
          {
              Console.WriteLine("Cosnumer Thread Notifying Producer Thread ");

              //Release lock and  Notifying Producer Thread
              Monitor.Pulse(m_smplQueue);


  //Release Lock and Block(Waiting for 1 second)The Current Thread(Consumer) until
  (Producer Thread) produce Data
              while (Monitor.Wait(m_smplQueue, 1000))
              {
                  //Consuming Data
                  int counter = (int)m_smplQueue.Dequeue();//Pop the first element.

                  //Print the first element.
          Console.WriteLine("Consumer Thread Consume Data: {0}" ,counter.ToString());

                  //Release lock and  Notifying Producer Thread
                  Monitor.Pulse(m_smplQueue);

                  Console.WriteLine("Cosnumer Thread  Notifying Producer Thread");

              }
          }
      }

      //Return the number of queue elements.
      public int GetQueueCount()
      {
          return m_smplQueue.Count;
      }

      static void Main(string[] args)
      {
          //Create the MonitorSample object.
          MonitorSample test = new MonitorSample();

          //Create the Producer thread.
          Thread tFirst = new Thread(new ThreadStart(test.FirstThread));

          //Create the Consumer thread.
          Thread tSecond = new Thread(new ThreadStart(test.SecondThread));

          //Start threads.
          tFirst.Start();
          tSecond.Start();

          // Block (waiting) The Main Thread  until finish  two threads t1, t2
          tFirst.Join();
          tSecond.Join();

          //Print the number of queue elements.
          Console.WriteLine("Queue Count = " + test.GetQueueCount().ToString());
      }
```

```
        }
    }
}
```

والخرج يكون كما يلي:



الصنف ThreadPool

هو عبارة عن حوض من النياسب التي نقوم بإنشائها مسبقاً من أجل إعادة إستخدامها لاحقاً، أي يمكننا أن نضع هذه النياسب و التي تُمثل مهام يجب تحقيقها ضمن رتل Queue يحوي النياسب التي نريد تنفيذها.

بالإضافة إلى أن هذا الحوض يقوم بإدارة النياسب عند حدوث طلب I/O يحدث البديل بين النياسب بشكل غير متزامن بالإضافة إلى آلية (Wait/Notify).

يُستخدم عادةً حوض النياسب (ThreadPool) من قبل مخدمات الويب (Web Servers) ومخدمات قواعد المعطيات (Data Base Servers)، حيث يقوم كل منهما بإنشاء عدد من النياسب وتضعها ضمن حوض من أجل إعادة استخدامها لاحقاً.

في الحقيقة ليس من المجدي أن يقوم مخدم الويب (Web Server) أو أي مخدم (Server) بإنشاء نيسب لكل زبون يتصل به لأن هذه الطريقة لها العديد من المشاكل :

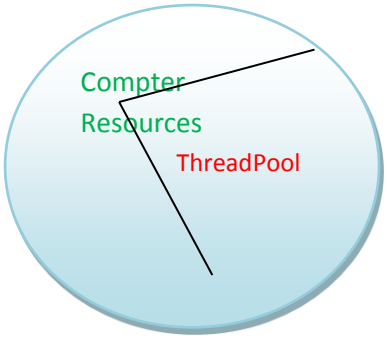
1- استهلاك عالي جداً للموارد (Ram + CPU Time).

2- إقلاع النيسب يأخذ وقت طويل نسبياً(تقريباً من 100 إلى 200 ميلي ثانية) بينما تبديل السياق (المكدس +مجموعة السجلات) بين النياسب يأخذ وقت قصير جداً.

3- يمكن لهذا المخدم أن يتعرض لهجمة عن طريق إرسال مليون طلب إلى مخدم معين من أجل تعطيله (من أجل إستنفاذ موارده)، لهذا السبب تطلب المواقع عند تسجيل شخص ضمنها أن يدخل أرقام موجودة ضمن صورة (لكي يتأكد من أن المتصل هو شخص وليس حاسب).

لذلك تخصص هذه الطريقة (ThreadPool) جزء من موارد الحاسب ليتم إعادة استخدامها للأسباب السابقة ولأن أقلاع النيسب يأخد وقت طويل نسبياً كما ذكرنا سابقا.

الشكل التالي يوضح الغرض من استخدام حوض موارد.



الجدير بالذكر أن النياسب التي نضعها ضمن هذا الحوض (ThreadPool) هي نياسب تعمل في الخلفية (Background)، أي أن النياسب ضمن هذا الحوض لا تموت إلا عندما يموت أخر نيسب أمامي (هو ال Main Thread في الحالة العامة).

لنرى الكود التالي، أرجو قراءة التعليقات فهي هامة.

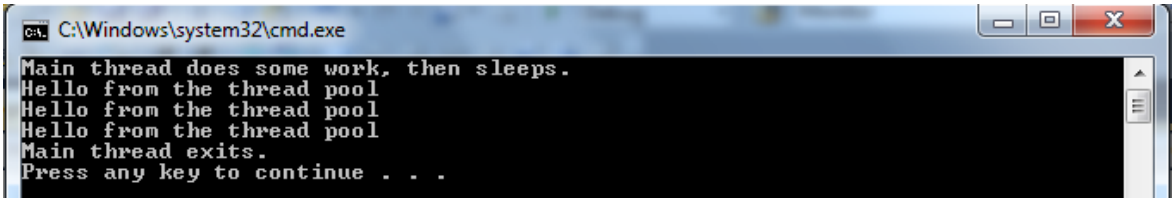
```
public static void Main()
{
    for (int i = 0; i <3; i++)
    {

        // Queue the task(Method)  for execution in the form of Background Thread
       ThreadPool.QueueUserWorkItem(new WaitCallback(ThreadProc),null);

    }

    Console.WriteLine("Main thread does some work, then sleeps.");
    // If you comment out the Sleep, the main thread exits before
    // the thread pool task runs.  The thread pool uses background
    // threads, which do not keep the application running.  (This
    // is a simple example of a race condition.)

    Thread.Sleep(1000);

   Console.WriteLine("Main thread exits.");
}


// This thread procedure performs the task.
static void ThreadProc(Object stateInfo)
{
    // No state object was passed to QueueUserWorkItem, so
    // stateInfo is null.

    Console.WriteLine("Hello from the thread pool.");
}
```

شرح الكود:

يقوم هذا البرنامج بإنشاء حوض موارد عن طريق الصنف ThreadPool وقمنا بإضافة ثلاثة نياسب إلى الحوض (والذي يُمثل رتل إنتظار يحوي مجموعة من النياسب التي تعمل في الخلفية)، كما يمكن تمرير بيانات إلى النيسب ليستعملها في عمله ولكننا لم نقم بذلك (قمنا بتمرير null).

تنتهي جميع النياسب ضمن حوض النياسب عندما يموت أخر نيسب أمامي وفي مثالنا هذا هو ال Main Thread أي عندما يموت هذا النيسب تموت كل النياسب التي تعمل ضمن الحوض (لأنها نياسب تعمل في الخلفية).

لنرى الخرج الآن.



الفصل التاسع

إدارة الذاكرة والتعامل

مع المؤشرات

Memory management

and Pointers

مقدمة

تُعتبر إدارة الذاكرة من أهم المواضيع التي على المبرمج المحترف أن يتقنها، مع أنَّ NET. تؤمن لنا ما يُسمى بالشيفرة المُدارة (Managed Code) وهذه الشيفرة تحظر علينا التعامل بشكل مباشر مع الذاكرة من خلال ما يُعرف بالمرجع (Reference)، إلا أننا وفي بعض الأحيان قد نضطر إلى التعامل بشكل مباشر مع الذاكرة وبالتالي ينبغي علينا فهم المبادئ الأساسية المُستخدمة لكيفية حجز المساحات ضمن المنطقة Heap (تُسمى الكومة) وضمن المكدس (Stack) .

وتُعتبر لغتي C/C++ هي من أهم اللغات التي تتعامل مع المؤشرات بشكل صريح، أما في لغة C# فإستخدام المؤشرات غير محبذ خصيصا للأشخاص غير المتمرسين بالتعامل مع المؤشرات لأن التعامل معها صعب وإدارتها أصعب بكثير هذا بالإضافة إلى أن إيجاد الأخطاء في المؤشرات هو موضوع صعب قليلاً وبحاجة إلى معرفة جيدة بطريقة الحجز الذاكري من قبل المترجم ونظام التشغيل. سنتكلم أيضاً عن طريقة عمل جامع النفايات (Garbage Collector) وكيف يمكننا أن نقوم بتحرير الموارد الغير مُدارة (Unmanaged Resources) مثل الملفات وقواعد المعطيات، وذلك عن طريق الواجهة System.IDisposable و من خلال استخدام الهادم (Destructor). كما سنلقي نظره عن كيفية كتابة شيفرة غير آمنة (استخدام المؤشرات) ضمن اللغة C#.

إدارة الذاكرة تحت المجهر

أحد أهم مزايا اللغات التي تعمل تحت منصة NET. مثل C# أنها لا تُتعب المبرمج في تفاصيل إدارة الذاكرة وكيفية حجز الأغراض وتحريرها، وعلى وجه الخصوص جامع النفايات (Collector Garbage) والذي يوفرعلى المبرمج عناء تحرير المناطق المحجوزة من قِبَل البرنامج في الذاكرة، وبالنتيجة الحصول على لغة تملك من الفعالية ما يؤهلها لمنافسة لغة C++ من دون الخوض في تفاصيل إدارة الذاكرة. لكن إذا إردنا ان نكتب شيفرة فعالة وسريعة فإنه يجب علينا أن يكون لدينا نظرة عامة عن طريقة حجز الذاكرة ضمن الحاسب وهذا ما سنتكلم عنه في الفقرة التالية.

Reference Data Type Vs Value Data Type

1. Value Data Type:

يتم تخزين هذه المتحولات ضمن المكدس (Stack) ومن هذه المتحولات: المتحولات المحلية (Local Variable) بالإضافة إلى بارمترات الإستدعاءات للتوابع المختلفة.

في الحقيقة لكي يتوضح لنا كيف يعمل المكدس (Stack) وهو بنية معطيات تعتمد على القاعدة التالية (LIFO) أي (Last In First Out). لنرى المثال التالي والذي يُبين لنا طريقة حجز المتحولات من قبل مترجم اللغة (Compiler) والذي يستخدم توابع خاصة بنظام التشغيل لإدارة الذاكرة.

```
{
    int intval = 10;

    double doubleval = 300.0;
}
```

في البداية يجب أن نعلم أن نظام التشغيل يقوم بتخصيص مقطع خاص للمكدس لكل برنامج قيد التنفيذ (process)، يحوي هذا المكدس كافة أنماط القيمة (Value Data Type) التي تحدثنا عنها قبل قليل، و كما نعلم من لغة C++ وتحت عنوان "مجال الرؤية" (Scope Resolution) للمتحولات فإن المتحول intval يتم التصريح عنه في البداية فيقوم المترجم بحجز أربعة خانات (بايتات) في الذاكرة لتخزين المتحول الذي من نمط int (أربعة بايتات) وبالتالي يتم حجز أربعة عناوين متلاصقة لتخزين قيمة المتحول intval. لنرى الشكل التالي والذي يوضح لنا الفكرة:

عند تشغيل البرنامج فأن مؤشر المكدس يؤشر على قيمة ما في الذاكرة ولتكن 800000 وبالتالي عندما يقوم المترجم بحجز المتحول intval ويضع ضمنه قيمة 10 فإن هذه القيمة يتم تخزينها ضمن الخانات ذات العناوين من 800000 و حتى القيمة 799997 ويصبح مؤشر المكدس عند القيمة 799996 (أي أننا قمنا بطرح القيمة 4 من قيمة مؤشر المكدس)، وعندما نقوم بحجز المتحول الثاني وهو doubleval من النمط double (يتم تمثيله على 8 بايت) ونضع ضمنه القيمة 3000.0 و بالتالي نقوم بإنقاص قيمة مؤشر المكدس الحالية (799996) بمقدار 8 بايت فتصبح قيمة مؤشر المكدس الجديدة هي 799988 ويتم تخزين قيمة المتحول 3000.0 ضمن الخانات التي عناوينها من 799996 إلى 799989.

والآن يتم الخروج من ال Scope ({}) ويجب تحرير الذاكرة التي قمنا باستخدامها لحجز المتحولات. كل ما يقوم به المترجم هو زيادة قيمة مؤشر المكدس الحالي (799988) بمقدار 8 بايتات (لتحرير متحول من النمط double) لكي نعود إلى القيمة 799996، ومن ثم زيادة قيمة مؤشر المكدس الحالية(799996) بمقدار 4 بايتات (لأننا نقوم بتحرير متحول int) ليعود مؤشر المكدس إلى القيمة التي كان عليها من قبل أن نقوم بتعريف المتحولين السابقين ضمن الكتلة ({}) أي القيمة (800000)، بعد ذلك يقوم المترجم بحجز خانات لبقية المتحولات الموجودة في البرنامج ويتم كتابة قيمها فوق القيم السابقة في المكدس.

نلاحظ أن عملية حجز المتحولات والأغراض ضمن المكدس هي عملية سريعة جداً، ذلك بسبب طريقة الحجز البسيطة والتي تعتمد على زيادة وإنقاص مؤشر المكدس[1].

[1] للمزيد من المعلومات يمكن الإطلاع على الرابط التالي :
/http://www.programmerinterview.com/index.php/data-structures/difference-between-stack-and-heap

ملاحظة: مؤشر المكدس (Stack Pointer) يتم حفظه في مكان ما ضمن نظام التشغيل وهو يؤشر على قمة المكدس حيث يتم الحجز من الأعلى إلى الأسفل أما تحرير الذاكرة فيتم من الأسفل إلى الأعلى.

2. Reference Data Type :

في الحقيقة إن المكدس يعطينا وصول سريع جداً للمتحولات، لكن المكدس هو عبارة عن منطقة صغيرة مقارنةً بالكومة (Heap).
إنّ هذا النوع من الأنماط هو عبارة عن كل غرض يتم جحزه بالذاكرة أي "Class is Reference Type"، وبالتالي أي غرض يتم حجزه عن طريق التعليمة New يُعامل معاملة ال Reference Type، حيث يتم تخزين المرجع (Reference) في المكدس وهذا المرجع هو عبارة عن عنوان الغرض ضمن الذاكرة Heap.

مثال :

```
{
        List<int> Arr = new List<int>();
}
```

Arr: هو Reference (موجود في المكدس) يوشر على غرض موجود في الذاكرة Heap.

ملاحظات هامة :

1- الفرق بين المؤشر (Pointer) و المرجع (Reference) هو أن الأول يقبل العمليات الحسابية مثل الجمع والطرح أما الثاني فلا يقبل ذلك (خشية الخروج إلى خارج الذاكرة والدخول في مشاكل نحن بغنىً عنها).

2- تلكمنا قبل قليل أن مقطع المكدس (Stack) المخصص للبرنامج قيد التنفيذ (والذي يُسمى Process) هو أصغر من مقطع الكومة (Heap) المخصص لهذه ال Process لأن المكدس هو منطقة خاصة بكل نيسب (Thread) أما المنطقة Heap فهي منطقة مشتركة لكل النياسب[1].

3- أن الحجز الذاكري ضمن المنطقة Heap هي أبطئ من الحجز ضمن المنطقة Stack هذا ما سنراه في الفقرة التالية .

4- عملية الحجز ضمن المنطقة Heap تتم من الأسفل إلى الأعلى(على عكس المكدس) بينما يكون تحرير الذاكرة من الأعلى إلى الأسفل.

[1] للمزيد من المعلومات يمكن الإطلاع على الرابط التالي : https://en.wikipedia.org/wiki/Stack-based_memory_allocation

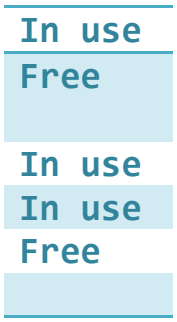
جامع النفايات Garbage Collector

لنرى المثال التالي :

```
{
    List<int> Arr = new List<int>();
}
```

في الحقيقة عند الخروج من ال Scope يموت ال Reference والذي هو Arr (يقوم المترجم بتحرير الذاكرة المستخدمة لتخزين Arr)، ولكن الغرض الحقيقي موجود في الكومة (Heap) وبالتالي يصبح هذا الغرض من دون أي مؤشر يؤشر عليه وعندها يقوم جامع النفايات (GC) بإزالة هذا الغرض من ال Heap.

يعمل جامع النفايات عندما تصبح الذاكرة Heap شبه ممتلئة فتقوم بحذف كل الأغراض التي ليس لها Reference يؤشر عليها في المكدس، والآن تصبح حالة الذاكرة Heap كما في الشكل التالي حيث الأغراض المحجوزة مبعثرة.



لو تركنا المنطقة Heap كما في الشكل السابق فأن عملية حجز الأغراض ضمنها ستكون عملية مربكة خلال زمن التنفيذ، خصوصاً أنه يوجد مساحات ضائعة لا يكن الإستفادة منها لأننا نقوم بالبحث عن منطقة من الذاكرة متلاصقة لتخزين غرض معين.ولهذا السبب يقوم GC بعملية ضغط (Compact) للمنطقة Heap (مشابهة لعملية ضغط القرص الصلب تماماً)، وباالتالي تُصبح المنطقة Heap تحوي كل الأغراض المحجوزة بشكل متسلسل ومتلاصق من دون أي مساحات ضائعة، تُسمى عندها المنطقة Heap ب "Managed Heap"[1] أي أنها مُدارة من قبل GC.

الجدير بالذكرأيضاً أنه بعد عملية الضغط هذه تُصبح عملية حجز الأغراض ضمن المنطقة Heap تتم بشكل سريع (يصبح العمل مع المنطقة Heap بشكل مشابه للمنطقة Stack من حيث السرعة)، وهذا ما يُميز " Managed Heap" عن ال Heap العادية.

كما ذكرنا سابقاً إن ال GC تعمل عندما تُصبح المنطقة Heap بحاجة إلى عملية تنظيف (Release) للأغراض التي ليس لها أي أي مرجع (Reference) يؤشر عليها، كما يمكننا أن نشغل ال GC بشكل قسري عن طريق التعليمة :

```
System.GC.Collect();
```

في الحقيقة لا يمكن أن نضمن عمل ال GC باللحظة التي نريدها.

[1] للمزيد يمكن الإطلاع على الرابط التالي: http://msdn.microsoft.com/en-us/library/f144e03t(v=vs.110).aspx

تحرير الموارد غير المُدارة من قبل .NET.

إن وجود جامع النفايات (GC) يعني أن المبرمج ليس بحاجة إرهاق نفسه في التخلص من الأغراض لذلك يترك هذه المهمة ل GC، لكن GC لا تعرف كيف تقوم بتحرير الموارد الغير مُدارة (مثل الملفات المفتوحة أو الإتصالات الشبكية، أو حتى الإتصالات مع قواعد البيانات).

عندما يكون لدينا صف معين (Class) يحوي مرجع إلى غرض غير مُدار، لدينا طريقتان لجعل الأغراض التي تتعامل مع موارد غير مُدارة تقوم بعملية تحرير لهذه الموارد قبل أن يتم هدمها من قِبَل GC :

1-استخدام تابع هادم ضمن الصنف

كما نعلم عند إنشاء الغرض يتم إستدعاء الباني (Constructor) وكذلك عند هدم الغرض يتم إستدعاء الهادم (Destructor)، إذاً الهادم هو المكان المناسب للقيام بعملية التنظيف قبل هدم الغرض من قبل GC.

يتم تعريف الهادم كما يلي (بالشكل التالي حصراً):

```
class Myclass
{
    ~Myclass()
    {
 //Implementation for free Unmanaged Resources used Directly by an Object: Examples :
        // Closing  connection To DB
        // Closing  Opened Files
        // Closing  Networking Connections
    }
}
```

قلنا قبل قليل أن الهادم يتم إستدعاءه عند هدم الغرض فوراً هذا الكلام صحيح في لغة C++، أما في لغة C# وبما أننا نستخدم GC لهدم الأغراض فأنه لا يوجد طريقة لمعرفة متى يتم استدعاء الهادم الخاص بالغرض الذي نريد تحريره عن طريق GC.

مشكلة أخرى هي أن تنفيذ كود الهادم يبطئ من عملية تحرير الذاكرة. وبالتالي فأن الأغراض التي ليس لها هادم (Destructor) تُحرر من الذاكرة بمرور واحد لل GC، أما الأغراض التي تحوي هادم فإنها تحتاج إلى مرورين لل GC ليتم هدمها، الأول عندما تقوم ال GC باستدعاء الهادم لكل الأغراض التي نريد هدمها (حذفها من الذاكرة)، والمرور الثاني لل GC هو من أجل عملية حذف الأغراض من الذاكرة بشكل فعلي.

كما أنه ضمن زمن التنفيذ يُستخدم Thread واحد لتنفيذ الهادم لكل الأغراض التي يريد GC أن يحررها من الذاكرة، وإذا كان الهادم يحوي مهام تنظيف تحتاج إلى وقت طويل نسبياً فإن الأداء ينخفض بشكل ملحوظ.

2-استخدام الواجهة IDisposable

الطريقة الموصى بها من قبل مايكروسوفت لتحرير الموارد غير المُدارة، لتجنب المشاكل المتعلقة باستخدام GC تزودنا هذه الواجهة بمنهج (Method) وحيد اسمه ()Dispose ويكون التحقيق بالشكل التالي:

```
class Myclass :IDisposable
  {

      public void Dispose()
      {

       //Implementation for free Unmanaged Resources used Directly by an Object: Examples :
          // Closing  connection To DB
          // Closing  Opened Files
          // Closing  Networking Connections
      }
  }
```

إن استخدام المنهج Dispose يزودنا بتحكم دقيق عندما نقوم بتحرير الموارد الغير مُدارة.

لنرى مقطع الشيفرة التالي :

```
Myclass a = new Myclass();

//Do Processing

a.Dispose();
```

نقوم في هذه الشيفرة بإنشاء غرض من الصف Myclass والذي يستخدم موارد خارجية غير مُدارة (Unmanaged Resources)، ثم قمنا بمجموعة من العمليات على هذا الغرض ومن ثم قمنا بتحرير الموارد التي استخدمها هذا الغرض عن طريق استدعاء المنهج Dispose. لكن ولسوء الحظ هذه الشيفرة سوف تفشل في تحرير الموارد الغير مُدارة في حال حدوث إستثناء (Exception) لذلك يجب أن نضع هذه التعليمات ضمن كتله Try أما تحرير الموارد فيتم ضمن الكتلة Finally وذلك كما يلي:

```
Myclass a=null;
try
{
     a= new Myclass();

    //Do Processing
}
finally
{
    if(a!=null)
        a.Dispose();
}
```

في هذا الكود نضمن استدعاء المنهج Dispose جتى عندما يحصل استثناء، ولكن إستدعاء المنهج Dispose من أجل كل غرض قد يكون أمراً مربكاً على المبرمج لهذا السبب تؤمن لنا C# التعليمة using والتي تعرف لنا كتلة من الكود (Scope) أي {} والتي تضمن لنا استدعاء تلقائي ومضمون للمنهج Dispose (طبعاً لغرض يحقق الواجهة IDisposable). وبالتالي يصبح الكود كما يلي:

```
using (Myclass a=new Myclass())
 {
     //Do Processing
 }
```

في الحقيقة إنّ هذه الشيفرة تكافئ الشيفرة السابقة تماماً لها، حيث أننا عند الخروج من ال Scope الخاص بالتعليمة Using فإن المنهج Dispose سيتم استدعاءة تلقائياً من أجل الغرض الذي يؤشر عليه المرجع a، حتى عند حدوث استثناء (Exception) فإن المنهج Dispose سيتم استدعاءه تلقائياً.

3-إستخدام الطريقتين الهادم والواجهة IDisposable

قبل قليل ناقشنا كل حل من الحلين على حدا كطريقة لتحرير الموارد الغير مُدارة من قبل غرض معين وكانت :

1- تنفيذ الهادم (Destructor) بشكل قسري ضمن زمن التنفيذ ولكن هذه الطريقة غير حتمية (أي لا يمكننا معرفة متى يتم استدعاء الهادم الخاص بالغرض الذي نريد تحريره عن طريق GC)، بالإضافة إلى أن لها تأثير غير مقبول على الأداء بسبب طبيعة عمل ال GC الغير حتمية، باختصار هي عبارة عن آلية أمنه ولكنها بطيئة وغير حتمية.

2- تحقيق الواجهة IDisposable والذي يزودنا بألية حتمية لتحرير الموارد ولكنها تتطلب استدعاء صحيح للمنهج Dispose، باختصار هي أفضل من ناحية الأداء ولكنها تتطلب مجهود إضافي من خلال إستدعاء المنهج Dispose.

الطريقة الأفضل هي تحقيق كلا الآليتين معاً، في المثال التالي سنقوم بتحقيق الواجهة IDisposable وذلك بفرض أن معظم المبرمجين سوف يقومون باستدعاء المنهج Dispose بشكل صحيح، ولكن سنقوم بتحقيق الهادم (Destructor) كآلية آمنة توفر على المبرمج عناء استدعاء المنهج Dispose. لنرى المثال التالي والذي يتكلم عن تحقيق كلا الآليتين :

```
class Myclass :IDisposable
{

    private bool isDisposed = false;

    public void Dispose()
    {

     //Implementation for free Unmanaged Resources used Directly by an Object: Examples :

        Console.WriteLine("Disposing");
```

```
            Dispose(true); //Cleaning  managed and unmanaged Resources

    //Requests that the system does not call the finalizer(Destructor) for the specified object
            GC.SuppressFinalize(this);
        }


        protected virtual void Dispose(bool disposing)
        {
            if (!isDisposed)
            {
                if (disposing)
                {
        //Cleaning Magnaged objects by Calling Their Dispose() Methods ( Release Then Run GC)
                }

                //Cleaning Unmagnaged Resources
            }

            isDisposed = true;
        }

        ~Myclass()
        {
            Dispose(false);//Cleaning Unmanaged Resources

            Console.WriteLine("Destructor");
        }

    }
```

في الحقيقة إن هذا الكود يصلح لنوعين من المبرمجين:

الأول يحب أن يقوم باستدعاء المنهج Dispose() لذلك يقوم بكتابة الكود التالي:

```
            Myclass a=null;
            try
            {
                 a= new Myclass();

                //Do Processing
            }
            finally
            {
                if (a != null)
                    a.Dispose();
            }
```

شرح الكود:

أولاً يمكن للمبرمج أن يستبدل هذا الكود بتعلمية using والتي تؤمن استدعاء تلقائي للمنهج ()Dispose عند الخروج من الكتلة using(Scope). لكن لو نظرنا إلى المنهج ()Dispose الخاص بالصف Myclass لوجدنا أنه يحوي طباعة جملة ثم استدعاء للمنهج ;Dispose(true) في هذه الحالة تكون قيمة المتحول isDisposed=False وبما أننا قمنا بتمرير قيمة true فإن هذا المنهج Dispose(true) سيقوم بتحرير الموارد المُدارة وغير المُدارة ومن ثم يقوم بتغيير قيمة المتحول isDisposed إلى true ثم ضمن المنهج ()Dispose نقوم بإرسال طلب إلى GC لكي لا تقوم باستدعاء الهادم (لأننا قمنا بعملية التحرير) وذلك عن طريق التعليمة:

```
GC.SuppressFinalize(this);
```

حيث تدل هذه التعليمة أن ال GC سيتعامل مع هذا الغرض كما لو أنه ليس له تابع هادم (Destructor).

أما المبرمج الثاني، فإنه يكتب:

```
{
    Myclass a = new Myclass();
    //Do Processing
}
```

نلاحظ أن هذا المبرمج لم يقم باستدعاء المنهج ()Dispose وبالتالي عند هدم الغرض تقوم ال GC باستدعاء المنهج ;Dispose(false) والذي يقوم بتحرير الموارد غير المدارة فقط أما الموارد المُدارة (الذاكرة المحجوزة لهذا للأغراض الموجودة ضمن هذا الغرض) فإن ال GC سيقوم بهدمها.

الشيفرة غير الآمنة Unsafe Code

إلى الآن وجدنا أن لغة #C هي لغة ممتازة في إخفاء مواضيع إدارة الذاكرة عن المطور، و مع وجود ال GC والتعامل مع ال References أصبح التعامل مع الذاكرة أكثر راحة للمطور، لكن هناك بعض الحالات التي يتوجب علينا أن نقوم باستخدام المؤشرات بشكل صريح (وصول مباشر إلى الذاكرة)، مثلاً:نريد الوصول إلى تابع موجود في ملف DLL غير مُدار(Unmanaged DLL) مكتوب بلغة C/C++ وهذا الوصول يتطلب تمرير مؤشر كبارمتر لهذا التابع (كما هو الحال في معظم توابع ال Windows API)، أو يمكن إستخدام المؤشرات لأسباب تتعلق بتحسين الأداء. في هذا القسم سنتكلم عن التسهيلات التي تقدمها لغة #C من أجل الوصول بشكل مباشر إلى الذاكرة.

المؤشرات Pointers

على الرغم من أننا سنقوم بشرح المؤشرات وكأنها موضوع جديد إلا أنه في الحقيقة ليس بجديد علينا.

تعاملنا مع المراجع (References) بشكل مريح ضمن الشيفرة ولكن ما الفرق بين المؤشر (Pointer) وال Reference؟ في الحقيقة إن ال References هو مؤشر من النوع الآمن (References is type safe Pointer) وبكلمة أخرى هو "Reference is Read only Pointer ". أما المؤشر فهو عبارة عن متحول يحوي عنوان شيئ ما (قد يؤشر على قيمة صحيحة وبالتالي فإنه يؤشر على منطقة من الذاكرة بحجم 4 بايت مثلاً وقد يؤشر على غرض)، يمكننا أيضاً أن نقوم بإضافة 4 بايت إلى قيمة المؤشر. وبالتالي لدينا من القوة ما يكفي للتحم بالذاكرة بشكل كامل.

يوجد سببين رئيسيين لإستخدم المؤشرات:

1-التوافقية مع التقنيات القديمة (مثل التوافقية بين .NET و COM) وهذا ما يُعرف بأسم" backwards compatible"، على الرغم من كل التسهيلات التي تقدمها .NET. إلا أنها ما تزال تستخدم توابع المكتبات الخاصة بويندوز (Windows API) والتي تحتاج غالباً إلى تمرير مؤشرات كبارمترات. أي أنه لا يمكننا التخلي عن المؤشرات.

2- تحسين الأداء(performance): في العديد من الحالات التي يكون فيها السرعة هو العامل الأهم، نستطيع باستخدام المؤشرات أن نحصل على ما نريد من خلال الاستخدام الأمثل للذاكرة المحجوزة ولكن هذا الأمر يحتاج إلى خبرة كبيرة.

الوصول المباشر إلى الذاكرة يحتاج إلى عدة أمور منها :

1- استخدام المؤشرات أصعب بكثير من التعامل مع ال References.

2- كما أن استخدام المؤشرات يتطلب مهارات برمجية ممتازة لكي لا نقوم بتخريب معطيات ضمن الذاكرة أو الوصول إلى منطقة غير مسموح الوصول إليها وكل هذا ينتج عنه مشاكل كبيرة قد نكون بغنى عنها. على الرغم من كل هذه المشاكل تبقى المؤشرات أداة قوية جداً ومرنة مرونة كافية لكتابة تطبيقات فعالة.

كتابة شيفرة غير آمنة Writing Unsafe Code

يمكننا كتابة مقطع أو كتلة من الشيفرة الغير آمنة (مخصصة لإستخدام المؤشرات). فيما يلي نعرض المنهج GetNumber والذي يستخدم المؤشرات.

```
unsafe int GetNumber()
{

    //Code That can use Pointer

    int x = 10;

    int* px = &x;

    return 1;
}
```

تقوم الكلمة unsafe والتي تخبر المترجم بأن هذا المنهج يستخدم مؤشرات (كما يمكن أن تكون البارمترات هي عبارة عن مؤشرات أيضاً). بنفس الطريقة يمكننا استخدام هذه الكلمة مع الصفوف Classes ويمكن استخدمها مع الـ Struct.

```
unsafe class Myclass
 {
     int* px ; //Declaration of  a Pointer filed in a Class

    // Any Method in This Class Can now use Pointers
 }
```

كما يمُكننا كتابة كتلة من الشيفرة غير الآمنة ضمن منهج عادي. مثلاً:

```
 static void Main(string[] args)

 {

     Unsafe int* px; // Wrong and Generate Compiler Error

            unsafe
            {
                //Unsafe Code That uses Pointers Here
            }
           //More safe Code That deose't use Pointers
  }
```

ملاحظة:

لكي نستطيع ترجمة شيفرة غيرأمنة ضمن ال Visual studio.NET يجب علينا أن نُغير خصائص المشروع، فنقوم بتفعيل خيار Allow unsafe Code، إذا كنا نستخدم موجه الأوامر يُمكننا أن نكتب:

```
csc –unsafe myprogram.cs
```

نفترض أنه لديك معرفة مسبقة عن المؤشرات بلغة C++ ولكننا سنذكر ببعض المفاهيم.

الشكل العام لتعريف المؤشرات:

```
Typename* variablename;
```

أمثلة:

```
int x=10;

int* px=&x;

double* [] arr;
```

لنرى مقطع الشيفرة التالي:

```
unsafe
{
          //Unsafe Code That uses Pointers Here

         int x = 10;

         int* px = &x;

   Console.WriteLine("x={0}  &x={1}  px={2}  *px={3} ", (int)x, (int)&x, (int)px, (int)*px);
}
```

شرح الكود:

في البداية علينا أن نعلم ما هو الفرق بين المتحول والمؤشر؟ المتحول هو عبارة عن خانة في الذاكرة لها عنوان محدد وتحوي قيمة، مثل المتحول int x = 10; الذي يحوي القيمة 10 (أما الوصول عن طريق الاسم فإن هذه هي مهمة المترجم والذي يؤمن تقابل بين اسم المتحول وعنوانه ضمن الذاكرة)، أما المؤشر فإنه عبارة عن خانه في الذاكرة لها عنوان محدد وتحوي عنوان خانة أخرى، مثلاً : int* px = &x; المؤشر Px هو عبارة عن متحول في الذاكرة يحوي عنوان المتحول x حيث تدل لعملية & على عنوان المتحول وليس قيمته.

وكما نعلم أيضاً أن العملية * ضمن تعليمة الطباعة تدل على أننا نريد طباعة محتوى الموقع الذي يشير عليه هذا المؤشر (px) وذلك عن طريق التعليمة px* . (أي أن العمليتين * و & هما متعاكستان في الفعل).

لذلك نرى الخرج كما يلي:

في البداية قمنا بطباعة المتحول x ثم قمنا بطباعة عنوان المتحول x في الذاكرة وذلك عن طريق التعليمة &x، ثم قمنا بطباعة قيمة المتحول px والذي يمثل مؤشر فنجد أنه قيمته هي عنوان المتحول x، ومن ثم قمنا بطباعة محتوى الموقع الذي يشير عليه المؤشر px عن طريق التعليمة *px فنجد أن قيمته تساوي إلى قيمة x.

يُمكننا استخدام المؤشرات في C# بنفس الطريقة التي نستخدمها في C++ تقريباً (مع إختلافات بسيطة جداً).

الخاتمة

إلى هنا نكون قد وصلنا إلى نهاية جولتنا في عالم البرمجة ضمن بيئة NET.، تم التركيز على مفاهيم وتقنيات مشتركة في معظم لغات البرمجة الموجودة.

أرجو من الله أن يكون هذا الكتاب مفيداً لكل من يحتاجه، وأن يساهم هذا الكتاب في زيادة المعرفة بالتقنيات البرمجية المتقدمة باللغة العربية بشكل أصيل ومتين، سأكون مسروراً حقاً بملاحظاتكم على هذا الكتاب وأرجو ألا تبخلو بها. أرجو لكم المُتعة والفائدة، و إلى اللقاء في إصدارات أحدث للكتاب تحوي مواضيع إضافية أعمق وأحدث تواكب التطورات التي يشهدها العصر.

المراجع

- كتاب Professional_CSharp_2008_wrox

- كتاب Professional C# 4 and .NET4 worx

- كتاب Mastering C Sharp Database Programming

- موقع شبكة مطوري مايكروسوفت MSDN- Microsft_Developer Network