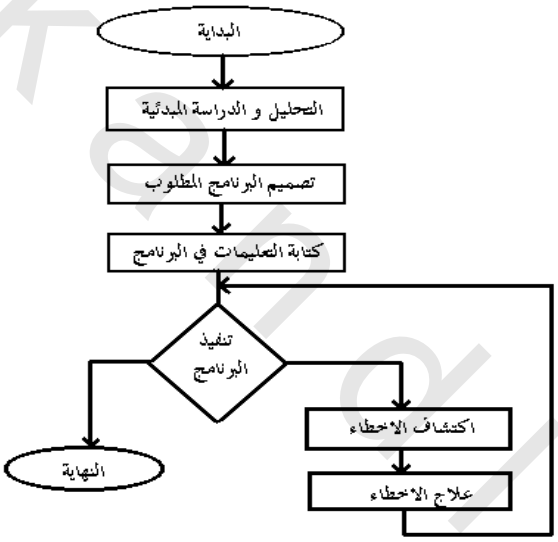
الفصل الخامس
تعقب وتصحيح الأخطاء
Debugging

تسمي أخطاء البرنامج **Bugs** ويطلق علي عمليـــة تنقيـــة البرنامج من هذه الأخطاء **Debugging** وتعني تعقب الأخطـــاء وتصحيحها. سنشرح في هذا الفصل أنواع الأخطاء التي يمكـــن أن تقع فيها أثناء كتابة وإعداد البرنامج وكيفية تجنبها وسنشرح الطرق المختلفة لتعقب هذه الأخطاء واكتشافها حتى يصبح البرنامج خاليا منها وبالتالي يمكن تنفيذه والحصول علي النتائج المرجوة.

بانتهاء هذا الفصل ستتعرف علي:

- الأخطاء الهجائية
- الأخطاء أثناء التنفيذ
- الأخطاء المنطقية
- إصدارات المشروع
- إجراء عملية التصحيح
- تعيين نقاط التوقف **Breakpoints**
- استخدام نوافذ المصحح

عند الشروع فى إعداد برنامج معين لحل مشكلة معينة يتم التعرف أولاً على نوع المشكلة والقيام بعمل الدراسة المبدئية للمشكلة ثم تصميم البرنامج الذى يقوم بحل هذه المشكلة ثم كتابة تعليمات البرنامج واختباره وفى حالة وجود أخطاء فلابد من معالجتها. وكما ترى فى شكل ٥-١ أن اكتشاف الأخطاء وعلاجها داخل البرنامج يعتبر عنصراً هاماً فى دورة إعداد أى برنامج وهى خطوة مطلوبة للوصول إلى برنامج ناجح. وذلك لأن أى مبرمج مهما بلغت خبرته قد يقع فى خطأ أثناء كتابة البرنامج ولذلك فإن عليه أن يتعرف على إمكانيات اكتشاف الأخطاء المختلفة والطرق السريعة للعلاج لتوفير الوقت والجهد.



شكل ٥-١ اكتشاف الأخطاء وتصحيحها عنصر هام في مرحلة تنفيذ البرنامج

وبما أن البرنامج سيحتوي على أخطاء في كل الأحوال، فيصبح من الضروري العثور على هذه الأخطاء والتي تدعى **Bugs**، ومن ثم تصحيحها. يمدك **Visual Basic** بكثير من الأدوات التي تساعدك في ذلك إذ يشتمل علي بنية تنقيح كاملة، تحتوي هذه البنية على:

- مراجعة هجائية، لتتأكد من أنك كتبت الأوامر بصورة صحيحة.
- رؤية آنية للمتغيرات **Watching Variables**، لترى قيم المتغيرات أثناء تشغيل البرنامج.

- تعقب الكود **Code Tracing** يجعلك تعرف السطر الذي ينفذه المترجم وتأثير تنفيذه.
- سرد للإجراءات قيد النداء **Procedure Call Listing** ، لتعرف من أي الإجراءات تم نداء الدوال أو الإجراءات الحالية.

وقبل أن نتعرض بالشرح لطرق تتبع الأخطاء وتصحيحها، سنتعرف أولا على أنواع الأخطاء التي يمكن أن يقع فيها المبرمج وكيفية تلافيها.

أنواع الأخطاء

يمكن تقسيم الأخطاء التى يقع فيها المبرمج إلى ثلاثة أنواع كالآتى:

- أخطاء هجائية **Syntax Errors**
- أخطاء أثناء التنفيذ **Run-Time Errors**
- أخطاء منطقية **Logical Errors**

أولا: الأخطاء الهجائية

يطلق عليها **Syntax Error** وتنتج هذه الأخطاء عند كتابة تعليمات لا تُراعى فيها قواعد اللغة المستخدمة للبرمجة عموماً ويقوم محرر **Visual Basic** (نافذة الكود) بالتعرف على بعض الأخطاء من هذا النوع بمجرد كتابة التعليمات، بينما يتم التعرف على باقى هذه الأخطاء فى مرحلة التنفيذ. ومن أمثلة هذا النوع من الأخطاء .. الخطأ فى كتابة أمر داخل البرنامج.

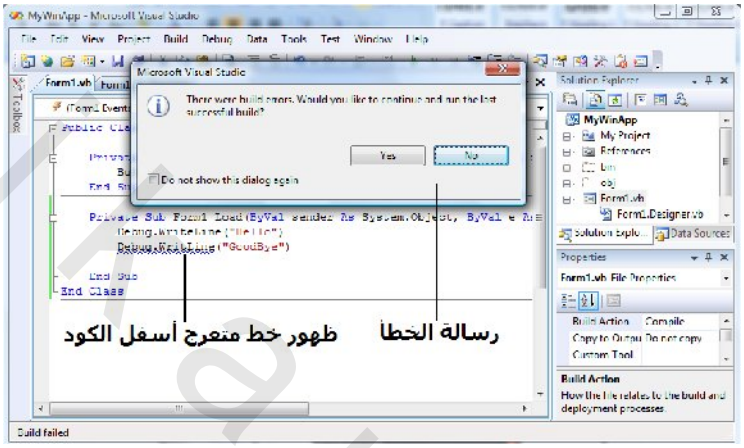
مثال:

اتبع الخطوات التالية لطباعة العبارتين **"Hello"** و**"Goodbye"** على شاشة الحاسب:

١. قم بإنشاء مشروع نوافذى جديد أو افتح مشروع موجود مسبقاً.
٢. انقر نموذج المشروع نقراً مزدوجاً. يفتح إجراء جديد باسم **Form1_load**.
٣. اكتب الأمران التاليان داخل الإجراء

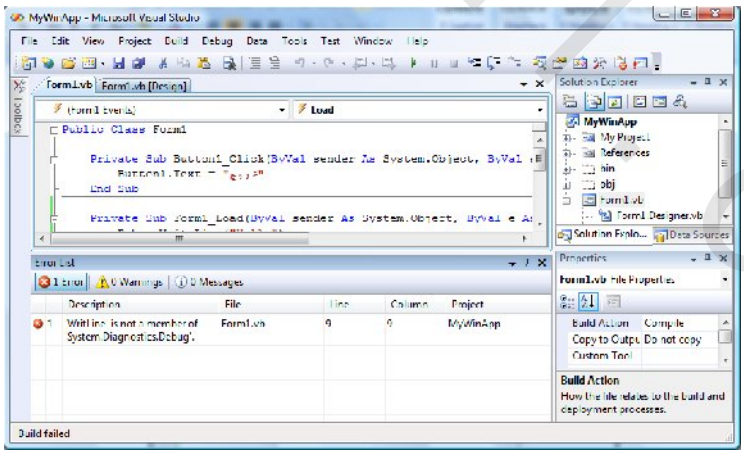
```
Debug.WriteLine("Hello")
Debug.WritLine("Goodbye")
```

٤. اضغط مفتاح F5 لتنفيذ البرنامج، تظهر رسالة خطأ تفيد وجود خطأ داخل الإجراء لتتـــمكن من تصحيحها (انظر شكل ٥-٢).



شكل ٥-٢ تظهر رسالة الخطأ دلالةً على وجود أخطاء بالكود.

٥. انقر الزر No، يقوم Visual Basic بإيقاف تنفيذ التطبيق وتظهر قائمة بالأخطاء داخل نافذة قائمة الأخطاء مع وصف مختصر لكل خطأ من هذه الأخطـــاء حيـــث يوجد خطأ واحد فقط فى الكود السابق (انظر شكل ٥-٣).



شكل ٥-٣ يظهر الخطأ بنافذة قائمة الأخطاء.

٦. انقر الخطأ نقراً مزدوجاً، تلاحظ ظهور شريط مضاء فوق الخطأ داخل نافذة الكود.
٧. لعلاج هذا الخطأ عدل الكلمة " **WritLine**" إلى "**WriteLine**" (أى أضف حرف **e** بعد الحرف **t**) وحينئذٍ يختفى الخطأ من لوحة قائمة الأخطاء. أعد تنفيذ البرنامج. تظهر لك العبارتين التاليتين داخل النافذة الفورية **Immediate Window**:

Hello
Goodbye

نفهم من ذلك أن **Visual Basic** تعرف على الأمرين بعد التصحيح ونفذ البرنامج وأظهر الرسالتين المطلوبتين.

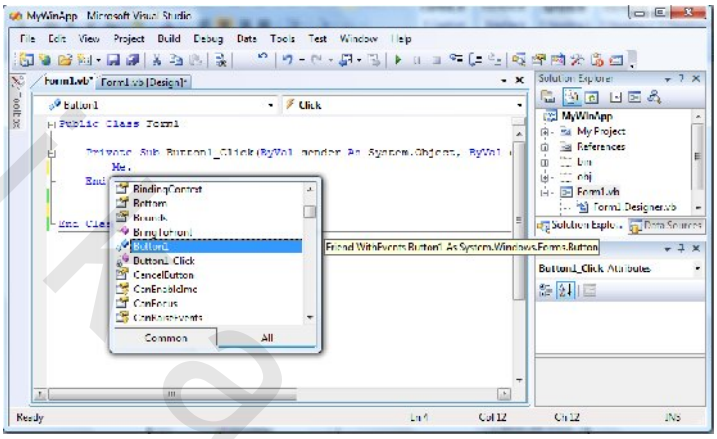
تجنب الأخطاء الهجائية

كما يقال الوقاية خير من العلاج. لتلافي أو تقليل الأخطاء التي تقع نتيجة كتابة كود البرنامج، فإننا ننصحك باتباع الآتي:

التكملة التلقائية

من المزايا الرائعة في **Visual Basic 2008** استخدام ما يسمى بالتكملة التلقائية للكود **Auto code-completion** من خلال السرد التلقائي للعناصر المتوقعة **Auto List Members**، مثلا لو كتبت اسم كائن ثم نقطة، سيدرك **Visual Basic** أنك ستكتب أحد خصائص أو وظائف الكائن ومن ثم يظهر قائمة بالخصائص والوظائف المتاحة لهذا الكائن. يوضح شكل ٥-٤ هذه الميزة فيكفي أن تكتب اسم كائن معرف في **Visual Basic**، ثم نقطة ليبدأ **Visual Basic** في سرد الخصائص المتاحة لهذا الكائن حيث يمكنك اختيار أحدها ونقره نقرا مزدوجا ليتولي **Visual basic** كتابته نيابة عنك في السطر قيد الكتابة، أو كتابة الأحرف الأولى من اسم الخاصية ليتم تعليمها في قائمة العناصر.

من الأشياء التي يتكفل بهـــا أيضـــا Visual Basic عمليـــة ضـــبط حالـــة الأحـــرف Capitalizing للكلمات المحجوزة ولأسماء المتغيرات مما يعطيك انطباع مباشر عن كتابتك الصحيحة للكلمات.



شكل ٥-٤ السرد التلقائي لخصائص الكائن من خلال خاصية التكملة التلقائية للكود

يمكنك تغيير ألوان نافذة تحرير الكود إذا أردت من خلال التبويب Fonts and Colors بالمجلد Environment داخل المربع الحوارى Options.

ثانياً: أخطاء أثناء التنفيذ

أثناء تشغيل البرنامج قد تحدث أخطاء، تسمي هذه الأخطاء أخطاء التشغيل Run-time Errors كما يطلق عليها "استثناءات" Exceptions.

وهى أخطاء تظهر فى مرحلة التنفيذ، وتحدث نتيجة عدم مقـــدرة Visual Basic علـــى التعرف عليها، وبذلك يعجز عن تنفيذها. ومن أمثلة هذا النوع من الخطأ "القسمة علـــى صفر"، حيث تظهر رسالة داخل Visual Basic تشير إلى خطأ وقع فيه المبرمج لأنـــه لم يتخيل أن قيمة مقام المعادلة قد تساوى صفر، وفى هذه الحالة لابد من علاج هذا الخطـــأ بأسلوب منطقى يجعل قيمة هذا المقام لا يساوى صفر تحت أى ظرف أو عند أى احتمال،

حيث من المعروف في علم الحساب أن أى قيمة كسرية تحتوى على بسط ومقام وتكـــون قيمة المقام صفرا، فإن الناتج لن يكون رقماً حقيقياً.

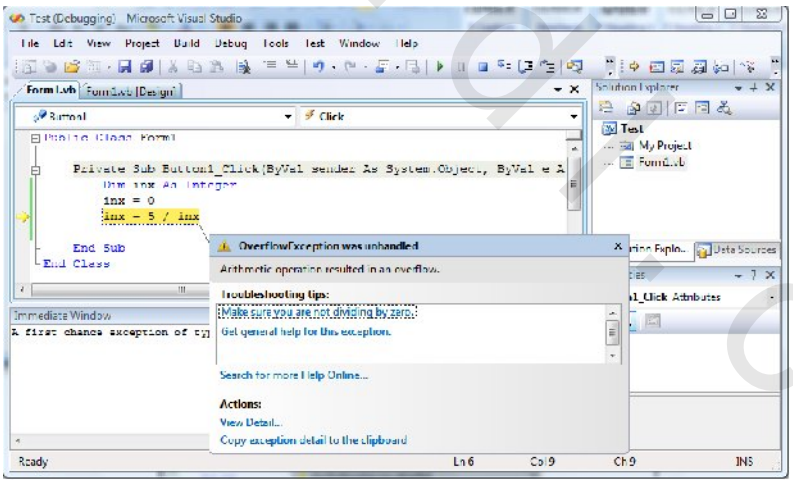
مثال:

سنقوم فيما يلي بكتابة برنامج بسيط وملاحظة الناتج في حالة القسمة على صفر.

١. أضف زر أمر إلى النموذج المفتوح.
٢. انقر زر الأمر الجديد نقراً مزدوجاً لفتح نافذة الكود.
٣. قم بكتابة الكود التالى داخل إجراء حدث نقر الزر:

```
Dim inx As Integer
inx = 0
inx = 5 / inx
```

٤. بعد كتابة التعليمات السابقة نفذ البرنامج. يظهر النموذج فى مرحلة التنفيذ ويظهر بداخله زر الأمر **Button1**.
٥. انقر زر الأمر **Button1**، يوقف **Visual Basic** تنفيذ البرنامج ويظهر رســـالة خطأ تحدد نوع الخطأ (الاستثناء) كما في شكل ٥-٥.



شكل ٥-٥ الرسالة التي تظهر عند حدوث أخطاء التشغيل.

يظهر في شكل ٥-٥ رسالة الخطأ، ومعناها أن هناك عملية حسابية تسببت فى طفح البيانات **Overflow**.

يظهر في مربع الرسالة الأسباب التى ربما أدت إلى ظهور الخطأ، كما يظهر السطر الذى تسبب فى الخطأ داخل نافذة الكود لكى تتمكن من تصحيحه.

أحيانا يكون سبب الخطأ سهل الاكتشاف مثل الكتابة الخاطئة، وأحيانا أخرى يكون السبب صعب الاكتشاف ويحتاج لخبرة وتعمق في تعقب الخطأ في البرنامج، لأن الخطأ في هذه الحالة يكون في سطر آخر من الكود غير الذي يحدده **Visual Basic**.

ثالثا: أخطاء منطقية

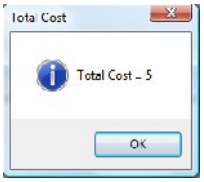
الأخطاء المنطقية ليست أخطاءً نحوية فى تعليمات البرنامج وليست أخطاءً تظهر فى مرحلة التنفيذ حيث أن **Visual Basic** يراجع كل التعليمات الموجودة بالبرنامج في مرحلة الكتابة ومرحلة التنفيذ، ولكن الخطأ هنا من نوع جديد وهو أن الناتج المطلوب من البرنامج غير متوقع أو غير منطقى.

مثال

الإجراء التالي يقوم بحساب إجمالى القيمة المطلوبة لشراء ألف جهاز كمبيوتر من نوع معين إذا كان سعر الجهاز الواحد يساوى خمسة آلاف جنيه.

```
Dim Num_Comps As Integer, Comp_Cost As Decimal
    Num_Comps = 1000
    Comp_Cost = 5000
    MessageBox.Show("Total Cost = " & Comp_Cost /
Num_Comps, "Total Cost", MessageBoxButtons.OK,
MessageBoxIcon.Information)
```

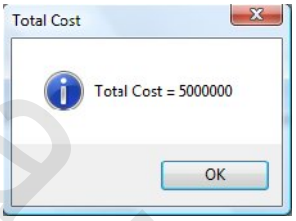
عند تنفيذ هذا البرنامج ستحصل على مربع الرسالة الموضح فى شكل ٥-٦.



شكل ٥-٦ نتيجة الإجراء السابق.

وهى بالطبع ليست النتيجة المتوقعة أو الصحيحة لتكلفة ألف جهاز قيمة كل جهاز خمسة آلاف جنية. والخطأ هنا أننا بدلا من ضرب تكلفة الجهاز في عدد الأجهزة، قمنا بقسمة تكلفة الجهاز على عدد الأجهزة. بعبارة أخرى الخطأ في توجيه معادلة حساب التكلفة الكلية، ومثل هذا الخطأ لن يكتشفه **Visual Basic** لا في مرحلة الكتابة ولا في مرحلة التنفيذ لأنه كما قلنا خطأ منطقياً.

ولتصحيح هذا الخطأ عد مرة أخرى إلى مرحلة التصميم وعدل معامل القسمة (/) ليكون معامل ضرب (*). ثم نفذ البرنامج كى تظهر النتيجة صحيحة (انظر شكل ٥-٧).



شكل ٥-٧ النتيجة بعد تصحيح الخطأ المنطقى.

وهى نتيجة مقبولة منطقياً. أما النتيجة السابقة فليست كذلك بأى حال من الأحوال حيث أنه لا يعقل أن يكون إجمالى التكلفة أقل من سعر جهاز واحد.

تجنب الأخطاء المنطقية

فيما يلي مجموعة من النصائح تساعدك في تجنب الوقوع في الأخطاء المنطقية أو تقليلها:

- بعد انتهائك من كتابة تعليمات البرنامج لا تتسرع بتنفيذه قبل أن تقوم بمراجعة هذه التعليمات.
- لا تستخدم أسماء متغيرات متشابهة.
- لا تتسرع بتعريف كل متغيراتك على أنها متغيرات عامة.
- تأكد من أن جميع تعليمات، إجراءات، وظائف البرنامج تعمل كما هو متوقع ومنطقى لتصميم البرنامج.

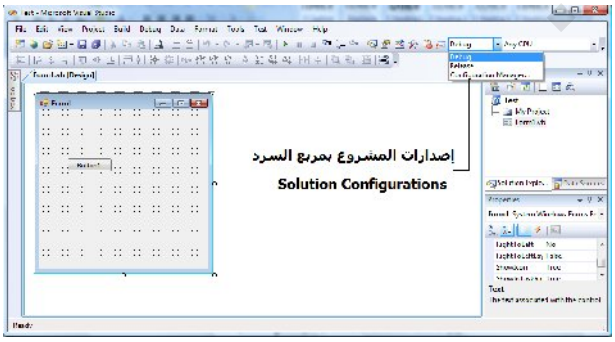
إصدارات المشروع

من الممكن أن يحتوى أى مشروع يتم تنفيذه داخل Visual Studio 2008 على إصدارين، الأول هو إصدار النشر Release version أو ما يطلق عليه "طور النشر" والثانى هو إصدار التصحيح Debug أو ما يطلق عليه "طور التصحيح".

طور النشر هو إصدار من البرنامج أو التطبيق الجاهز للتوزيع. لذا فهذا هو الإصدار الأخير من البرنامج والذى تتكاتف فيه جميع العوامل للحصول على أقصى تحسين فى البرنامج من حيث قلة الحجم وسرعة التنفيذ. فهذا الإصدار لا يحتوى على أية معلومات خاصة بعملية تتبع الأخطاء أو تصحيحها، لذا لا يمكنك اكتشاف أى أخطاء بالبرنامج من خلال هذا الإصدار ومن ثمَّ لا يمكن تصحيحها.

تطلق عبارة "أقصى تحسين للبرنامج" Optimization على العملية التى يتم من خلالها تنفيذ البرنامج بسرعة وكفاءة عالية مع تقليل حجم الكود المستخدم. وحقيقةً فإن نطاق NET. يدعم العديد من السمات التى تؤدى إلى الوصول لأقصى تحسين للبرنامج مثل سمة تجميع النفايات Garbage Collection.

يمكنك عرض طور النشر الخاص بالبرنامج الحالى عن طريق اختيار Release من مربع السرد Solutions Configurations بشريط الأدوات القياسى الموجود بنافذة بيئة تطوير Visual Studio 2008 (انظر شكل ٥-٨).

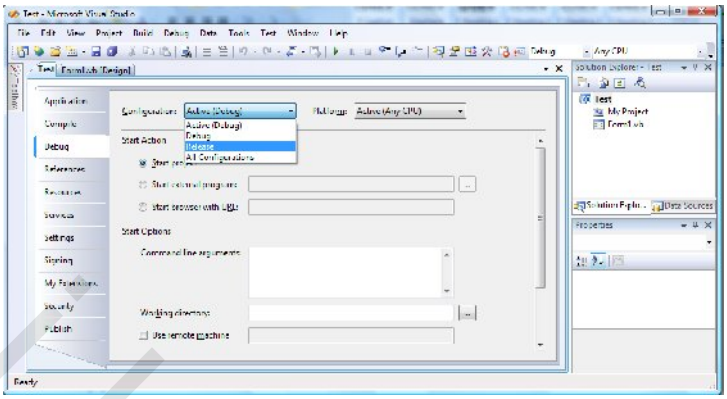


شكل ٥-٨ اختيار الإصدار المناسب من مربع السرد Solutions Configurations.

فى حالة عدم ظهور مربع السرد Solution Configurations بشريط الأدوات القياسى أو مربع السرد Configuration بصفحات الخصائص Property Pages ، تأكد من تنشيط مربع الاختيار Show advanced build configurations بالتبويب Projects and Solutions بالمربع الحوارى Options ثم افتح قائمة Tools بشريط القوائم واختر Import and Export Settings من القائمة المنسدلة الناتجة. انقر زر Next مرتين للانتقال إلى الشاشة الثالثة من المعالج ومنها اختر General Development Settings وقم بتكملة المعالج.

يمكنك التحكم فى إعدادات طور النشر من خلال شاشة الخصائص المصاحبة للمشروع. لأداء ذلك، تابع معنا الخطوات الآتية:

١. تأكد أنك داخل بيئة تطوير Visual Studio 2008 ثم افتح قائمة View من شريط القوائم واختر Solution Explorer من القائمة المنسدلة لإظهار مستكشف الحل إذا لم يكن ظاهراً أمامك.
٢. من نافذة المستكشف اختر اسم المشروع.
٣. افتح قائمة View مرةً أخرى ثم اختر Property Pages من القائمة المنسدلة، تظهر شاشة الخصائص الخاصة بالمشروع الحالى (انظر شكل ٥-٩).
٤. نشط التبويب Debug من العمود الأيسر بالشاشة ثم اختر Release من مربع السرد Configuration.
٥. قم بتعديل الخيارات حسبما يروق لك وذلك من الجزء السفلى بالشاشة.
٦. قم بحفظ تعديلاتك ثم قم بإغلاق شاشة الخصائص.



شكل ٥-٩ شاشة الخصائص Property Pages الخاصة بالمشروع الحالى

أما طور التصحيح Debug version فهو الإصدار المستخدم لتعقب أخطاء البرنامج وتصحيحها. لذا فهو لا يستخدم أياً من تقنيات التحسين لأن ذلك سيؤدى بالطبع إلى تعقيد الكود المستخدم بدلاً من تسهيله. وبدلاً من ذلك فإن هذا الإصدار يحتوى على جميع معلومات التصحيح الغير موجودة أساساً داخل طور النشر.

يمكنك عرض طور التصحيح الخاص بالمشروع الحالى عن طريق اختيار Debug من مربع السرد Solution Configurations بشريط الأدوات القياسى الموجود بنافذة بيئة تطوير Visual Studio 2008 (راجع شكل ٥-٨).

ويمكنك أيضاً من خلال شاشة صفحات الخصائص Property Pages التحكم فى إعدادات طور التصحيح. لأداء ذلك، تابع معنا الخطوات الآتية:

١. من نافذة المستكشف اختر اسم المشروع.

٢. افتح قائمة View ثم اختر Property Pages من القائمة المنسدلة، تظهر شاشة صفحات الخصائص Property Pages الخاصة بالمشروع الحالى (راجع شكل ٥-٩).

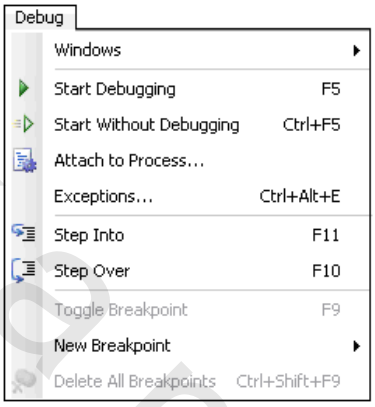
٣. اختر Debug من العمود الأيسر بالنافذة ثم اختر Debug من مربع السرد Configuration.

٤. قم بتعديل الخيارات حسبما يروق لك وذلك من الجزء السفلى من الشاشة.

٥. قم بحفظ التعديلات التى قمت بها ثم قم بإغلاق شاشة صفحات الخصائص.

إجراء عملية التصحيح

يمكنك البدء فى عملية تعقب الأخطاء وتصحيحها سطر بسطر باختيار أحد الخيارات الموجودة بقائمة **Debug** (انظر شكل ٥-١٠) وذلك كما يلى:



شكل ٥-١٠ خيارات قائمة **Debug** داخل بيئة التطوير **Visual Studio 2008**

- يتسبب الخيار **Start Debugging** (والذى يماثل ضغط مفتاح **F5** بلوحة المفاتيح) فى بدء تنفيذ التطبيق أو البرنامج إلى أن يجد نقطة توقف **Breakpoint**. وسوف نتعرض لنقاط التوقف بعد قليل.
- يتسبب الخيار **Start Without Debugging** (والذى يماثل ضغط الاختصار **Ctrl+F5** بلوحة المفاتيح) فى تنفيذ البرنامج دون اعتبار لعمليات التنقيح وهو بذلك يستخدم حينما نكون على ثقة من صحة الكود المستخدم ونرغب فى سرعة تنفيذ البرنامج.
- يتسبب الخيار **Step Into** (والذى يماثل ضغط مفتاح **F11** بلوحة المفاتيح) فى تنفيذ البرنامج للسطر الأول داخل الكود ثم التوقف، فإذا كان هذا السطر عبارة عن استدعاء لإحدى الدوال، يتم توجيه التحكم مباشرةً إلى داخل الدالة المستدعاة ثم التوقف داخل هذه الدالة قبل تنفيذ السطر الأول بها.

- يتسبب الخيار **Step Over** (والذى يماثل ضغط مفتاح **F10**) مثل الخيار السابق فى تنفيذ البرنامج للسطر الأول داخل الكود ثم التوقف. فإذا احتوى السطر الأول على استدعاء لإحدى الدوال، يتم توجيه التحكم إلى داخل الدالة ولكن فى هذه المرة يتم تنفيذ كود الدالة بالكامل ثم العودة مرةً أخرى إلى السطر التالى لسطر استدعاء الدالة ويتوقف تنفيذ البرنامج عند هذه النقطة.
- يتسبب الخيار **Step Out** (والذى يماثل ضغط الاختصار **Shift+F11** بلوحة المفاتيح) فى تغيير التحكم من داخل الدالة المستدعاة إلى الدالة التى قامت باستدعائها، وهذا الخيار يستخدم حينما يكون التحكم داخل الدالة المستدعاة، حيث يتم تنفيذ الدالة المستدعاة أولاً قبل الانتقال إلى الدالة التى قامت باستدعائها.
- يتسبب الخيار **Run to Cursor** فى بدء تنفيذ البرنامج حتى يتم الوصول إلى أقرب نقطة توقف أو الوصول إلى مكان المؤشر داخل الكود.
- يتسبب الخيار **Run to a Specified Function** فى بدء تنفيذ البرنامج حتى يتم الوصول إلى دالة معينة داخل الكود وحينئذٍ تتوقف عملية التنفيذ.

بعض هذه الخيارات لا يكون متاحاً أثناء التصميم وإنما يمكنك مشاهدته أثناء التوقف.

إيقاف عملية التصحيح

يمكنك فى أى وقت إيقاف عملية تصحيح البرنامج. لأداء ذلك، اختر **Stop Debugging** من قائمة **Debug**، حيث يظهر هذا الخيار فقط حينما تكون عملية التصحيح قد بدأت بالفعل. وبمجرد اختيارك لهذا الخيار، يتم إيقاف عملية التصحيح. فإذا أردت البدء مرةً أخرى فى تصحيح نفس البرنامج، اختر **Restart** من قائمة **Debug**.

نقاط التوقف Breakpoints

كما ذكرنا من قبل فإن نقطة التوقف **Breakpoint** عبارة عن مكان معين داخل الكود يتم عنده إيقاف تنفيذ البرنامج. يتيح لك المصحح تعيين أنواع نقاط التوقف التالية:

- نقطة توقف الملف **File Breakpoint** ويتم تعيينها فى مكان ما داخل الملف وبالتالى يتم إيقاف تنفيذ البرنامج بمجرد الوصول إلى هذا المكان.
- نقطة توقف الدالة **Function Breakpoint** ويتم تعيينها لدالة معينة وبالتالى يتم إيقاف تنفيذ البرنامج بمجرد الوصول إلى هذه الدالة.
- نقطة توقف البيانات **Data Breakpoint** ويتم تعينها لقيمة أحد المتغيرات وبالتالى يتم إيقاف تنفيذ البرنامج بمجرد تغيير قيمة هذا المتغير. حيث يمكنك تعيين هذا النوع من نقاط التوقف داخل الكود القومى فقط.
- نقطة توقف العنوان **Address Breakpoint** ويتم تعينها لمكان أو عنوان معين داخل الذاكرة وبالتالى يتم إيقاف تنفيذ البرنامج بمجرد الوصول إلى هذا المكان.

إنشاء نقاط توقف البيانات

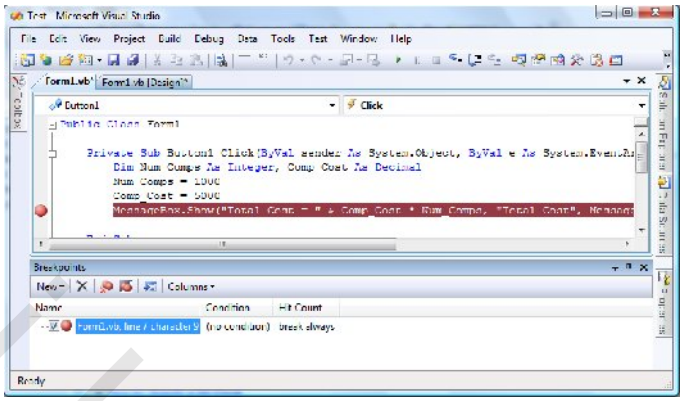
لإنشاء نقطة توقف بيانات بسيطة فى أى مكان داخل الكود، تابع معنا الخطوات الآتية:

١. انقر داخل السطر الذى ترغب فى وضع نقطة توقف عنده داخل نافذة الكود.

٢. قم بأداء أيٍ مما يلى:

- انقر بزر الفأرة الأيمن ثم اختر **Breakpoint** ثم **Insert breakpoint** من القائمة الموضعية الناتجة.
- افتح قائمة **Debug** بشريط القوائم ثم اختر **Toggle breakpoint** من القائمة المنسدلة الناتجة.
- اضغط مفتاح **F9** بلوحة المفاتيح.

وفى جميع الحالات يظهر الرمز ● إلى اليسار من السطر المحدد دلالةً على وجود نقطة توقف فى هذا المكان (انظر شكل ٥-١١).



شكل ٥-١١إضافة نقطة توقف إلى الكود.

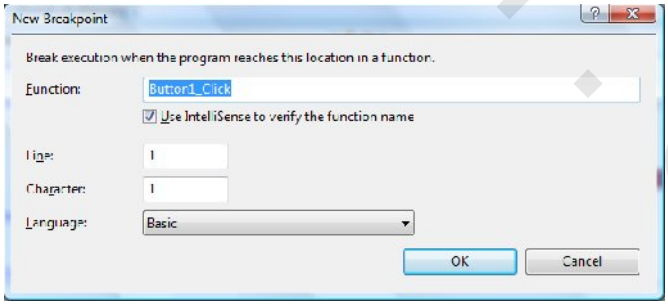
ويمكنك فى أى وقت حذف نقطة التوقف باستخدام نفس الخطوات السابقة، حيث تعمل نقاط التواقف كزر مفصلى يتم إنشاؤه أو حذفه باستخدام نفس الخطوات.

إنشاء نقطة توقف دالة

لإنشاء نقطة توقف داخل إحدى الدوال الموجودة داخل الكود، تابع معنا الخطوات التالية:

١. اختيارياً، انقر داخل الدالة التى ترغب فى وضع نقطة التوقف عندها.

٢. افتح قائمة **Debug** من شريط القوائم ثم اختر **New Breakpoint** ثم **Break at Function** من القوائم المنسدلة الناتجة (أو اضغط الاختصار **Ctrl+B** من لوحة المفاتيح)، يظهر المربع الحوارى **New Breakpoint** (انظر شكل ٥-١٢).



شكل ٥-١٢ المربع الحوارى **New Breakpoint** .

٣. فى حالة عدم ظهور اسم الدالة داخل مربع النص **Function** (حيث يظهر الاسم بطريقة تلقائية بمجرد النقر داخل اسم الدالة فى الخطوة رقم ١)، قم بإدخال اسم الدالة إلى مربع النص. تأكد من تنشيط مربع الاختيار **Use IntelliSense to verify the function name** إذا أردت التعرف على اسم الدالة بصورة تلقائية.

٤. يتم تلقائياً وضع نقطة التوقف فى بداية الدالة. فإذا أردت وضعها فى مكان آخر بالدالة، قم بتغيير قيم مربعى النص **Line** و**Character**، حيث يتم من خلال الأول تعيين رقم السطر داخل الدالة، بينما يتم من خلال الثانى تعيين مكان نقطة التوقف داخل السطر المحدد بالمربع الأول.

٥. تأكد من اختيار **Basic** بالقائمة المنسدلة **Language** ثم انقر زر **Ok** لإغلاق المربع الحوارى **New Breakpoint** وإضافة نقطة التوقف إلى الكود.

تعديل سلوك نقاط التوقف

يمكنك تعديل سلوك نقاط التوقف عن طريق إضافة بعض الخصائص إليها، حيث يمكنك استخدام الخاصية **Hit Count** والخاصية **Condition** كما يلى:

الخاصية Hit Count

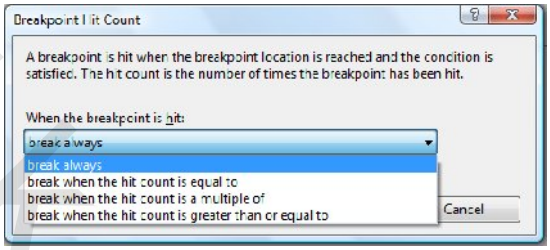
تقوم هذه الخاصية بحساب عدد المرات التى يتم فيها الوصول إلى نقطة توقف معينة قبل أن يقوم البرنامج بالتوقف بالفعل. وفى حالة نقاط توقف البيانات **Data Breakpoints** تقوم هذه الخاصية بحساب عدد المرات التى تغيرت فيها قيمة هذا المتغير.

وحقيقةً فإن القيمة الافتراضية لهذه الخاصية تتيح توقف البرنامج عند كل نقطة توقف (القيمة **Break always**)، إلا أنك تستطيع تغيير هذه القيمة. لأداء ذلك، تابع معنا الخطوات الآتية:

١. من نافذة نقاط التوقف **Breakpoints**، انقر نقطة التوقف التى ترغب فى تعيين قيم الخاصية لها بزر الفأرة الأيمن ثم اختر **Hit Count** من القائمة الموضعية الناتجة.

(أو انقر نقطة التوقف داخل نافذة الكود بزر الفأرة الأيمن ثم اختر **Breakpoint** من القائمة الموضعية الناتجة ومنها اختر **Hit Count**).

فى الحالتين يظهر المربع الحوارى **Breakpoint Hit Count** (انظر شكل ٥-١٣).



شكل ٥-١٣ المربع الحوارى **Hit Count**.

٢. اختر الخيار الذى يروق لك من مربع السرد **When the breakpoint is hit**. اختر **break always** إذا أردت الوقوف عند نقطة التوقف دائماً، أو **break when the hit count is equal to** إذا أردت الوقوف عند نقطة التوقف بعد عدد معين من المرات يتم تحديده من خلال مربع النص المجاور، وكذلك الحال بالنسبة للخيار الثالث والرابع.

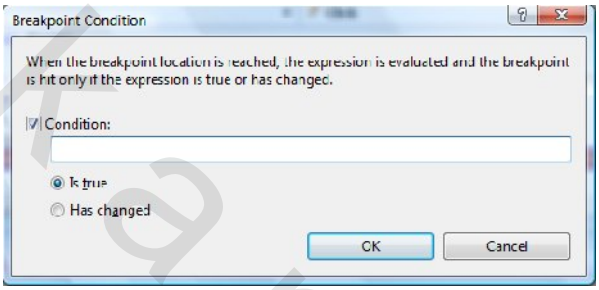
٣. للرجوع فى أى وقت إلى القيمة الافتراضية للخاصية، انقر الزر **Reset**.

٤. انقر زر **Ok** لإغلاق المربع الحوارى **Breakpoint Hit Count** وتعيين القيمة الجديدة للخاصية **Hit Count**.

الخاصية Condition

تقوم هذه الخاصية بتعيين شرط معين يتم حسابه بمجرد الوصول إلى نقطة التوقف، ومن ثمَّ يتم استخدام قيمة هذا الشرط لتحديد الوقوف عند هذه النقطة من عدمه. فإذا تحقق الشرط يتم الوقوف عند نقطة التوقف وإلا يتم تجاهلها والاستمرار فى تنفيذ البرنامج كأنها غير موجودة بالمرة. لإنشاء نقطة توقف جديدة مع تعيين الخاصية **Condition**، تابع معنا الخطوات الآتية:

١. من نافذة نقاط التوقف **Breakpoints**، انقر نقطة التوقف التى ترغب فى تعـــيين قيم الخاصية لها بزر الفأرة الأيمن ثم اختر **Condition** من القائمة الموضعية الناتجة.
أو (انقر نقطة التوقف داخل نافذة الكود بزر الفأرة الأيمن ثم اختر **Breakpoint** من القائمة الموضعية الناتجة ومنها اختر **Condition**).
فى الحالتين سيظهر قم بإظهار المربع الحوارى **Breakpoint Condition** (انظر شكل ٥-١٤).



شكل ٥-١٤ المربع الحوارى **Breakpoint Condition** .

٢. قم بإدخال الشرط المناسب إلى مربع النص **Condition**.

٣. نشط زر الاختيار **Is true** إذا أردت التوقف عند تحقق الشرط المحدد فى الخطـــوة السابقة أو زر الاختيار **Has changed** إذا أردت التوقف عند تغير قيمة الشرط المحدد.

٤. انقر زر **Ok** لإغلاق المربع الحوارى **Breakpoint Condition** وتعيين القيمـــة الجديدة للخاصية **Condition**.

إزالة نقاط التوقف

يمكنك فى أى وقت إزالة إحدى نقاط التوقف المعرفة داخل الكود كما يمكنك أيضاً إزالة جميع نقاط التوقف مرةً واحدة. لإزالة إحدى نقاط التوقف، انقر هذه النقطة بزر الفـــأرة الأيمن ثم اختر**Breakpoint** ثم **Delete Breakpoint** من القائمة الموضعية الناتجة. أما

إذا أردت إزالة جميع نقاط التوقف مرةً واحدة، فافتح قائمة **Debug** ثم اختـــر **Delete All Breakpoints** من القائمة المنسدلة.

نوافذ المصحح

إحدى الوظائف الأساسية للمصحح تتمثل فى توفير معلومات عن البرنـــامج فى المراحـــل المختلفة لتنفيذه، وهذه المعلومات غالباً ما تكون متاحة فى طور التوقـــف **Break Mode** وهو الوقت الذى يتم فيه إيقاف تنفيذ البرنامج. وأبسط طرق الحصول على المعلومـــة فى هذا الوقت تتمثل فى الإشارة إلى الكائن الذى ترغب فى استرجاع معلوماته وليكن متغيراً أو دالة مثلاً وذلك من النافذة الأساسية **Source Window** التى تحتوى على الملفـــات الأساسية للبرنامج، وحينئذٍ تظهر قيمة الكائن المختار داخل مربع نص صغير يسمى مربع المعلومات **Information Box**.

الطريقة الأخرى للحصول على معلومات عن البرنامج هى استخدام نوافذ المصحح، حيث يتيح نطاق **.NET** عدداً من النوافذ التى يمكنك استدعائها للحصـــول علـــى المعلومـــات المختلفة عن البرنامج قيد الفحص أو التصحيح. وهذه النوافذ هى:

- النافذة **Autos**
- النافذة **Breakpoints**
- النافذة **Call stack**
- النافذة **Locals**
- النافذة **Modules**
- النافذة **Threads**
- النافذة **Watch**

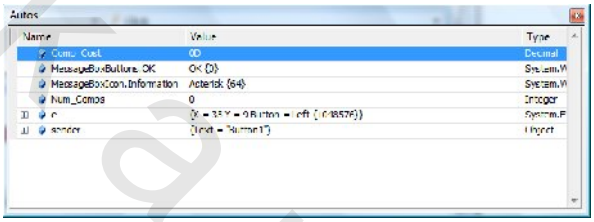
وفيما يلى نقوم بإلقاء نظرة خاطفة على كل نافذة من هذه النوافذ.

النافذة Autos

يمكنك استخدام هذه النافذة فى طور التوقف فقط، وتحتوى على معلومات تعكـــس

قيم وأسماء المتغيرات المستخدمة فى العبارة الحالية وكذلك العبارة السابقة، حيث تقوم النافذة بالتعرف على قيمة متغيرات العبارة تلقائياً ومن هنا جاء اسم النافذة.

تقوم هذه النافذة افتراضياً بإظهار الأرقام الصحيحة كأنواع عشرية إلا أنك تستطيع تغيير هذا السلوك باستخدام القائمة الموضعية. للوصول إلى النافذة **Autos**، افتح قائمة **Debug** ثم اختر **Windows** و**Autos** من القوائم المنسدلة حيث تظهر النافذة كما فى شكل ٥-١٥. يمكنك تغيير قيمة أى من المتغيرات الموجودة. لأداء ذلك، انقر القيمة المراد تغييرها نقراً مزدوجاً ثم قم بكتابة القيمة الجديدة مع تحرى الدقة فى تعيين القيم الجديدة.



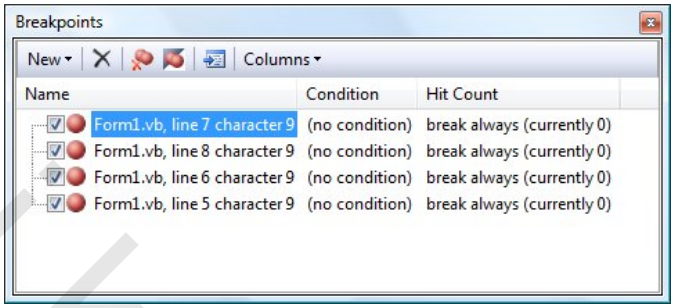
شكل ٥-١٥ تقوم النافذة **Autos** بعرض المتغيرات المستخدمة فى العبارة الحالية والسابقة.

النافذة Breakpoints

تقوم هذه النافذة بعرض قائمة بجميع نقاط التوقف الموجودة بالبرنامج، كما تقوم النافذة بعرض الشروط المصاحبة لنقاط التوقف إن وجدت. كما تحتوى النافذة أيضاً على شريط أدوات يحتوى على العديد من الخيارات التى تمكنك من إضافة نقطة توقف جديدة أو تعديل نقطة موجودة أو حتى حذفها. كما تقوم النافذة بعرض معلومات عن اسم الدالة أو الملف الذى يحتوى على نقطة التوقف.

يتم افتراضياً عرض اسم نقطة التوقف وقيم الخاصيتين **Condition** و**Hit Count**. فإذا أردت الحصول على المزيد من المعلومات عن إحدى النقاط، انقر هذه النقطة بزر الفأرة الأيمن داخل النافذة **Breakpoints** ثم اختر **Properties** من القائمة الموضعية الناتجة حيث تظهر نافذة خصائص نقطة التوقف التى يمكنك من خلالها التعرف على معلومات إضافية عن نقطة التوقف المختارة.

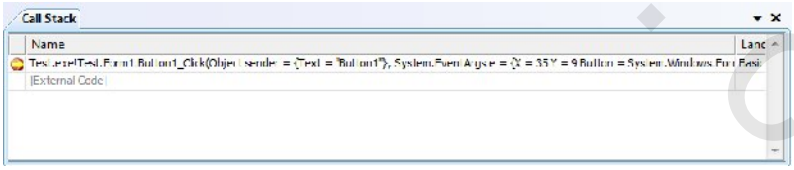
لفتح النافذة **Breakpoints**، افتح قائمة **Debug** ثم **Windows** واختر **Breakpoints** من القائمة الناتجة، تظهر نافذة **Breakpoints** (انظر شكل ٥-١٦).



شكل ٥-١٦ النافذة **Breakpoints**.

النافذة Call Stack

يمكنك استخدام هذه النافذة فى طور التوقف فقط، وتحتوى على معلومات عن الدوال الموجودة فى الوقت الحالى داخل جزء الذاكرة **Stack** وكذلك معاملات الدالة وقيم هذه المعاملات. لفتح نافذة **Call Stack**، افتح قائمة **Debug** ثم **Windows** واختر **Call Stack** من القائمة المنسدلة تظهر نافذة **Call Stack** (انظر شكل ٥-١٧). لتغيير المعلومات التى تظهر بالنافذة، انقر النافذة بزر الفأرة الأيمن ثم قم بتنشيط أو تعطيل الخيارات .... **Show** من القائمة الموضعية الناتجة. أما إذا أردت مشاهدة الكود المصدرى لإحدى الدوال، فانقر هذه الدالة بزر الفأرة الأيمن داخل نافذة **Call Stack** ثم اختر **Go To Source Code** من القائمة الموضعية.



شكل ٥-١٧ النافذة **Call Stack**.

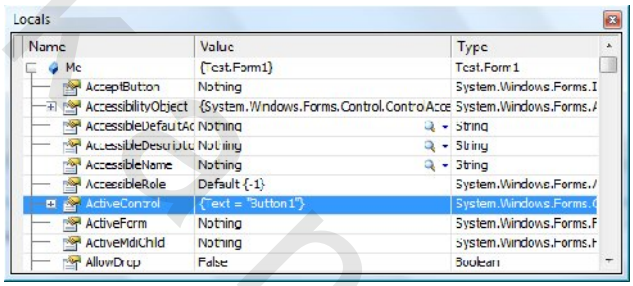
النافذة Locals

تحتوى هذه النافذة على معلومات عن المتغيرات المحلية الموجودة داخل سياق الكود الحالى حيث تقوم بعرض اسم المتغير وقيمته وكذلك نوعه. لعرض النافذة، افتح قائمة **Debug**

وWindows ثم اختر Locals من القائمة المنسدلة، تظهر نافذة Locals (انظر شكل ٥-١٨).

يقصد بالسياق الحالى افتراضياً الدالة التى تحتوى على موضع التنفيذ الحالى. فإذا أردت تغيير هذا السياق، افتح قائمة View ثم اختر Toolbars وDebug Location من القوائم المنسدلة.

لتغيير المعلومات الموجودة داخل النافذة Locals، تأكد أنك داخل طور التوقف وانقر القيمة المطلوب تغييرها نقراً مزدوجاً ثم قم بإدخال القيمة الجديدة.

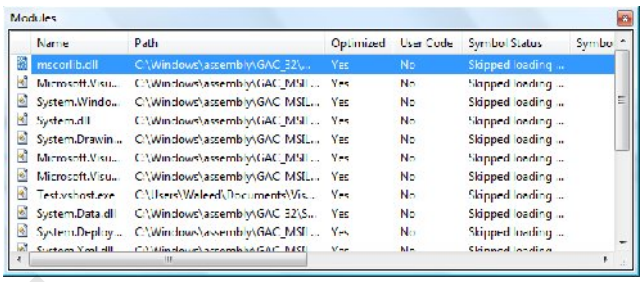


شكل ٥-١٨ النافذة Locals.

النافذة Modules

تعرض هذه النافذة قائمة بجميع النماذج Modules المستخدمة من قِبَل البرنامج قيد التنفيذ حيث تتكون هذه النماذج من الملفات التنفيذية exe. أو ملفات الربط dll. حيث يظهر اسم النموذج وعنوانه ومساره وكذلك ترتيب النماذج داخل البرنامج بالإضافة إلى رقم الإصدار والمعرف الذى يخصصه البرنامج لكل نموذج من هذه النماذج. لعرض النافذة، افتح قائمة Debug ثم Windows واختر Modules من القوائم المنسدلة، تظهر نافذة Modules (انظر شكل ٥-١٩).

يتم ترتيب النماذج داخل النافذة طبقاً لترتيب تحميلها فى هذه النافذة، إلا أنك تستطيع تغيير هذا الترتيب بنقر الزر المناسب الموجود أعلى كل عمود داخل النافذة.



شكل ٥–١٩ النافذة Modules.

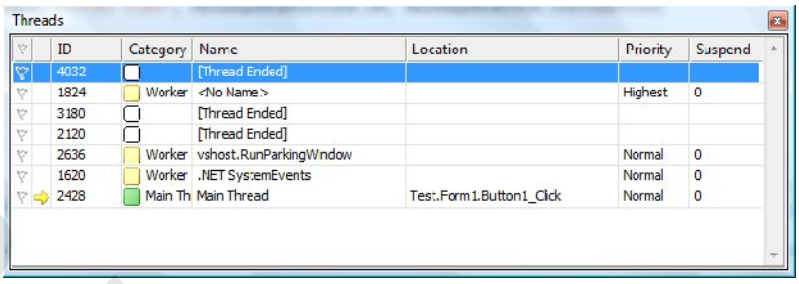
النافذة Threads

تعرض هذه النافذة قائمة بجميع الخيوط أو المسارات Threads فى البرنامج متعدد المسارات أو الاتجاهات، حيث تحتوى على معلومات إضافية عن اسم المسار ومعرفه وأولويته وعنوانه وكذلك اسم الوظيفة التى يرتبط بها هذا المسار.

والخيط أو المجرى Thread عبارة عن مسار تنفيذى داخل البرنامج، حيث يحتوى البرنامج افتراضياً على مجرى واحد فقط. فإذا قام البرنامج باستخدام أكثر من مجرى تنفيذى، يطلق عليه فى هذه الحالة برنامج متعدد الخيوط أو المسارات Multithreaded Program. وعلى الرغم من ذلك يتم تنفيذ مسار واحد فقط فى الوقت الواحد ويسمى فى هذه الحالة المسار النشط Active Thread.

يمكنك استخدام النافذة Threads للتحكم فى الخيوط أو المسارات الموجودة داخل البرنامج. فإذا أردت نشيط أحد الخيوط، انقر هذا الخيط بزر الفأرة الأيمن ثم اختر Switch to Thread من القائمة الموضعية. كما يمكنك تجميد الخيط بمعنى عدم تنفيذه. لأداء ذلك، انقر اسم الخيط بزر الفأرة الأيمن ثم اختر Freeze من القائمة الموضعية. فإذا أردت تنفيذه مرةً أخرى، انقره بزر الفأرة الأيمن ثم اختر Thaw فى هذه المرة من القائمة الموضعية.

لعرض هذه النافذة، تأكد أنك فى طور التوقف ثم افتح قائمة Debug واختر Windows وThreads من القوائم المنسدلة، تظهر نافذة Threads (انظر شكل ٥–٢٠).



شكل ٥-٢٠ النافذة **Threads**.

النافذة Watch

يمكنك استخدام هذه النافذة لكتابة تعبير وإيجاد قيمته، كما يمكنك تحرير هذا التعبير إذا احتوى على اسم مسجل أو متغير بينما لا يمكنك بحالٍ من الأحوال تحرير التعبير الثابت. لفتح هذه النافذة، افتح قائمة **Debug** ثم اختر **Windows** و**Watch** من القوائم المنسدلة ومنها اختر إحدى نوافذ المراقبة المتعددة.

لإدخال التعبير الذى ترغب فى إيجاد قيمته، انقر أى صف خالى بالعمود **Name** داخل النافذة نقراً مزدوجاً ثم قم بإدخال التعبير المطلوب حيث تظهر قيمة التعبير مباشرةً داخل العمود **Value**. وإذا أردت تعديل قيمة أحد المتغيرات أو المسجلات، اختر هذه القيمة ثم انقرها نقراً مزدوجاً وقم بإدخال القيمة الجديدة مباشرةً.