**الفصل العاشر**

**التعامل مع المجلّدات والملفّات**

من أهّم المهامّ في البرمجة، تعاملك مع الملفّات لحفظ واستعادة البيانات.. وفي هذا الفصل سنناقش الخلايا التي تمكّنك من التعامل مع المجلّدات والملفّات.

في البداية يجب أن تعرفَ أنّ كلّ هذه الخلايا مجموعةٌ تحت الاسم System.IO، حيث IO هي اختصار InputOutput أي المدخلات والمخرجات.. ولو كتبت

System.IO.

داخل أيّ إجراء، فستظهر لك فور كتابتك للنقطة، قائمة بكلّ أعضاء هذا الفضاء Name Space، حيث ستجد كلّ الخلايا والاستثناءات والمرقّمات المسئولة عن التعامل مع الملفّات والمجلّدات.. أنصحك بأن تقرأ أسماءها في عجالة، حتّى تأخذ فكرةً عمّا تمنحه لك VS من قدرات.

على العموم، ستكون أوّل خطوة في تعاملك مع خلايا المجلدات والملفّات، هي أن تكتب في بداية الملفّ جملة الاستيراد التالية:

**Imports System.IO**

لا تنسَ ذلك، وإلا فلن تعمل الأمثلة التي ستجدها هنا.

انته أيضا إلى أنّنا لن نقتصر على التعامل مع الملفّات باستخدام خلايا إطار العمل، فسنستخدم كذلك كائن نظام الملفّات FileSystemObject، ودوال VB التقليديّة.. هذا بالإضافة للتعرّف على بعض الأدوات، مثل مراقب نظام الملفّات، ومربعي حوار فتح ملفّ وحفظ ملفّ.

* [**التعامل مع المجلّدات والملفّات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\10-%20الملفّات\1-%20التعامل%20مع%20المجلّدات%20والملفّات.htm)
* [**القراءة من الملفّ والكتابة فيه:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\10-%20الملفّات\2-%20القراءة%20من%20الملفّ%20والكتابة%20فيه.htm)
* [**مُراقب نظام الملفّات FileSystemWatcher:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\10-%20الملفّات\3-%20مُراقب%20نظام%20الملفّات.htm)
* [**مربعا حوار "فتح ملفّ" و"حفظ ملفّ":**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\10-%20الملفّات\4-%20مربعا%20حوار%20فتح%20ملفّ%20وحفظ%20ملفّ.htm)

**التعامل مع المجلّدات والملفّات**

* + [**استخدام خلايا إطار العمل .NET FrameWork:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20التعامل%20مع%20المجلّدات%20والملفّات\1-%20التعامل%20مع%20المجلّدات%20والملفّات.htm)
  + [**استخدام كائن نظام الملفّات FileSystemObject:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20التعامل%20مع%20المجلّدات%20والملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات.htm)
  + [**استخدام جمل ودوالّ VB6:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20التعامل%20مع%20المجلّدات%20والملفّات\3-%20استخدام%20جمل%20ودوالّ%20VB6.htm)

**استخدام خلايا إطار العمل FrameWork**

قبل أن تستخدم أيّا من الخلايا التالية، لا تنسَ أن تكتب في بداية الملفّ جملة الاستيراد التالية:

**Imports System.IO**

* + - [**خليّة المجلّد Directory Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\0-%20خليّة%20المجلّد.htm)
    - [**خليّة الملفّ File Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\1-%20خليّة%20الملفّ.htm)
    - [**خليّة معلومات المجلّد DirectoryInfo Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\2-%20خليّة%20معلومات%20المجلّد.htm)
    - [**خليّة معلومات الملفّ FileInfo Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\3-%20خليّة%20معلومات%20الملفّ.htm)
    - [**خليّة المسار Path Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\4-%20خليّة%20المسار.htm)
    - [**مشروع "المتصفّح الخاصّ" CustomExplorer:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\5-%20مشروع%20المتصفح%20الخاص.htm)
    - [**المرور عبر كل المجلّدات الفرعية لمجلّد:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20خلايا%20إطار%20العمل\6-%20المرور%20عبر%20كل%20المجلّدات%20الفرعية%20لمجلّد.htm)

**خليّة المجلّد Directory Class**

لن تحتاج لتعريف متغيّر من هذه الخليّة، فوسائلها وخصائصها مشتركة Shared، يمكن استخدامها مباشرة من الخليّة نفسها.

**الوسائل:**

**إنشاء مجلّد CreateDirectory:**

يمكنك أن تُنشئ مجلّدا جديدا عن طريق هذه الوسيلة.. وطبعا لا بدّ أن يكون لها معامل على الأقلّ، هو مسار هذا المجلّد الجديد:

**Directory.CreateDirectory(مسار المجلّد)**

ويمكن أن يكون المسار كاملا، مثل:

**Directory.CreateDirectory("C:\folder1\folder2\folder3")**

ولو لم يكن أيّ من المجلّدين folder1، و folder2 موجودا، فإنّه يُنشأ أولا قبل إنشاء المجلّد folder3.

كما يمكن أن يكون المسار نسبيّا مثل:

**Directory.CreateDirectory("folder3")**

حيث يتمّ إنشاء المجلّد في هذه الحالة في المحرّك Drive الحاليّ والمسار الحاليّ، وسنعرف بعد قليل كيف نتحكّم فيهما.

ولدالة إنشاء المجلّد قيمة معادة، هي كائن من النوع "معلومات المجلّد" DirectoryInfo، وهي خليّة أخرى سنتعرّف عليها سريعا في هذا الفصل، مهمّتها كما يبدو من اسمها، أن تمنحك معلومات عن المجلّد الجديد، وبعض الوسائل للتعامل معه.

ومن الممكن أن يحدث استثناء عند استخدام هذه الدالة، إذا كان المسار أطول من اللازم، أو كنت تحاول إنشاء المجلّد في مسار لا تملك تصريحا بالتعامل معه.

ولكن لن يحدث أيّ استثناء إذا حاولت إنشاء مجلّد موجود بالفعل.. كلّ ما سيحدث هو أن ترجع لك الدالة كائن معلومات المجلّد الموجود، دون أن تنشئ شيئا.

**حذف Delete:**

هذه الوسيلة تحذف المجلّد إذا كان موجودا:

**Directory.Delete(مسار المجلّد)**

فإذا كان بالمجلّد مجلّدات فرعيّة، فلن يتّم حذفها، وبالتالي لن يتمّ حذف المجلّد أيضا، وسيتمّ إطلاق استثناء.. فإذا أردت حذف المجلّد بكلّ ما يتفرّع منه من مجلّدات، فاستخدم الصيغة التالية:

**Directory.Delete(مسار المجلّد, True)**

انظر لهذا المثال:

**Directory.CreateDirectory("c:/folder1/folder2/folder3")**

**Try**

**' حاول حذف المجلّد باعتبار أنّه فارغ**

**Directory.Delete("c:\folder1", False)**

**Catch exc As IOException**

**' هل يحتوي نصّ الاستثناء على النصّ "المجلّد غير فارغ"؟**

**If exc.ToString.IndexOf("The directory is not empty") >= 0 Then**

**Dim reply As MsgBoxResult**

**reply = MsgBox("هل تريد حذف المجلّد بما يحتويه؟", \_**

**MsgBoxStyle.YesNo)**

**If reply = MsgBoxResult.Yes Then**

**Directory.Delete("c:\folder1", True)**

**Else**

**MsgBox(exc.Message)**

**End If**

**End If**

**End Try**

وهناك استثناء آخر سينطلق، إذا حاولت أن تحذف مجلدا غير موجود.

**موجود Exists:**

ترجع هذه الوسيلة True إذا كان المجلّد الذي أرسلته لها كمعامل موجودا.

**Directory.Exists(مسار المجلّد)**

**قراءة وتغيير وقت إنشاء المجلّد:**

لسبب لا تدريه، تمّ منحك هاتين الإمكانيتين في صورة وسيلتين: GetCreationTime، و SetCreationTime، مع أنّ خاصيّة واحدة كان بإمكانها أن تقوم بالوظيفتين بطريقة أفضل!

**Dim CreatedOn As Date = Directory.GetCreationTime(مسار المجلّد)**

**Directory.SetCreationTime(مسار المجلّد, تاريخ الإنشاء)**

**قراءة وتغيير آخر وقت تمّت فيه قراءة المجلّد:**

يمكنك أن تفعل هذا عن طريق الوسيلتين GetLastAccessTime، و SetLastAccessTime.. ويمكنك استخدام هذه الدالة، في حذف المجلّدات التي لم تعد تُستخدم من وقت طويل.

**قراءة وتغيير آخر وقت تمّت فيه الكتابة في المجلّد:**

يمكنك أن تفعل هذا عن طريق الوسيلتين GetLastWriteTime و SetLastWriteTime.

**قراءة وتغيير المجلّد الحاليّ:**

المجلّد الحاليّ هو ذلك الذي تتمّ القراءة أو الحذف منه أو الكتابة فيه (على حسب الوسيلة التي تستخدمها) إذا لم تذكر مسار المجلّد أو الملفّ كاملا لهذه الوسيلة.. وفي الوضع الافتراضيّ يكون المجلّد الحاليّ هو مجلّد المشروع.. فمثلا لو استخدمت الجملة التالية:

**Directory.Exists("folder1")**

فإنّ الوسيلة Exists ستتحقّق من وجود المجلّد folder1 في المجلّد الحاليّ، وذلك لأنّ مسار المجلّد ليس كاملا.

ويمكنك أن تعرف المجلّد الحاليّ عن طريق الوسيلة GetCurrentDirectory، ويمكنك أن تغيّره عن طريق الوسيلة SetCurrentDirectory.

افترض الآن أنّ المجلّد الحالي هو:

C:\folder1\folder2\folder3\folder4

يمكنك أن تستخدم الجملة التالية لتغيير المسار الحالي إلى المجلّد "C:\folder1\folder2\folder2\_1" :

**Directory.SetCurrentDirectory("C:\folder1\folder2\_1")**

كما يمكنك أن تفعل نفس الشيء بالجملة التالية:

**Directory.SetCurrentDirectory("..\..\..\folder2\_1")**

حيث تدلّ النقطتان ".." على المجلّد الأعلى في المستوى من المجلّد الحاليّ.. فالجملة السابقة تخبر الدالّة أن تنتقل من المجلّد الحالي folder4)) ثلاثة مستويات لأعلى (حيث ستصل للمجلّد folder1) ، ثمّ تجعل المجلّد folder2\_1 (تحت المجلّد folder1) هو المجلّد الحاليّ ـ بافتراض أنّ هناك مجلدا بهذا الاسم.

غنيّ عن الذكر أنّ استثناءً سوف ينطلق إذا حاولت جعل مجلّد غير موجود هو المجلّد الحاليّ.

**اقرإ المجلّدات GetDirectories:**

يمكنك بهذه الوسيلة أن تحصل على مصفوفة نصّيّة، تحتوي على أسماء جميع المجلّدات الموجودة داخل المجلّد المطلوب (ولكن من دون أسماء المجلّدات الفرعيّة من هذه المجلّدات.. باختصار: ستحصل على المستوى الأوّل فقط من المجلّدات الفرعيّة).

**Dim Dirs() As String = Directory.GetDirectories(مسار المجلّد)**

ولهذه الوسيلة صورة أخرى شيّقة، تمكّنك من استخدام صيغة معيّنة لاختيار المجلّدات الفرعيّة على أساسها:

**Dirs = Directory.GetDirectories(مسار المجلّد, صيغة البحث)**

وتتكّون صيغة البحث من بعض الحروف التي تريد وجودها في اسم المجلّد، مع إمكانيّة استخدام العلامة "\*" التي ذكرنا من قبل أنّها تقوم بدور أيّ عدد من الحروف، أو العلامة "?" التي تقوم بدور حرف واحد فقط.. والجملة التالية مثلا ترجع لك كلّ المجلّدات الموجودة في المجلّد Windows ويحتوي اسم كلّ منها على كلمة SYSTEM مسبوقة أو متبوعة بأيّ عدد من الحروف:

**Dirs = Directory.GetDirectories("C:\Windows", "\*SYSTEM\*")**

وباستخدام العلامتين "\*" و"?" يمكنك تكوين صيغ مركّبة، فمثلا لو أردت أن تبحث عن أيّ اسم يبدأ بحرف واحد مهما كان، تليه كلمة Sys ثمّ تتبعه مجموعة من الحروف ثمّ حرف m ثم أيّ عدد من الحروف، فاستخدم الصيغة: "?Sys\*m\*".

**اقرإ الملفّات GetFiles:**

هذه الوسيلة مشابهة للسابقة، إلا إنّها تعيد مصفوفة بأسماء الملفّات الموجودة في المجلّد المرسل إليها كمعامل:

**Dim files() As String = Directory.GetFiles(مسار المجلّد)**

وطبعا توجد صيغة أخرى من هذه الوسيلة، تمكّنك من البحث عن أسماء معيّنة أو امتدادات معيّنة للملفّات:

**files = Directory.GetFiles(مسار المجلّد, صيغة البحث)**

وتتبع صيغة البحث نفس القواعد التي ذكرناها من قبل، مع إمكانيّة تحديد الامتداد.. فمثلا، يمكنك أن تحصل على كلّ الملفّات ذات الامتداد EXE في المجلّد WINNT بالجملة التالية:

**files = Directory.GetFiles("C:\WINNT", "\*.EXE")**

اقرأ عناصر المجلّد GetFileSystemEntries:

هذه الوسيلة هي مجموع الوسيلتين السابقتين معا، حيث تعيد لك مصفوفة بأسماء كلّ العناصر (مجلدات وملفات) الموجودة في المجلّد الذي ترسله لها كمعامل:

**Dim items() As String = Directory.GetFileSystemEntries(مسار المجلّد)**

وطبعا تتوقّع أنّ لديك صورة أخرى من هذه الوسيلة، تمكّنك من البحث عن العناصر التي توافق صيغة معيّنة:

**items = Directory.GetFileSystemEntries(مسار المجلّد, صيغة البحث)**

ولكن كيف تعرف المجلّد من الملفّ في المصفوفة الناتجة؟

هذا المثال يريك إحدى الطرق، وذلك باستخدام الوسيلة Exists لمعرفة إذا كان العنصر مجلدا موجودا أم لا:

**Dim path As String = "c:\Program Files"**

**Dim items() As String = Directory.GetFileSystemEntries(path)**

**Dim itm As String**

**For Each itm In items**

**If Directory.Exists(itm) Then**

**Console.WriteLine("المجلّد " & itm)**

**Else**

**Console.WriteLine("الملف " & itm)**

**End If**

**Next**

**اقرأ جذر المجلّد GetDirectoryRoot:**

تعيد لك هذه الدالّة أوّل مجلّد في مسار المجلّد الذي ترسله لها، والذي لا يشترط أن يكون مسارا موجودا على الجهاز بالفعل:

**Directory.GetDirectoryRoot(مسار المجلّد)**

وناتج هذه الدالة هو نصّ مثل: "C:\" أو "D:\".

**اقرإ المحرّكات المنطقيّة GetLogicalDrives:**

هذه الوسيلة تمنحك مصفوفة نصّيّة بأسماء محرّكات الأقراص الموجودة على القرص الصلب Hard Disk، مثل "C:\" و "D:\".. إلخ.

**اقرإ المجلّد الرئيسيّ GetParent:**

عندما ترسل لهذه الوسيلة مجلّدا كمعامل، فإنّها ترجع لك كائن معلومات المجلّد DirectoryInfo للمجلّد الذي يحتوي مجلّدك.

**Dim parent As DirectoryInfo = Directory.GetParent(مسار المجلّد)**

**نقل Move:**

يمكنك بهذه الوسيلة نقل مجلّد بكلّ ما يحتويه من موقع لآخر، بشرط أن يكون على نفس المحرّك Drive، وطبعا على ألا يكون الموقع الجديد داخل المجلّد المنقول!:

**Directory.Move(المجلّد المنقول, الموقع الجديد)**

وللأسف لا توجد وسيلة مباشرة لنسخ مجلّد بما يحتويه!.. ولفعل ذلك، لا بدّ أن تنشئ تركيبة المجلّد بما يحتويه في موقع جديد، ثم تقوم بنسخ ما بداخل كل منها من ملفات!!

**خليّة الملفّ File Class**

تتيح لك هذه الخليّة التعامل مع الملفّات.. ولن تحتاج لتعريف متغيّر من هذه الخليّة، فكلّ وسائلها مشتركة.

**الوسائل:**

معظم الوسائل واضحة الأسماء، وتأخذ معاملا هو مسار الملفّ الذي تتعامل معه.. ونحن نقصد هنا بكلمة مسار الملفّ، موقعه واسمه وامتداده معا، على غرار: "C:\folder1\file1.txt".

**إنشاء Create:**

تقوم هذه الوسيلة بإنشاء ملفّ جديد وترجع لك كائنا سنتعرّف عليه في هذا الفصل، هو الكائن Stream، الذي يمكنك أن تستخدمه للقراءة والكتابة في هذا الملفّ.

**FStream = File.Create(مسار الملفّ)**

وهناك صيغة أخرى تمكّنك من تحديد حجم الملفّ كالتالي:

**FStream = File.Create(مسار الملفّ, حجمه)**

حاذر: فلو كان الملفّ الذي تنشئه موجودا بالفعل، فسيتمّ استبداله.

ويمكن أن تنطلق العديد من الاستثناءات: إذا كان المسار الذي ذكرته غير موجود، إذا كان اسم الملفّ غير صالح، إذا كان المجلّد الذي تحاول إنشاء الملفّ به يحتاج لتصريح.. إلخ.

**إنشاء نصّ CreateText:**

هذه الوسيلة مماثلة للوسيلة السابقة، ولكنّها تنشئ ملفّا نصّيا، وترجع كائنا من النوع "كاتب البيانات" StreamWriter ليسمح لك بالكتابة في الملفّ، وسنتعرّف عليه بدوره في هذا الفصل.

**Dim SW As StreamWriter = File.CreateText(مسار الملفّ)**

**إضافة نصّ AppendText:**

تمكّنك هذه الوسيلة من فتح ملفّ لإضافة بعض النصوص لنهايته، فإذا لم يكن الملفّ موجودا، يتمّ إنشاؤه أولا.

وتُرجع هذه الوسيلة كائنا من النوع "كاتب البيانات" StreamWriter.

**نسخ Copy:**

تمكّنك هذه الوسيلة من نسخ ملفّ موجود إلى موقع جديد.

فإذا كان هناك ملف بنفس الاسم الذي اخترته للملفّ المنسوخ، فإنّ هذه الوسيلة لن تنجح في عمليّة النسخ.. فإذا ما أردت أن تكتب الملفّ المنسوخ بدلا من الملفّ الموجود، فاستخدم هذه الصيغة:

**File.Copy(الملف الأصليّ, مسار الملفّ الجديد, True)**

وهذه الوسيلة تعمل عبر المحركات Drives المختلفة.. ولكن للأسف لا يمكن نقل أكثر من ملف واحد في كلّ مرّة بهذه الوسيلة!

**حذف Delete:**

استخدم هذه الوسيلة لحذف الملفّ الذي تريده:

**File.Delete(مسار الملفّ)**

وإذا كان الملفّ مفتوحا في هذه اللحظة فلن يمكن حذفه، وسينطلق استثناء.

**موجود Exists:**

مماثلة لتلك الخاصّة بالمجلّد.. وفي المثال التالي نحذف الملفّ إذا ما تحققنا من وجود أولا:

**If File.Exists(المسار) Then**

**File.Delete(المسار)**

**Else**

**MsgBox("الملفّ غير موجود")**

**End If**

**اقرإ السمات GetAttributes:**

تعيد لك هذه الوسيلة كائنا من النوع "سمات الملفّ" FileAttributes، يحتوي على جميع سمات الملفّ المنشود.. ولكن ما هي هذه السمات؟

|  |  |
| --- | --- |
| Archive | الملفّ سجلّ.. معظم ملفّات النظام تمتلك هذه السمة. |
| Compressed | الملفّ مضغوط. |
| Encrypted | الملفّ مشّفر. |
| Hidden | الملفّ مُخفى. |
| NotContentIndexed | الملفّ ليس مرقّما بواسطة خدمة ترقيم محتويات النظام system's content indexing service. |
| Offline | الملفّ غير متّصل عبر الشبكة، وقد لا تكون محتوياته متاحةً طيلة الوقت. |
| ReadOnly | الملفّ للقراءة فقط، ولا يمكن تغيير محتوياته. |
| SparseFile | الملفّ ضئيل المحتوى. |
| System | الملفّ من ملفات النظام. |
| Temporary | الملفّ مؤقّت، وسيمحوه التطبيق الذي أنشأه بعد أن يفرغ منه. |
| Normal | الملفّ عاديّ، لا يمتلك أيّ سمة ممّا سبق. |

وفي الغالب يمتلك الملفّ أكثر من سمة من هذه السمات، فإذا أردت اختبار إذا ما كانت القيمة المعادة من هذه الدالة تحتوى على سمة معيّنة، فقم بإجراء عمليّة And بينها وبين هذه القيمة، كالتالي:

**If File.GetAttributes(fpath) And FileAttributes.ReadOnly Then**

**Console.WriteLine("الملفّ " & fpath & " للقراءة فقط")**

**Else**

**Console.WriteLine("يمكنك أن تكتب في الملفّ " & fpath)**

**End If**

**تغيير سمات الملفّ SetAttributes:**

يمكنك بهذه الوسيلة تغيير سمات الملفّ.. وهي على الصيغة:

**SetAttributes (مسار الملفّ, سمات الملفّ)**

وطبعا ستستخدم مرقّم سمات الملفّ FileAttributes لوضع هذه السمات.. والجملة التالية تريك كيف يمكنك جعل ملف مخفىً ومؤقّتا:

**File.SetAttributes("c:\file1.txt", FileAttributes.Hidden \_**

**And FileAttributes.Temporary)**

**قراءة وتغيير وقت إنشاء الملفّ:**

لديك في هذا الصدد الوسيلتان **GetCreationTime** و **SetCreationTime**.

**قراءة وتغيير وقت آخر قراءة للملفّ:**

لديك في هذا الصدد الوسيلتان **GetLastAccessTime** و **SetLastAccessTime**.

**قراءة وتغيير وقت آخر كتابة في الملفّ:**

لديك في هذا الصدد الوسيلتان **GetLastWriteTime** و **SetLastWriteTime**.

**نقل Move:**

لنقل الملفّ من موضع لآخر.. ويمكنك أن تغيّر اسم الملفّ بهذه الوسيلة، عن طريق نقله إلى نفس مجلّده باسم جديد غير اسمه الحاليّ.

**فتح Open:**

هذه الوسيلة تفتح ملفّا للقراءة منه والكتابة فيه، وترجع كائنا من النوع Stream.

**FStream = File.Open(مسار الملفّ)**

ويمكنك أن تحدّد الطور الذي تفتح فيه الملفّ، بالصيغة التالية:

**FStream = fileObj.Open(مسار الملفّ, طور الفتح)**

ويمكن فتح الملفّ في أيّ من هذه الأطوار:

|  |  |
| --- | --- |
| Append | للإضافة إلى نهاية الملفّ مع المحافظة على المحتويات السابقة. |
| إنشاء Create | تنشئ ملفا جديدا، ولو كان هناك ملف بنفس الاسم يتم استبداله. |
| إنشاء ملفّ جديد CreateNew | تنشئ ملفا جديدا، فإذا كان هناك ملف بنفس الاسم ينطلق استثناء ولا يُنشأ الملفّ. |
| فتح Open | لفتح ملف موجود. |
| فتح أو إنشاء OpenOrCreate | لفتح ملف موجود، فإن لم يكن موجودا يتمّ إنشاؤه. |
| تقليص Truncate | تفتح ملفا موجودا وتمحو محتوياته. |

وهناك صيغة ثالثة، تمنح فيها هذه الدالة معاملا ثالثا، هو طور التعامل مع الملفّ:

**FStream = File.Open(مسار الملفّ, طور الفتح, طور التعامل مع الملفّ)**

حيث يأخذ هذا المعامل قيمة من قيم المرقّم FileAccess، وهي كالتالي:

|  |  |
| --- | --- |
| Read | الملفّ مفتوح للقراءة فقط.. لهذا فإنّ محاولة تغيير محتويات الملفّ تؤدّي لإطلاق استثناء. |
| ReadWrite | الملفّ مفتوح للقراءة والكتابة معا. |
| Write | الملفّ مفتوح للكتابة فقط، دون أن تستطيع قراءة محتوياته. |

وهناك صيغة رابعة من هذه الوسيلة، تمكّنك من تحديد كيفية مشاركة الملفّ بين التطبيقات المختلفة.

**FStream = File.Open(مسار الملفّ, طور الفتح, طور التعامل مع الملفّ,طور المشاركة)**

ويمكنك استخدام أيّ طور مشاركة من هذه الأطوار:

|  |  |
| --- | --- |
| None | الملفّ غير قابل للاشتراك بين أكثر من تطبيق.. فإذا حاول تطبيق آخر فتح الملفّ بينما هو مفتوح في تطبيقك، فلن ينجح في ذلك حتّى تغلقه أنت. |
| Read | عندما تفتح الملفّ، يمكن للتطبيقات الأخرى أن تفتحه للقراءة فقط، دونَ أن تمتلك القدرة على تغيير محتوياته. |
| ReadWrite | يمكن للتطبيقات الأخرى أن تفتح الملفّ للقراءة أو الكتابة. |
| Write | يمكن للتطبيقات الأخرى أن تفتح الملفّ الذي تفتحه حاليا، للكتابة فيه دونَ القراءة منه!! |

**فتح للقراءة OpenRead:**

تفتح هذه الوسيلة ملفّا موجودا للقراءة فقط، وترجع كائنا من النوع Stream.

وهذه الوسيلة مكافئة لفتح الملفّ بواسطة الوسيلة Open في طور القراءة فقط.

**فتح نصّ OpenText:**

تفتح ملفّا نصّيا للقراءة، وترجع كائنا من النوع "قارئ البيانات" StreamReader، والذي يقوم بإجراء التحويلات اللازمة التي تمكّنك من قراءة ملفّات النصّوص المحفوظة بتنسيقات مختلفة مثل ASCII و Unicode و UTF-8.

**فتح للكتابة OpenWrite:**

تفتح هذه الوسيلة ملفّا موجودا للكتابة فيه، وترجع كائنا من النوع StreamWriter.

**خليّة معلومات المجلّد DirectoryInfo Class**

هذه الخليّة تمنحك معلومات عن مجلّد بعينه، لهذا لا بدّ أن تعرّف نسخة من هذه الخليّة وتربطها بمجلّد ما أولا، حتّى يمكنك أن تستخدم خصائصها ووسائلها، والتي بالطبع لن تحتاج لإرسال اسم المجلّد إليها.

ولإنشاء نسخة جديدة من كائن معلومات المجلّد استخدم الصيغة التالية:

**Dim DI As New DirectoryInfo(مسار المجلّد)**

**الوسائل:**

كثير من وسائل هذه الخليّة مماثلة لوسائل خليّة المجلّد، لهذا سنكتفي بشرح الوسيلتين التاليتين فقط:

**إنشاء مجلّد فرعيّ CreateSubdirectory:**

تنشئ مجلدا (أو مجموعة مجلدات متداخلة) داخل المجلّد الذي تمثّله النسخة الحاليّة من الخليّة، وترجع كائن معلومات المجلّد الجديد.

**اقرأ معلومات نظام الملفات GetFileSystemInfos:**

ترجع مصفوفة من كائنات معلومات نظام الملفات FileSystemInfo لكلّ عنصر (مجلّد أو ملفّ) موجود داخل المجلّد الذي تمثّله النسخة الحاليّة من الخليّة.

ولمعرفة نوع العنصر في المصفوفة المعادة، استخدم خاصيّة "سمات" Attributes الخاصّة بالكائن FileSystemInfo.. فمثلا، للتأكّد من أنّ الخانة رقم i في المصفوفة (وليكن اسمها itemsInfo) تحتوي على مجلّد وليس على ملفّ، استخدم الصيغة التالية:

**If itemsInfo(i).Attributes And FileAttributes.Directory Then**

**MsgBox("هذا مجلّد")**

**Else**

**MsgBox("هذا ملفّ")**

**End If**

**خليّة معلومات الملفّ FileInfo Class**

هذه الخليّة مماثلة لخليّة الملفّ، ولكنّها تعمل على ملفّ واحد بعينه، لهذا لا بدّ من تعريف متغير من نوع هذه الخليّة أولا وربطه بالملفّ المعنيّ.

* [**الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20خليّة%20معلومات%20الملفّ\1-%20الخصائص.htm)
* [**الوسائل:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20خليّة%20معلومات%20الملفّ\2-%20الوسائل.htm)

**الخصائص**

**الطول Length:**

ترجع لك حجم الملفّ الذي تمثّله النسخة الحاليّة من هذه الخليّة، بوحدات البيانات Bytes.

**الخصائص الدالة على تاريخ الإنشاء وآخر قراءة وآخر كتابة:**

وقت الإنشاء **CreationTime** ـ وقت آخر قراءة **LastAccessTime** ـ وقت آخر كتابة **LastWriteTime**.

ولسبب لا تدريه، تمّ جعل هذه الإمكانيّات خصائص وليس أزواجا من الوسائل كما في خليّة الملفّ!!.. طبعا هذا أفضل، ولكن لا بدّ أن تتساءل لماذا لم يتمّ توحيد المفاهيم!

**الخصائص الدالة على اسم الملفّ ونوعه:**

اسم الملفّ **Name** ـ الاسم بالكامل (بالمسار) **FullName** ـ امتداد الملفّ **Extension**.

**الوسائل**

**نسخ CopyTo:**

تنسخ الملفّ إلى موقع آخر.

**نقل MoveTo:**

تنقل الملفّ إلى موقع آخر.

**المجلّد Directory:**

ترجع لك هذه الوسيلة كائن DirectoryInfo يحتوي على معلومات المجلّد الذي يحتوي الملفّ.

**اسم المجلّد DirectoryName:**

ترجع لك نصّا عبارة عن اسم المجّلد الذي يحتوي الملفّ.

انظر للمثال التالي لترى كيف نطبع اسم المجلّد بالوسيلتين السابقتين:

**Dim FI As New FileInfo("c:\folder1\folder2\folder3\test.txt")**

**Console.WriteLine(FI.Directory().FullName)**

**' c:\folder1\folder2\folder3**

**Console.WriteLine(FI.DirectoryName())**

**'c:\folder1\folder2\folder3**

**خليّة المسار Path Class**

يمكنك بهذه الخليّة أداء المهامّ البسيطة على مسار الملفّ، مثل استخلاص اسم الملفّ وامتداده منه.. ووسائل هذه الخليّة مشتركة Shared، بحيث يمكن استخدامها من الخليّة الأصليّة مباشرة، وبالتالي فهي تتطلّب منك إرسال المسار لها كمعامل.

* [**الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\4-%20خليّة%20المسار\2-%20الخصائصّ.htm)
* [**الوسائل:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\4-%20خليّة%20المسار\3-%20الوسائل.htm)

**الخصائصّ**

**حرف فاصل المجلدات البديل AltDirectorySeparatorChar:**

في المسار، ستجد أنّ الحرف "\" يفصل بين أسماء المجلّدات.. ويمكنك استخدام هذه الوسيلة لو كان هناك حرف بديل.. فمثلا في ويندوز 2000 سترجع لك هذه الوسيلة الحرف "/".

**حرف فاصل المجلدات DirectorySeparatorChar:**

لقراءة أو تغيير هذا الحرف.

**الحروف غير المسموح بها في المسار InvalidPathChars:**

ترجع لك مصفوفة من الحروف، تحتوى على تلك الرموز التي لا يمكنك استخدامها في المسار.. غنيّ عن الذكر أن هذه الخاصيّة لا تحتاج لمعاملات.

**فاصل المسارات PathSeparator:**

أحيانا يتمّ كتابة أكثر من مسار في نصّ واحد، بحيث تفصل بينها علامة معيّنة، مثل ";".

وهذه الخاصيّة ترجع هذا الحرف الفاصل.. وبالمثل تعيد لك الخاصيّة فاصل"اسمالمحرّك" **VolumeSeparatorChar** الرمز الذي يفصل اسم المحرّك عن باقي المسار.

**الوسائل**

**تغيير الامتداد ChangeExtension:**

تغيّر هذه الوسيلة امتداد الملفّ، وترجع لك الامتداد الجديد.

**newExtension = Path.ChangeExtension(مسار الملفّ, الامتداد الجديد)**

فإذا أردت محو الامتداد نهائيّا، فأرسل في المعامل الثاني نصّا فارغا.

**دمج Combine:**

استخدم هذه الوسيلة لدمج مسار مجلّد مع اسم ملفّ، كالتالي:

**newPath = Path.Combine(مسار المجلّد, اسم الملفّ)**

فمثلا: الجملة التالية تكوّن المسار "c:\textFiles\test.txt":

**Path.Combine("c:\textFiles", "test.txt")**

**اقرإ اسم المجلّد GetDirectoryName:**

تعيد لك هذه الوسيلة مسار المجلّد الذي يحتوي على الملفّ.. فمثلا الجملة:

**Path.GetDirectoryName("C:\folder1\folder2\folder3\Test.txt")**

ستعيد المسار:

C:\folder1\folder2\folder3

ولقراءة اسم الملفّ استخدم الوسيلة GetFileName، ولقراءته بدون الامتداد استخدم الوسيلة **GetFileNameWithoutExtension**.

**اقرإ المسار الكامل GetFullPath:**

سيبدو لك للوهلة الأولى أنّ هذه الوسيلة عبيطة وبلا فائدة، فهي ستعيد لك المسار الذي ترسله لها كمعامل!!.. ولكنّ استخدامها الحقيقيّ يكون عندما ترسل لها مسارا نسبيا، فتعيد لك المسار الكلّيّ بما في ذلك الجزء الخاصّ بالمسار الحاليّ.

فمثلا الجملة التالية:

**Path.GetFullPath("Test.txt")**

ستعيد لك المسار الحاليّ مضافا إليه اسم الملفّ وليكن:

C:\f1\f2\f3\f4\Test.txt

وهذا لا يشترط أن يكون الملفّ موجودا بالفعل.

كما أنّ بإمكانك استخدام جملة كالتالية:

**Path.GetFullPath("..\..\Test.txt")**

وفي هذه الحالة سيتمّ إضافة اسم الملفّ لمسار المجلّد الأعلى من المجلّد الحالي بمستويين.. أيّ أنّ الناتج سيكون:

C:\f1\f2\Test.txt

**اقرإ المسار المؤقّت GetTempPath:**

تعيد لك مسارا يمكنك أن تستخدمه في تخزين الملفات المؤقّتة، بحيث يمكن للنظام حذف هذا المجلّد بمحتوياته عندما يحتاج لمساحة.

**اقرإ الملفّ المؤقّت GetTempFile:**

تنشئ ملفا احتياطيا فارغا وتعيد لك اسمه.

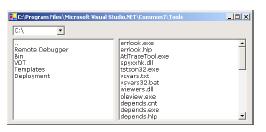
**له امتداد HasExtension:**

ترجع لك True إذا كان للملف امتداد.

**مشروع "المتصفّح الخاصّ" CustomExplorer:**

الآن سننشئ تطبيقا نستخدم فيه معظم ما تعلّمناه في هذا الفصل حتّى الآن.

كلّ ما نريد فعله، هو عرض قائمة بأسماء أجزاء القرص الصلب (سنسمّي هذه القائمة DrivesList)، فإذا اخترت واحدا منها، تظهر لك كلّ المجلّدات التي يحتوي عليها في قائمة المجلّدات (وسنسميها FoldersList)، فإذا اخترت مجلّدا منها، تتغيّر محتويات قائمة المجلّدات لتعرض مجلّداته الفرعيّة، كما تظهر لك ملفاته في قائمة الملفات (وسنسمّيها FilesList).. ولكي تعود إلى المجلّد الأصليّ، يمكنك أن تضغط أوّل عنصر في قائمة المجلّدات، والذي سنجعله الرمز التالي ".."، ليدّل على أنّ القائمة تحتوي على مجلّدات فرعيّة.



وعند تشغيل البرنامج، سنستدعى الإجراء ShowAllDrives لملء قائمة المحرّكات بأسماء أجزاء القرص الصلب.

**Sub ShowAllDrives()**

**Dim drives() As String**

**drives = Directory.GetLogicalDrives()**

**' تأكّد من أنّ القائمة فارغة**

**DrivesList.Items.Clear()**

**' ملء القائمة بعناصر المصفوفة**

**Dim aDrive As String**

**For Each aDrive In drives**

**DrivesList.Items.Add(aDrive)**

**Next**

**End Sub**

ونظرّا لأنّ مؤلّف المرجع الأصليّ ما زال يفكّر بعقليّة VB6، سأريك كيف تكتب الإجراء السابق في سطرين اثنين فحسب!!:

**Sub ShowAllDrives()**

**DrivesList.Items.Clear()**

**DrivesList.Items.AddRange(Directory.GetLogicalDrives)**

**End Sub**

ونظرًا لأنّ قائمة أسماء المحرّكات يجب أن تمتلئ عند تشغيل البرنامج، فسنقوم باستدعاء الإجراء ShowAllDrives عند تحميل النموذج:

**Private Sub Form1\_Load(ByVal sender As System.Object, \_**

**e As EventArgs) Handles MyBase.Load**

**ShowAllDrives()**

**' A:\ سنحدد العنصر الثاني في القائمة، لأنّ العنصر الأوّل هو القرص المرن**

**DrivesList.SelectedIndex = 1**

**' سنجعل عنوان النموذج هو المجلّد الحالي**

**Me.Text = Directory.GetCurrentDirectory**

**End Sub**

وعند ضغط أيّ محرّك Drive في قائمة المحرّكات، سنستدعي الإجراء ShowFoldersInDrive لعرض المجلّدات الموجودة على هذا الجزء من القرص الصلب:

**Sub ShowFoldersInDrive(ByVal drive As String)**

**Dim folders() As String**

**Try**

**folders = Directory.GetDirectories(drive)**

**Catch exception As Exception**

**MsgBox(exception.Message)**

**Exit Sub**

**End Try**

**FoldersList.Items.Clear()**

**Dim fldr As String**

**Dim DI As DirectoryInfo**

**' لاحظ أنّنا سنضع في القائمة كائن معلومات المجلّد حتّى نستخدمه بعد ذلك**

**' في معرفة المجلدات الفرعيّة**

**For Each fldr In folders**

**DI = New DirectoryInfo(fldr)**

**FoldersList.Items.Add(DI.Name)**

**Next**

**' تغيير المجلّد الحالي إلى اسم المحرّك**

**Directory.SetCurrentDirectory(drive)**

**Me.Text = Directory.GetCurrentDirectory**

**End Sub**

وأيضا.... يمكنك أن تكتب الجزء الذي يملأ قائمة المجلّدات بكائنات معلومات المجلّد في سطرين فحسب، بدلا من تلك الجملة التكراريةّ العقيمة .. تصوّر!.. ها هما تان تلك الجماتان:

**Dim DI As New DirectoryInfo(drive)**

**FoldersList.Items.AddRange(DI.GetDirectories)**

حيث استخدمنا كائن معلومات المجلّد DirectoryInfo في الحصول على مصفوفة كائنات معلومات المجلّد، بدلا من استخدام كائن المجلّد Directory للحصول على أسماء المجلّدات الفرعيّة ثم تعريف كائن معلومات المجلّد لكلّ منها، فهذا أشبه بتعبير: من أين أذنك يا جحا؟!!

ولا يتوقّف الأمر عند هذا الحدّ، بل يصل إلى أنّك تستطيع اختزال الجملتين في جملة واحدة كالتالي:

**FoldersList.Items.AddRange(New DirectoryInfo(drive).GetDirectories)**

وبهذا بدلا من كتابة 6 جمل طويلة تحتوي على جملة تكرارية عقيمة وبطيئة، يمكنك كتابة جملة واحدة أسرع وأكفأ.

بقيَ أن نستدعيَ هذا الإجراء، وكما قلنا، سنستدعيه في حدث تغيّر العنصر المحّدد في قائمة المحرّكات:

**Private Sub DrivesList\_SelectedIndexChanged(sender As Object, \_**

**e As EventArgs) Handles DrivesList.SelectedIndexChanged**

**ShowFoldersInDrive(DrivesList.Text)**

**End Sub**

والآن تعال نرى كيف نملأ قائمة الملفّات.

عندما تضغط أحد المجلّدات، يجب أن يحدث ما يلي:

- إفراغ قائمة المجلّدات.

- جعل أوّل عنصر في قائمة المجلّدات هو الرمز ".."، الذي عند ضغطه يتمّ الرجوع إلى المجلّد الأصليّ.

- ملء قائمة المجلّدات بالمجلّدات الفرعيّة التي يحتويها المجلّد الذي تمّ ضغطه.

- ملء قائمة الملفّات بالملفّات التي يحتويها المجلّد الذي تمّ ضغطه.

هذا هو ما فعلناه في حدث تغيّر العنصر المحّدد في قائمة المجلّدات:

**Private Sub FoldersList\_SelectedIndexChanged(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles FoldersList.SelectedIndexChanged**

**Dim DI As DirectoryInfo**

**Select Case FoldersList.Text**

**Case ""**

**MsgBox("اختر مجلّدا لعرض محتوياته")**

**Exit Sub**

**Case ".."**

**' الصعود مستوى واحد لأعلى، إلى المجلّد الرئيسيّ الذي يحتوي المجلّدات الحاليّة**

**Directory.SetCurrentDirectory("..")**

**Case Else**

**' عرض المجلّدات الفرعيّة للمجلّد المضغوط**

**Directory.SetCurrentDirectory(FoldersList.Text)**

**Me.Text = Directory.GetCurrentDirectory**

**End Select**

**' احتفظ باسم المجلّد المضغوط في متغيّر لأنّ محتويات القائمة ستتغيّر**

**Dim selectedFolder As String = FoldersList.Text**

**' تذكّر أنّنا جعلنا المجلّد المضغوط هو المجلّد الحالي**

**Dim Dr As String = Directory.GetCurrentDirectory**

**' اعرض مجلّدات المجلّد الحالي**

**ShowFolders(Dr)**

**' إذا كان المجلّد الحالي هو اسم المحرّك نفسه، فلن نحتاج لوضع الرمز ".."**

**If Dr <> Directory.GetDirectoryRoot(selectedFolder) Then \_**

**FoldersList.Items.Insert(0, "..")**

**' اعرض ملفّات المجلّد الحالي**

**ShowFilesInFolder()**

**End Sub**

وهذا هو الإجراء الذي يعرض ملفّات المجلّد الحاليّ في قائمة الملفّات:

**Sub ShowFilesInFolder()**

**FilesList.Items.Clear()**

**FilesList.Items.AddRange(Directory.GetFiles(\_**

**Directory.GetCurrentDirectory))**

**End Sub**

لقد عدّلت الكود الأصليّ لمؤلّف الكتاب، فصارت القائمة تمتلئ في سطر واحد، وهذه الطريقة أسرع بكثير من الجملة التكراريّة التي تضيف عنصرا بعد عنصر، والتي أنصحك باجتنابها ما لم يكن هناك مناص.

ولو كانت الجملة التي تملأ القائمة معقّدة، فسأسردها لك في عدّة جمل مكافئة كالتالي:

**Dim Dr As String = Directory.GetCurrentDirectory**

**Dim F() As String = Directory.GetFiles (Dr))**

**FilesList.Items.AddRange(F)**

تلاحظ أنّنا نضع في مصفوفة الملفّات أسماء الملفّات فقط، ولو تطلّب الأمر وضع كائنات معلومات الملفّات لاستخدامها في شيء آخر، فيمكنك استخدام الجملة التالية:

**FilesList.Items.AddRange(New \_**

**DirectoryInfo (Directory.GetCurrentDirectory).GetFiles())**

والآن سنعرض للمستخدم رسالة تحتوي على سمات الملفّ كلّما ضغط ملفا في القائمة.

هذا هو الكود الذي يُنفّذ عند تغيير العنصر المحدّد في قائمة الملفّات:

**Private Sub FilesList\_SelectedIndexChanged(sender As Object, \_**

**e As EventArgs) Handles FilesList.SelectedIndexChanged**

**Dim FI As New FileInfo(FilesList.Text)**

**Dim Msg As New System.Text.StringBuilder()**

**Msg.Append("اسم الملفّ: " & FI.Name & vbCrLf)**

**Msg.Append("حجمه: " & FI.Length.ToString & vbCrLf)**

**Msg.Append("امتداده: " & FI.Extension & vbCrLf)**

**Msg.Append("وقت إنشائه: " & FI.CreationTime.ToString & vbCrLf)**

**Msg.Append("وقت آخر قراءة: ")**

**Msg.Append(FI.LastAccessTime.ToShortDateString)**

**MsgBox(Msg.ToString, MsgBoxStyle.Information Or \_**

**MsgBoxStyle.OKOnly Or MsgBoxStyle.MsgBoxRtlReading \_**

**Or MsgBoxStyle.MsgBoxRight, "")**

**End Sub**

تلاحظ أنّنا أرسلنا معاملا ثانيا لدالة الرسالة MsgBox، وذلك لأنّنا أردنا أن تظهر الرسالة من اليمين لليسار MsgBoxRtlReading مع محاذاتها لليمين MsgBoxRight.

**المرور عبر كل المجلّدات الفرعية لمجلّد:**

لن تستطيع التعامل مع المجلّدات والملفّات بكفاءة، ما لم تستطع المرور عبر التركيب الشجريّ للمجلّدات.. إذن فستحتاج هنا لكتابة إجراء ارتداديّ Recursive Procedure.. لا تقلق، فمع استخدام المندوب Delegate سيصبح هذا الإجراء بالصورة البسيطة التالية:

**Delegate Sub FolderAction(ByVal Dir As String)**

**Sub ScanFolder(ByVal currDir As String, Action As FolderAction)**

**Dim Dir As String**

**' استدع الإجراء الذي تريده للتعامل مع المجلّد الحالي**

**Action(CurrDir)**

**' انتقل للمجلّدات الفرعيّة للمجلّد الحالي**

**' لاحظ أنّ شرط التوقّف هو انتهاء الجملة التكراريّة**

**For Each Dir In Directory.GetDirectories(currDir)**

**ScanFolder(Dir)**

**Next**

**End Sub**

الآن يمكنك استخدام الإجراء السابق في أيّ تطبيق بدون أيّ تعديل.. كلّ ما ستحتاج إليه هو كتابة الإجراء الذي يتعامل مع المجلّدات، وإرساله إلى الإجراء ScanFolder.. افترض أنّنا نريد حساب عدد المجلّدات التي يحتوى عليها مجلّد ما.. سنكتب الإجراء DirCount لفعل ذلك:

**Sub DirCount(ByVal Dir As String)**

**Count += 1**

**End Sub**

حيث إنّ المتغيّر Count هو متغيّر عامّ للنموذج، نعرّفه في بداية النموذج كالتالي:

**Dim Count As Integer**

والآن تعال نجرّب حساب عدد المجلّدات الموجودة على المحرّك "C:\".. لاحظ أنّ عدد المجلّدات التي يحسبها الإجراء DirCount يتضمّن المجلّد الرئيسيّ نفسه، لذلك لا بدّ أن تطرح منه 1.. أو كبديل، ضع في المتغيّر Count الرقم -1 قبل بدء الإجراء:

**Count = -1**

**Scanfolder("C:\", AddressOf DirCount)**

**MsgBox (Count)**

ويمكنك كتابة عشرات الإجراءات الأخرى، لحساب حجم المجلّد وملفّاته، أو لحذف الملفّات، أو للبحث عن ملفّ معيّن... إلخ.. المهمّ أن يكون لكلّ هذه الإجراءات معامل واحد نصّيّ String حتّى تتّفق مع صيغة المندوب Delegate الذي عرّفناه.

**استخدام كائن نظام الملفّات FileSystemObject**

* + - [**ما هو كائن نظام الملفّات؟:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\_ما%20هو%20كائن%20نظام%20الملفّات.htm)
    - [**وسائل التعامل مع المحرّكات Drives:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\0-%20وسائل%20التعامل%20مع%20المحرّكات.htm)
    - [**خصائص كائن المحرّك Drive:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\1-%20خصائص%20كائن%20المحرّك.htm)
    - [**وسائل التعامل مع المجلّدات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\2-%20وسائل%20التعامل%20مع%20المجلّدات.htm)
    - [**وسائل التعامل مع الملفّات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\3-%20وسائل%20التعامل%20مع%20الملفّات.htm)
    - [**خصائص ووسائل كائن المجلّد Folder و كائن الملفّ File:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\4-%20خصائص%20ووسائل%20كائن%20المجلّد%20و%20كائن%20الملفّ.htm)
    - [**وسائل التعامل مع المسارات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20كائن%20نظام%20الملفّات\5-%20وسائل%20التعامل%20مع%20المسارات.htm)

**ما هو كائن نظام الملفّات؟:**

رغمَ كلِّ الإمكانيّاتِ التي تمنحُها لكَ الخلايا السابقة للتعامل مع المجلّدات والملفّات، إلا إنّك ما زلتَ تحتاجُ للمزيد.. فمثلا، لا يمكنك حتّى الآن أن تعرف حجم أحد المحرّكات Drives أو المساحة التي ما تزالُ فارغةً عليه، أو الاسم الذي سمّاه به المستخدم، كما لا تستطيع أن تعرف ما هو محرّك الأسطوانات CD Room.. إلخ.

هنا يكونُ الحلّ عن طريق استخدام الأداة FileSystemObject، وهي لا تنتمي لإطار العمل .Net Framework، ولكنّها إحدى أدوات VB6 التي تنتمي لتقنية COM.. لهذا سنبدأ بإضافةِ مرجع لها في التطبيق.

في متصفّح المشاريع Solution Explorer اضغط العنصر References بزرّ الفأرةِ الأيمن، ومن القائمة الموضعيّة اضغط الأمر Add Reference.. في مربّع الحوارِ الذي سيظهر، اضغط شريط COM، ومن قائمةِ الأدوات اعثر على الأداة Microsoft Scripting Runtime.. انقرها مرّتينِ بالفأرة.. سيظهر اسمها في القائمة السفليّة.. اضغط OK لإغلاقِ مربّع الحوار.

ستجد أنّ الاسم Scripting قد أضيفَ لقائمة المراجع.. الآن يمكنك أن تعرّف متغيّرا من هذه الأداة كالتالي:

**Dim Fs As New Scripting.FileSystemObject()**

وتحتوي هذه الأداة على العديد من الخصائص والوسائل، كثيرٌ منها لن يقدّم لنا جديدا، حيث تتشابه وظيفته مع وظائف الخلايا التي سبق لنا التعرّف عليها، بل إنّ بعض هذه الوظائف يمكن أداؤه بأكثر من طريقة من داخل أداة نظام الملفّات.

**وسائل التعامل مع المحرّكات Drives:**

**المحرّك موجود DriveExists:**

ترجع هذه الوسيلة True إذا كان الحرف المرسل لها كمعامل هو أحد المحرّكات الموجودة على القرص الصلب.. والجمل التالية تعيد كلّها القيمة True:

**MsgBox(Fs.DriveExists("C"))**

**MsgBox(Fs.DriveExists("C:"))**

**MsgBox(Fs.DriveExists("C:\"))**

**اقرإ المحرّك GetDrive:**

أرسل لهذه الوسيلة حرف المحرّك Drive Letter، لتحصل على كائن من النوع Scripting.Drive، يحتوى على خصائص هذا المحرّك.

**Dim Dr As Scripting.Drive = Fs.GetDrive("C:\")**

ويمكنك عمل نفس الشيء باستخدام خاصيّة Drives، وهي مجموعة Collection، تحتوى على خاصّيّتي العنصر Item والعدد Count:

**Dim Dr As Scripting.Drive = Fs.Drives("C:\")**

**خصائص كائن المحرّك Drive:**

**المساحة المتاحة AvailableSpace:**

تعيد هذه الخاصيّة المساحة الفارغة على المحرّك بوحدة Byte.

الطريف أنّ هناك خاصيّة أخرى تقوم بنفس الوظيفة، هي خاصيّة FreeSpace.

**ملحوظة:**

لكي تحوّل هذه المساحة للميجا، يمكنك أن تقسم على مليون.. وإن كان هذا غير دقيق، فالكيلو بايت يساوي: 2 10 = 1024 بايت ، والميجا بايت = 2 20 = 1048576 بايت.. إذن فالأدقّ أن تقسم على هذا الرقم الأخير.. لاحظ كذلك أنّ الجيجا بايت تساوي: 2 30 = 1073741824.

والجملة التالية تعرض المساحة الفارغة على المحرّك C: :

**MsgBox(Fs.GetDrive("C:\").AvailableSpace/(2^20))**

**المساحة الكلّيّة TotalSize:**

تعيد لك هذه الوسيلة مساحة المحرّك الكلّيّة، أيضا بالبايت.

**حرف المحرّك DriveLetter:**

تعيد لك هذه الخاصيّة الحرف المناظر للمحرّك، مثل C، D.. إلخ.

وهناك خاصيّة أخرى مماثلة، هي خاصيّة المسار Path، إلا إنّها تعيد نتيجةً على شاكلة C:، D:... إلخ.

**اسم المحرّك VolumeName:**

تعيد لك هذه الخاصيّة النصّ الذي وضعه المستخدم كعنوانٍ للمحرّك.

**نوع المحرّك DriveType:**

استخدم هذه الخاصيّة لمعرفة نوع المحرّك، حيث تعيد واحدةً من قيم المرقّم Scripting.DriveTypeConst، وهي:

|  |  |
| --- | --- |
| CDRom | محرّك أسطوانات. |
| Fixed | ثابت.. بمعنى أنّه جزءٌ من القرص الصُّلب. |
| Remote | محرّك موجود على جهاز آخر في الشبكة. |
| Removable | محرّك قابل للإزالة.. مثل القرص المرن. |
| UnknownType | نوع المحرّك غير معروف. |

والمثال التالي يعرض اسم محرّك الأسطوانة على جهازك:

**Dim Fs As New Scripting.FileSystemObject()**

**Dim Drs As IEnumerator = Fs.Drives.GetEnumerator**

**Do While Drs.MoveNext**

**Dim Dr As Scripting.Drive = CType(Drs.Current, Scripting.Drive)**

**If Dr.DriveType = Scripting.DriveTypeConst.CDRom Then**

**MsgBox("محرّك الأسطوانة هو " & Dr.DriveLetter)**

**End If**

**Loop**

**نظام الملفّات FileSystem:**

تعيد لك هذه الوسيلة نظام الملفّات على هذا المحرّك: FAT16، FAT32.. إلخ.

**المحرّك جاهز IsReady:**

استخدم هذه الخاصيّة للتأكّد من أنّ هناك أسطوانة في محرّك الأسطوانات، أو قرص في محرّك الأقراص المرنة، حيث تعيد هذه الخاصيّة True في هذه الحالة.

**الرقم المسلسل SerialNumber:**

تعيد لك هذه الوسيلة الرقم المسلسل الذي منحه النظام للمحرّك.. ولا أدرى فيم سيفيدك هذا!!

**المجلّد الرئيسيّ RootFolder:**

أعرفُ أنّك ستتعجّب من اسم هذه الخاصيّة، إذ يبدو أنّها بلا فائدة، فالمجلّد الرئيسيّ للمحرّك هو المجلّد نفسه!

بل هناك فائدة جمّة، فهذه الوسيلة تُعيد كائنا من النوع Scripting.Folder، وهو يحتوي على خصائص المجلّد.. مثلا: هل هو للقراءة فقط أم مخفىً.... إلخ.

وسنتعرّف على كائن المجلّد لاحقا.

**وسائل التعامل مع المجلّدات:**

تمنحك أداة نظام الملفّات الوسائل التالية للتعامل مع المجلّد:

**إنشاء مجلّد CreateFolder:**

تُنشئ مجلّدًا جديدا في المسار الذي تريده.. وتعيد هذه الوسيلة كائنا من النوع Scripting.Folder، يحتوى على خصائص المجلّد الجديد.. وسنتعرّف على هذا الكائن بعد قليل.

**Dim Dir As Scripting.Folder** = **Fs.CreateFolder (مسار المجلّد)**

**نسخ مجلّد CopyFolder:**

تنسخ المجلّد بما يحتويه من موضعٍ لآخر:

**Fs.CopyFolder(مسار المجلّد, المسار الجديد)**

ولو كان هناك مجلّد بنفس الاسم في المسار الجديد، فسيتمّ استبدال الملفّات المتشابهة في المجلّدين، مع الإبقاء على الملفّات المختلفة بدون حذف.. فإذا لم تكن تريد إجراء عمليّة النقل في حالة وجود مجلّد بنفس الاسم، فاستخدم الصيغة التالية:

**Fs.CopyFolder(مسار المجلّد, المسار الجديد, False)**

**نقل مجلّد MoveFolder:**

تقوم بنقل المجلّد بكلّ ما يحتويه من موضع لآخر، مع إمكانيّة تغيير اسمه:

**Fs.MoveFolder(مسار المجلّد, المسار الجديد)**

**حذف مجلّد DeleteFolder:**

مماثلة للوسيلة السابقة، ولكنّها تقوم بحذف المجلّد بما يحتويه.

**المجلّد موجود FolderExists:**

تعيد True إذا كان المجلّد موجودا.

**اقرإ أحد المجلّدات الخاصّة GetSpecialFolder:**

يمكنك باستخدام هذه الوسيلة أن تعرف مجلّد الويندوز أو مجلّد النظام System أو المجلّد المؤقّت Temperary، حيث تعيد هذه الوسيلة كائنا من النوع Scripting.Folder، يحتوى على خصائص هذا المجلّد:

**Dr = Fs.GetSpecialFolder(المجلّد الخاصّ)**

حيث يأخذ المعامل أحد قيم المرقّم Scripting.SpecialFolderConst التالية:

|  |  |
| --- | --- |
| SystemFolder | مجلّد النظام. |
| TemporaryFolder | المجلّد المؤقّت. |
| WindowsFolder | مجلّد الويندوز. |

**اقرإ المجلّد GetFolder:**

أرسل لهذه الوسيلة مسار المجلّد، لتحصل على كائن من النوع Scripting.Folder، يحتوى على خصائص هذا المجلّد:

**Dim Dir As Scripting.Folder = Fs.GetFolder("C:\1.txt")**

**وسائل التعامل مع الملفّات:**

تمنحك أداة نظام الملفّات الوسائل التالية للتعامل مع الملفّات:

**نسخ ملفّ CopyFile:**

استخدم هذه الوسيلة لنسخ ملفّ إلى موضعٍ آخر (مع إمكان تغيير اسمه):

**Fs.CopyFile(الملفّ الأصليّ, المسار الجديد)**

ولو كان هناك ملفّ بنفس الاسم في المسار الجديد، فسيتمّ استبدال الملفّ المنقول به.. فإذا لم تكن تريد فعل ذلك، فاستخدم الصيغة التالية:

**Fs.CopyFile(الملفّ الأصليّ, المسار الجديد, False)**

**نقل ملفّ MoveFile:**

استخدم هذه الوسيلة لنقل الملفّ من مسار لآخر، مع إمكانيّة تغيير اسمه:

**Fs.MoveFile(مسار الملفّ واسمه, المسار الجديد)**

**حذف ملفّ DeleteFile:**

استخدم الصيغة التالية لحذف الملفّ:

**Fs.DeleteFile(مسار واسم الملفّ)**

ولو كان الملفّ للقراءة فقط ReadOnly فلن يتمّ حذفه.. فإذا أردت أن تجبر VB على حذف، فاستخدم الصيغة التالية:

**Fs.DeleteFile(مسار واسم الملفّ, True)**

**الملفّ موجود FileExists:**

تعيد True إذا كان الملفّ موجودا.

**اقرإ إصدار الملفّ GetFileVersion:**

تعيد هذه الوسيلة رقم الإصدار للملفّ.. فمثلا، يمكنك استخدام الجملة التالية لمعرفة رقم إصدار تطبيق Microsoft Word:

**MsgBox(Fs.GetFileVersion( \_**

**"C:\Program Files\Microsoft Office\Office10\WINWORD.EXE"))**

**اقرإ اسما مؤقّتا GetTempName:**

استخدم هذه الوسيلة للحصول على اسم ملفّ عشوائيّ (له الامتداد .tmp) لتستخدمه كملفّ مؤّقت.

**MsgBox(Fs.GetTempName())**

**اقرإ الملفّ GetFile:**

أرسل لهذه الوسيلة مسار الملفّ واسمه، لتحصل على كائن من النوع Scripting.File، يحتوى على خصائص هذا الملفّ:

**Dim F As Scripting.File = Fs.GetFile ("C:\1.txt")**

**خصائص ووسائل كائن المجلّد Folder و كائن الملفّ File:**

يمتلك كائنا المجلّد والملفّ ـ اللذين تعيدهما وسائل كائن نظام الملفّات السابقة ـ نفس الخصائص والوسائل.. وهي:

**الاسم Name:**

تعيد هذه الخاصيّة اسم المجلّد (بدون المسار).

**الاسم القصير ShortName:**

تعيد لك هذه الخاصيّة الاسم الذي يستخدمه نظام DOS للتعامل مع الملفّ.. فبينما يسمح الويندوز بأن يصل طول الملفّ إلى 255 حرفا، لا يسمح الدوس بأكثر من 8 حروف فقط!.. ولو جرّبت المثال التالي، فستظهر لك رسالة تعرض الاسم "DOCUME~1":

**MsgBox(Fs.GetFolder("C:\Documents and Settings").ShortName)**

**المسار Path:**

تعيد هذه الخاصيّة مسار المجلّد كاملا، بما في ذلك اسم المجلّد نفسه.

**المسار القصير ShortPath:**

تعيد لك هذه الخاصيّة مسار المجلّد، مع عرض الأسماء القصيرة Short Names لكلّ المجلّدات الموجودة في هذا المسار.

**هو المجلّد الرئيسيّ IsRootFolder:**

تعيد هذه الخاصيّة True إذا لم يكن هذا المجلّد موجودا داخل مجلّد آخر.. باختصار: إذا كان محرّكا Drive.

**المجلّد الحاوي ParentFolder:**

تعيد هذه الوسيلة المجلّد الذي يحتوي على المجلّد الذي تمثّله النسخة الحاليّة من المتغيّر.

**الحجم Size:**

تعيد هذه الخاصيّة حجم المجلّد بالبايت.

**السمات Attributes:**

تعيد هذه الخاصيّة سمات المجلّد، التي هي قيمة أو أكثر (عن طريق المعامل Or) من قيم المرقّم Scripting.FileAttribute، وهي:

|  |  |
| --- | --- |
| Alias | مستعار. |
| Archive | أرشيف. |
| Compressed | مضغوط. |
| Directory | مجلّد. |
| Hidden | خفيّ. |
| Normal | عادي. |
| ReadOnly | للقراءة فقط. |
| System | خاصّ بالنظام. |
| Volume | محرّك. |

**المحرّك Drive:**

تعيد لك هذه الخاصيّة كائنا من النوع Scripting.Drive، يحتوي على خصائص المحرّك الذي ينتمي إليه المجلّد.

**الملفّات Files:**

هذه الخاصيّة عبارة عن مجموعة Collection، تحتوى على كائنات من النوع Scripting.File، تمثّل الملفّات الموجودة في المجلّد.. ولا يمكن التعامل مع عناصر هذه المجموعة بالأرقام، ولكن عن طريق استخدام اسم الملفّ (بدون المسار)، كالتالي:

**Dim F As Scripting.File = Dr.Files("1.txt")**

ولكن كيف نستخدم اسم الملفّ ونحن في الغالب نريد أن نعرفه؟!

هنا يمكن استخدام الوسيلة GetEnumerator للمرور عبر كلّ ملفّات المجلّد كالتالي:

**Dim Fs As New Scripting.FileSystemObject()**

**Dim Fls As IEnumerator = Fs.GetFolder("C:\").Files.GetEnumerator**

**Do While Fls.MoveNext**

**Dim F As Scripting.File = CType(Fls.Current, Scripting.File)**

**MsgBox(F.Path)**

**Loop**

**المجلّدات الفرعيّة SubFolders:**

هذه الخاصيّة عبارة عن مجموعة Collection، تحتوي على كلّ المجلّدات الفرعيّة الموجودة داخل هذا المجلّد.. ويمكنك استخدام الوسيلة GetEnumerator للمرور عبر كلّ هذه المجلّدات.. كما يمكنك استخدام الوسيلة Add لإضافة مجلّد فرعيٍّ جديد لهذا المجلّد.

**النوع Type:**

تعيد لك هذه الخاصيّة نصّا يمثّل نوع المجلّد.. هل هو محرّك أم مجلّد ... إلخ.

**وطبعا الأعضاء التالية لن تكون غريبة عليك:**

نسخ **Copy** ـ نقل **Move** ـ حذف **Delete** ـ تاريخ الإنشاء **DateCreated** ـ تاريخ آخر تعامل **DateLastAccessed** ـ تاريخ آخر تعديل **DateLastModified**.

**وسائل التعامل مع المسارات:**

تمنحك أداة نظام الملفّات الوسائل التالية للتعامل مع المسارات.. ولا يشترط أن تكون هذه المسارات موجودةً بالفعلِ على الجهاز، فهذه مجرّد عمليّات نصّيّة:

**إنشاء مسار BuildPath:**

تقوم هذه الوسيلة بإنشاء المسار الكامل لملفّ:

**MsgBox(Fs.BuildPath("C:\", "1.txt")) ' C:\1.Text**

**اقرإِ المسار المجرّد GetAbsolutePathName:**

استخدم هذه الوسيلة للحصول على المسار الكامل للمسار النسبيّ الذي ترسله كمعامل.. فمثلا، الجملة التالية:

**MsgBox(Fs.GetAbsolutePathName("1.txt"))**

تعرض الرسالة: "E:\VS.Net\Project1\1.txt"، حيث إنّ هذا هو مسار المجلّد الحالي Current Folder.. وهو طبعا سيختلف على جهازك، حيث سيكون في الغالب، هو اسم مسار المشروع الذي تنفّذ فيه هذه الجملة.

أمّا الجملة التالية:

**MsgBox(Fs.GetAbsolutePathName("C:\2\..\1.txt"))**

فهي تعرض الرسالة: "C:\2\1.txt".

**اقرإ المجلّد الرئيسيّ GetParentFolderName:**

استخدم هذه الوسيلة لمعرفة المسار الذي يوجد بداخله مجلّد أو ملفّ:

**MsgBox(Fs.GetFileName("C:\2\1.txt")) ' C:\2\**

**اقرإ اسم المحرّك GetDriveName:**

استخدم هذه الوسيلة لاستخلاص اسم المحرّك من المسار.. فمثلا، تعرض الجملة التالية الرسالة "C:":

**MsgBox(Fs.GetDriveName("C:\1.txt"))**

**اقرإِ الاسم الأساسيّ GetBaseName:**

استخدم هذه الوسيلة لاستخلاص اسم الملفّ أو المجلّد من المسار.. وكمثال: كلّ الجمل التالية تعرض الرسالة: "1":

**MsgBox(Fs.GetBaseName("C:\1.txt"))**

**MsgBox(Fs.GetBaseName("1.txt"))**

**MsgBox(Fs.GetBaseName("C:\1 "))**

**اقرإ اسم الملف GetFileName:**

استخدم هذه الوسيلة لاستخلاص اسم الملفّ وامتداده من المسار.. فمثلا، تعرض الجملة التالية الرسالة "1.txt":

**MsgBox(Fs.GetFileName("C:\1.txt"))**

**اقرإ الامتداد GetExtensionName:**

استخدم هذه الوسيلة لاستخلاص الامتداد من المسار.. فمثلا، تعرض الجملة التالية الرسالة "txt":

**MsgBox(Fs.GetExtensionName("C:\1.txt"))**

**استخدام جمل ودوالّ VB6**

رأينا إلى الآن طريقتين للتعامل مع المجلّداتِ والملفّات.. ما زالت هناك طريقةٌ ثالثة، تلك هي استخدام الجمل والدوالّ التي اعتاد مبرمجو VB6 وما قبلَه على استخدامها، فهي ما زالت متاحة.

هذه الدوال والجمل تنتمي لفضاء الاسم Microsoft.VisualBasic، وهو مدرج تلقائيّا في مراجع أيّ تطبيق تنشئه.

* + - [**دوال التعامل مع المحرّكات والمجلّدات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20استخدام%20جمل%20ودوالّ%20VB6\1-%20دوال%20التعامل%20مع%20المحرّكات%20والمجلّدات.htm)
    - [**دوال التعامل مع الملفّات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20استخدام%20جمل%20ودوالّ%20VB6\2-%20دوال%20التعامل%20مع%20الملفّات.htm)

**دوال التعامل مع المحرّكات والمجلّدات:**

يمنحك VB الدوال التالية للتعامل مع المحرّكات والمجلّدات:

**اقرإ السمات GetAttr:**

تعيد هذه الدالة السمات Attributes الخاصّة بالملفّ أو المجلّد أو المحرّك المرسل لها كمعامل.. مثال:

**If GetAttr("C:\1") And FileAttribute.ReadOnly Then …..**

**غيّر السمات SetAttr:**

استخدم هذه الدالة لتغيير سمات مجلّد أو ملفّ.. والجملة التالية تجعل الملفّ للقراءة فقط، دون تغيير باقي سماته:

**SetAttr("C:\2.txt", GetAttr("C:\2.txt") Or FileAttribute.ReadOnly)**

لاحظ أنّ استخدام الجملة:

**SetAttr("C:\2.txt", FileAttribute.ReadOnly)**

سيجعل الملفّ للقراءة فقط، ولكنّه سيفقده باقي سماته.. فلو كان الملفّ خفيّا Hidden مثلا، فلن يظلّ كذلك بعد تنفيذ الجملة السابقة.. ولو أردت عكس سمة "للقراءة فقط" دون تغيير باقي السمات، فعليك باستخدام المعامل Xor ـ انظر استخدام الأقنعة في ملحق رقم 2:

**SetAttr("C:\2.txt", GetAttr("C:\2.txt") Xor FileAttribute.ReadOnly)**

**إنشاء مجلّد MkDir:**

تنشئ هذه الدالّة المجلّد المرسل لها كمعامل.. ولا تستطيع هذه الدالة إنشاء أكثر من مجلّد في المرّة الواحدة.

**حذف مجلّد RmDir:**

استخدم هذه الدالة لحذف أحد المجلّدات، شريطةَ أن يكون فارغا.. فإذا لم يكن فارغا، فسيحدث خطأ.. لاحظ كذلك أنّك لا تستطيع حذف أكثر من مجلّد في كلّ مرّة تستدعي فيها هذه الدالة.

**تغيير المجلّد ChDir:**

استخدم هذه الدالة لتغيير المجلّد الحالي Current Directory، الذي يستخدم كمسار افتراضيّ لأسماء الملفّات والمجلّدات التي لا تكتب مسارها بالكامل.. والجملة التالية تجعل مجلّد الويندوز هو المجلّد الحاليّ:

**ChDir("C:\Windows")**

الآن لو استخدمت الجملة التالية:

**MkDir ("مجلّد جديد")**

فسيتمّ إنشاء المجلّد داخل مجلّد الويندوز.

ولجعل المسار الحالي هو مسار برنامجك، استخدم الجملة التالية:

**ChDir(Application.ExecutablePath)**

ويمكنك استخدام النقطتين ".." في المسار لتعبّر عن المجلّد الرئيسيّ لأحد المجلّدات.. فمثلا، الجملة التالية تغيّر المجلّد الحالي ليصبح هو المجلّد الذي يحتوي على المجلّد الحالي:

**ChDir("..")**

أمّا الجملة التالية فهي تغيّر المجلّد الحالي ليصبح هو المجلّد MyFiles الوجود معه في نفي المجلّد الرئيسيّ:

**ChDir("..\MyFiles")**

**تغيير المحرّك ChDrive:**

استخدم هذه الدالة لتغيير المجلّد الحاليّ Current Drive.. ويمكن استخدام الجملة التالية لجعل المحرّك الموجود عليه برنامجك هو المحرّك الافتراضيّ:

**ChDrive(Application.ExecutablePath)**

**المجلّد الحاليّ CurDir:**

تعيد هذه الدالة المجلّد الحاليّ.. ونظرا لأنّ لكلّ محرّك مجلّد حاليّ خاصّ به، فيمكنك إرسال اسم المحرّك الذي تريد معرفة المجلّد الحاليّ عليه.. ولو لم ترسل هذا المعامل، فستحصل على المجلّد الحاليّ على المحرّك الحاليّ.. مثال:

**MsgBox (CurDir("D"))**

**محتويات المجلّد Dir:**

تقبل هذه الدالة معاملين اختياريين على الصيغة التالية:

**Dir([المسار [, السمات]])**

وتستخدم هذه الدالّة لاسترجاع أسماء الملفّات والمجلّدات الموجودة في المسار المحدّدة، ولها السمات المحدّدة Attributes.

المشكلة أنّ دالة Dir موجودة منذ إصدارات VB الأولى، حينما لم يكن مسموحا بأن تعيد الدوال مصفوفات!.. إذن فكيف سنحصل على أسماء الملفّات والمجلّدات التي تعيدها هذه لدالة؟

في الواقعِ إنّ لهذا طريقة غريبة.. إنّ عليك أن تستدعي هذه الدالّة أكثر من مرّة، لتعيد لك اسم أحد الملفّات أو المجلّدات في كلّ مرّة، ولا تتوقّف عن فعل ذلك إلى أن تعيد هذه الدالة نصّا فارغا "" دلالة على أنّه لم يعد هناك مزيد!

وفي أوّل مرّة تستدعي فيها هذه الدالّة، يجب أن تمدّها بالمعاملات، حيث ستعيد لك أوّل ناتج، أمّا حينما تستدعيها بعد ذلك، فعليك ألا ترسل لها معاملات.

واضح طبعا أن استخدام كائني المجلّد والملفّ للحصول على محتويات المجلّد أفضل بمراحل، ليس فقط لسهولتها، ولكن أيضا لأنّك لن تستطيع استخدام دالة Dir في إجراء ارتدادي Recursive لسبر أغوار مجموعة متداخلة من المجلّدات.

والمثال التالي يريك كيفيّة استخدام هذه الدالة للحصول على الملفّات ذات الامتداد .DLL الموجودة في مجلّد النظام وحساب عددها:

**Dim DLLFile As String, DLLFiles As Integer**

**DLLFile = Dir("C:\WINNT\SYSTEM\\*.DLL")**

**If DLLFile <> "" Then DLLFiles += 1**

**While DLLFile <> ""**

**DLLFile = Dir()**

**DLLFiles += 1**

**End While**

**MsgBox(DLLFiles)**

ولو أردت أن تجعل المثال السابق يحسب عدد ملفّات DLL الخفيّة Hidden، فعليك تغيير أول استدعاء للدالة ليصبح كالتالي:

**HiddenFile = Dir("C:\WINNT\SYSTEM\\*.DLL", FileAttribute.Hidden)**

**دوال التعامل مع الملفّات:**

يمنحك VB الدوال التالية للتعامل مع الملفات:

**تاريخ الملفّ FileDateTime:**

تعيد هذه الدالة تاريخ إنشاء الملفّ.

**طول الملفّ FileLen:**

تعيد هذه الدالة حجم الملفّ المرسل لها كمعامل.

**حذف Kill:**

تسمح لك هذه الدالة بحذف الملفّ المرسل لها كمعامل، شريطةَ ألا يكونَ مفتوحا بواسطة أحد التطبيقات.. ويمكن أن تستخدمها لحذف أكثر من ملفّ، وذلك باستخدام الصيغ التي تحتوي على العلامتين \*، ?.. فمثلا، يمكنك حذف كلّ الملفّات ذات الامتداد txt الموجودة على المحرّك C:\ بالجملة التالية:

**Kill(C:\\*.txt)**

ويمكنك حذف كلّ الملفّات الموجودة في المجلّد C:\1 بالجملة التالية:

**Kill(C:\1\\*.\*)**

**نسخ ملفّ FileCopy:**

استخدم هذه الدالة لنسخ ملفّ من موضع لآخر، سواءٌ بنفس الاسم أو باسم جديد:

**FileCopy(الملفّ الأصليّ, المسار والاسم الجديدين)**

وتشترط هذه الدالة أن يكون الملفّ موجودا، وليس مفتوحا للاستخدام بواسطة أيّ تطبيق.. وهو ما يمثّل مشكلةً لا مبرّر لها بالنسبة لهذه الدالة، فلا يوجد مبرّر منطقيّ لمنعك من نسخ أحد الملفّات المفتوحة حاليّا، وهو ما يمكن أن تفعله في متصفّح الويندوز بدون أيّ مشاكل!

**إعادة التسمية Rename:**

تتيح لك هذه الدالة إعادة تسمية مجلّد أو ملفّ، تبعا للصيغة التالية:

**Rename(الاسم القديم, الاسم الجديد)**

ويجب أن تراعي هذه الشروط عند استخدام هذه الدالة:

* أن يكون الملفّ أو المجلّد موجودين.
* ألا يكون الملفّ مفتوحا بواسطة أحد التطبيقات.. وإذا كنت تحاول تغيير اسم مجلّد، فيجب ألا يكون أيّ ملفّ من الملفّات التي يحتويها مفتوحا بواسطة أحد التطبيقات.
* ألا يكون الاسم الجديد هو اسم أحد الملفّات أو المجلّدات الموجودة بالفعل.

وليس شرطا أن تضع الملفّ ذا الاسم الجديد في نفس مسار الملفّ ذي الاسم القديم.. معنى هذا أنّك تستطيع استخدام هذه الوسيلة لنقل الملفّ من موضع لآخر ـ شريطة أن يكون المسار الجديد موجودا.. ينطبق هذا على الملفّات فقط، ولا ينطبق على المجلّدات، إذ لا يمكنك نقلها من موضعها باستخدام هذه الدالة.

**القراءة من الملفّ والكتابة فيه**

* [**استخدام خلايا إطار العمل:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20القراءة%20من%20الملفّ%20والكتابة%20فيه\1-%20استخدام%20خلايا%20إطار%20العمل.htm)
* [**استخدام دوال VB:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20القراءة%20من%20الملفّ%20والكتابة%20فيه\2-%20استخدام%20دوال%20VB.htm)

**استخدام خلايا إطار العمل**

* + [**أنواع الملفّات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\0-%20أنواع%20الملفّات.htm)
  + [**كائن بيانات الملفّ FileStream:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\1-%20كائن%20بيانات%20الملفّ.htm)
  + [**كاتب البيانات النصّيّة StreamWriter:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\2-%20كاتب%20البيانات%20النصّيّة.htm)
  + [**قارئ البيانات النصّيّة StreamReader:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\3-%20قارئ%20البيانات%20النصّيّة.htm)
  + [**كائن البيانات Stream:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\4-%20كائن%20البيانات.htm)
  + [**كاتب البيانات الثنائيّة BinaryWriter:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\5-%20كاتب%20البيانات%20الثنائيّة.htm)
  + [**قارئ البيانات الثنائيّة BinaryReader:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\6-%20قارئ%20البيانات%20الثنائيّة.htm)
  + [**مشروع تخزين السجلات RecordSave:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\7-%20مشروع%20تخزين%20السجلات.htm)
  + [**خليّة التسلسل Serialization Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20استخدام%20خلايا%20إطار%20العمل\8-%20خليّة%20التسلسل.htm)

**أنواع الملفّات:**

بالنسبة لإطار العمل، هناك نوعان من الملفّات: ملفّات النصوص Text files، والملفّات الثنائيّة Binary files.. وفي النوع الأخير يتمّ تخزين البيانات في صورة أرقام ورموز غير قابلة للقراءة كنصوص، وهو ما تفعله عند تخزين الصور والأصوات وغيرها.. ويعتبر الملفّ الثنائي إحدى وسائل حماية البيانات، فمنشئه هو الوحيد الذي يعرف تنسيقه.. فمثلا: يمكنك تخزين مجموعة من السجلات في ملف ثنائي، بحيث:

- تضع في أول وحدتي تخزين 2 Bytes عدد السجلات (وليكن 100).

- وتضع في الوحدة التالية عددا تتحقّق به من أنّ الملفّ لم يعبث به أحد أو يغيّر بياناته (وليكن 236).

- وتضع في الوحدة التالية طول كل سجلّ (وليكن 53).

- ثمّ تبدأ في حفظ سجلاتك في الملفّ على التوالي، كلّ سجلّ في 53 وحدة تخزين 53 Bytes.

إنّ هذا التنسيق خاصّ بك، ومن العسير على أيّ شخص أن يستنتجه، ولو حاول قراءة الملفّ فسيبدو له كمجموعة لا معنى لها من الأرقام.

ومن الممكن قراءة ملفات النصوص على أنّها ملفّات ثنائيّة، ولكنّها في هذه الحالة ستبدو كمجموعة من الأرقام غير المقروءة، وستحتاج لكتابة الكود الذي يحوّل كلّ رقم إلى الحرف المناظر له.

وللقراءة أو الكتابة في ملفّ، يمكنك استخدام الكائنات المختلفة التي تُرجعها وسائل خليّة الملفّ وخليّة معلومات الملفّ، وهي: **Stream**، و **StreamReader** و **BinaryReader** و **StreamWriter** و **BinaryWriter**.. غنّي عن الذكر أنّ هذا الكائنات موجودة في الفضاء System.IO، فلا تنسَ استيراده في مشروعك.

**كائن بيانات الملفّ FileStream:**

يعتبر هذا الكائن حلقة الوصل بينك وبين الملفات، رغم أنّه لا يمكّنك بمفرده من القراءة من الملفّ أو الكتابة فيه.

ابدأ بتعريف متغيّر من هذا الكائن، مع ملاحظة أنّ منشئ هذا الكائن يأخذ نفس معاملات وسيلة الفتح Open في خليّة الملفّ:

**Dim FS As New FileStream(مسار الملفّ, طور فتح الملفّ)**

حيث يأخذ طور الملفّ قيمة من قيم المرقّم FileMode، وقد تعرّفنا عليها من قبل.

ويمكن أن تستخدم في التعريف أيضا الصيغة التالية:

**Dim FS As New FileStream(مسار الملفّ, طور فتح الملفّ, طور التعامل مع الملفّ)**

حيث يأخذ طور الملفّ قيمة من قيم المرقّم FileAccess، وقد تعرفنا عليه من قبل.

وهناك صيغة رابعة، تُمدّها بكيفية مشاركة الملفّ:

**Dim FS As New FileStream(مسار الملفّ, طور فتح الملفّ, طور التعامل مع الملفّ, مشاركة الملفّ)**

حيث يأخذ المعامل الأخير قيمة من قيم المرقّم FileShare، وقد تعرفنا عليه من قبل.

**الخصائص:**

لن نسهبَ في شرح هذه الخصائص، فأسماؤها تشرح وظائفها بوضوح:

تمكن قراءته **CanRead** ـ يمكن الانتقال لموضع فيه **CanSeek** ـ تمكن الكتابة فيه **CanWrite** ـ طول الملفّ (حجمه) **Length** ـ الموضع الحالي (الموضع الذي تقرأ منه أو تكتب فيه في الملفّ) **Position**.

**الوسائل:**

**إغلاق Lock:**

يمكنك بهذه الوسيلة إغلاق الملفّ أو جزء منه، بحيث لا يمكن تغيير محتويات هذا الجزء:

**FileStream.Lock(الموضع, الطول)**

ولإغلاق الملفّ كلّه:

**FileStream.Lock(1, FileStream.Length)**

**انتقال إلى Seek:**

للانتقال إلى أيّ موضع في الملفّ:

**FileStream.Seek(الإزاحة, الأصل)**

إنّ عمليّة الانتقال تكون عبارة عن إزاحة عن موضع الأصل.. ويمكن استخدام أيّ قيمة من قيم المرقّم SeekOrigin كأصل للانتقال.. وهذه القيم هي:

|  |  |
| --- | --- |
| Begin | الإزاحة منسوبة إلى بداية الملفّ. |
| Current | الإزاحة منسوبة إلى الموضع الحاليّ في الملفّ. |
| End | الإزاحة منسوبة إلى نهاية الملفّ. |

ما يجب أن تلاحظه هنا، هو أنّ التعامل مع الملفّ ليس كالتعامل مع المصفوفة، ففي المصفوفة يمكنك قراءة أيّ خانة برقمها، ولكن في الملفّ، لا بدّ أن تتحرّك للموضع الذي تريده أولا، وعند كتابتك أو قراءتك من الملفّ، ستتمّ الكتابة والقراءة في هذا الموضع، الذي سيتغيّر بعد إتمام العمليّة، ليتوقّف بعد آخر حرف تمّت قراءته أو كتابته.. فمثلا: عند فتح الملفّ تكون في أوّله، ولو قرأت 10 حروف، فستحصل على أوّل عشرة حروف من الملفّ.. ولو قرأت 10 حروف أخرى، فستحصل على الحروف العشرة التالية ( من 11 إلى 20)، وهكذا.. فإذا أحببت أن تقرأ الحرف رقم 100، فيمكنك استخدام وسيلة الانتقال Seek للوصول إليه أولا، ثم تقرأه بعد ذلك هو وما تريد من الحروف التي تليه.

**تغيير حجم الملفّ SetLength:**

يمكنك استخدام هذه الوسيلة لتقليص أو زيادة حجم الملفّ.

**FileStream.SetLength(الحجم الجديد)**

**كاتب البيانات الثنائيّة BinaryWriter:**

استخدم هذا الكائن لكتابة البيانات في الملفّات الثنائيّة Binary Files.. ولتعريف متغير من هذه الخليّة، يمكنك استخدام صيغ مشابهة لصيغ تعريف "كاتب البيانات النصّيّة" StreamWriter.

**الوسائل:**

**كتابة نصّ WriteString:**

يمكنك أن تكتب النصوص في الملفّ الثنائيّ بهذه الوسيلة.

وطبعا أن تألف هذه الوسائل:

إغلاق **Close** ـ محو **Flush** ـ انتقال **Seek** ـ كتابة **Write**.

**قارئ البيانات الثنائيّة BinaryReader:**

استخدم هذا الكائن لقراءة البيانات من الملفّات الثنائيّة Binary Files.. ولتعريف متغير من هذه الخليّة، يمكنك استخدام صيغ مشابهة لصيغ تعريف "قارئ البيانات النصّيّة" StreamReader.

**الوسائل:**

**BaseStream:**

ترجع كائنا من النوع FileStream، هو ذلك الذي تمّ تعريف كاتب البيانات الثنائيّ BinaryReader منه.

**قراءة نصّ ReadString:**

يمكنك استخدام هذه الوسيلة لقراءة النصوص التي كتبت في الملفّ بواسطة الوسيلة WriteString.

**حرف القمّة PeekChar:**

مماثلة للوسيلة Peak في قارئ البيانات النصّيّة StreamReader.. ولاختلاف الاسمين حكمة، ففي الملفات الثنائيّة يمكن أن يكون الحرف مُخزّنا في وحدة واحدة Byte أو اثنتين 2 Bytes، على حسب نوع الترميز Encoding.. وهذه الوسيلة تقرأ الحرف تبعا للترميز مهما اختلف عدد وحداته.. وترجع لك هذه الوسيلة عددا صحيحا Integer وليس حرفا.

**وسائل القراءة:**

تمنحك هذه الخليّة وسائل عديدة لقراءة كل أنواع البيانات التي يمكن أن تخزنها في ملفّ ثنائي، وهي:

|  |  |
| --- | --- |
| ReadBoolean | لقراءة متغير منطقي. |
| ReadByte | لقراءة وحدة واحدة. |
| ReadBytes | لقراءة مجموعة من الوحدات في مصفوفة:  **ReadBytes(مصفوفة وحدات, عدد الوحدات)** |
| ReadChar | لقراءة حرف، سواء أكان مخزنا في وحدة أم اثنتين، تبعا لنوع الترميز. |
|  | لقراءة مجموعة من الحروف في مصفوفة:  **ReadChars(مصفوفة حروف, عدد الحروف)** |
| ReadDecimal | لقراءة عدد عشري. |
| ReadDouble | لقراءة عدد مزدوج. |
| ReadInt16 | لقراءة عدد صحيح قصير Short (وحدتان). |
| ReadInt32 | لقراءة عدد صحيح Integer (4 وحدات). |
| ReadInt64 | لقراءة عدد صحيح طويل Long (8 وحدات). |
| ReadSByte | لقراءة وحدة ذات إشارة Signed Byte. |
| ReadSingle | لقراءة عدد مفرد (4 وحدات). |
| ReadString | لقراءة نصّ. |
| ReadUInt16 | لقراءة عدد صحيح قصير بدون إشارة Unsigned Short Integer (وحدتان). |
| ReadUInt32 | لقراءة عدد صحيح بدون إشارة Unsigned Integer (4 وحدات). |

ولاستخدام هذه الوسائل، فمن المفترض أن تعرف تركيب وتنسيق الملفّ الثنائيّ، بحيث تقرأ أنواع البيانات بالترتيب الصحيح والنوع الصحيح.. فإذا أخطأت في قراءة قيمة واحدة، فلن تكون أيّ قيمة ممّا تليها صحيحة.

لاحظ كذلك أنّ كلّ هذه الوسائل تقرأ قيمةً بقيمة، ولو أردت قراءة أكثر من قيمة متتالية، فلا بد من أن تستدعي نفس الوسيلة أكثر من مرّة (في جملة تكراريّة مثلا).. أمّا لو أردت أن تخزّن سجلا أو مصفوفة من السجلات أو أيّ نوع من المصفوفات أو الكائنات أو مجموعات الكائنات Collections، فانتظر حتّى نشرح سَلسَلة الكائنات Serialization عن طريق الخليّة BinaryFormater لاحقا في هذا الفصل.

**كائن البيانات Stream:**

يعتبر كائنا عاما للتعامل مع الملفّات أو أجهزة الإدخال والإخراج أو الاتصال بين العمليّات المختلفة أو برتوكولات الإنترنت.. وعن طريق هذا الكائن يمكنك أداء ثلاث عمليّات: القراءة من الملفّ، والكتابة فيه، وتغيير الموضع الحاليّ فيه.. لهذا يعتبر هذا الكائن شاملا للكائنات FileStream و StreamReader و StreamWriter، حيث ستجد أنّه يجمع معظم خصائصها ووسائلها مثل:

يمكن القراءة **CanRead** ـ يمكن البحث **CanSeek** ـ يمكن الكتابة **CanWrite** ـ حجم الملفّ **Length** ـ الموضع **Position** ـ إغلاق **Close** ـ محو **Flush** ـ قراءة **Read** ـ قراءة وحدة **ReadByte** ـ بحث **Seek** ـ تغيير حجم الملفّ **SetLength** ـ كتابة **Write** ـ كتابة وحدة **WriteByte**.

ولن نستفيض في شرحها فهي واضحة.. ولكنّ ما يجب أن تلاحظه هنا، هو أنّ الكائنات الثلاثة السابقة تمنحك قدراتٍ أكثر قليلا من هذا الكائن، باعتبار أنّ كلا منها متخصّص في وظيفة معيّنة.. عامّة لديك حريّة اختيار ما يناسب برنامجك.

**كاتب البيانات النصّيّة StreamWriter:**

يمكّنك هذا الكائن من الكتابة في الملفّات النّصّيّة:

**Dim SW As New StreamWriter(مسار الملفّ)**

وتمكّنك الصيغة التالية من استبدال الملفّ الجديد بأيّ ملفّ موجود يحمل نفس الاسم:

**Dim SW As New StreamWriter(مسار الملفّ, True)**

أمّا الصيغة التالية، فهي تمكّنك من تحديد نوع الترميز Encoding الذي يحفظ به النصّ في الملفّ:

**Dim SW As New StreamWriter(المسار, استبدال, ترميز)**

وعليك باستخدام نوع الترميز التالي: System.Text.Encoding.Default.

أمّا الصيغة الرابعة، فتسمح لك بتحديد حجم مساحة التخزين في الذاكرة Buffer، التي سيتمّ وضع النصوص بها مؤقّتا قبل كتابتها في الملفّ:

**Dim SW As New StreamWriter(المسار, استبدال, ترميز, حجم المخزن)**

أمّا الصيغة الخامسة، فتسمح لك بإنشاء "كاتب بيانات نصّيّة" StreamWriter من كائن بيانات الملفّ FileStream:

**Dim SW As New (FStream)**

مثال:

**Dim FS As New FileStream ("C:\TextData.txt", FileMode.Create)**

**Dim SW As New StreamWriter(FS)**

وحتّى لا تكون محروما من شيء، لديك صيغتان إضافيتان لتعريف هذا الكائن:

**Dim FS As New StreamWriter(FStream, الترميز)**

**Dim FS As New StreamWriter(FStream, الترميز, حجم المخزن)**

**الخصائص:**

**سطر جديد NewLine:**

ينتهي السطر في الوضع التلقائيّ بالحرف Chr(13)، بينما يبدأ السطر التالي له بالحرف Chr(10).. وتمكّنك هذه الخاصيّة من تغيير الحرف الذي ينتهي به السطر إلى أيّ رمز آخر تريده.

**الوسائل:**

**محو تلقائي AutoFlush:**

لو جعلت هذه الخاصيّة True، فإنّ النصوص الموجودة في مخزن الذاكرة Buffer ستمحى تلقائيّا بعد كتابتها للملفّ.. فإذا جعلت قيمتها False، فستبقى هذه النصوص في الذاكرة إلى أن يجد نظام التشغيل فرصة مناسبة لمحوها.. طبعا الحالة الثانية ستسرّع برنامجك، خاصّة عندما تتابع عمليات الكتابة في الملفّ أو في الملفّات الكبيرة، ولكنّ الثمن قد يكون على حساب مساحة الذاكرة.

**محو Flush:**

استخدم هذه الوسيلة لتمحو بنفسك أيّ نصوص كتبت في الذاكرة.

**إغلاق Close:**

هذه الوسيلة تغلق كاتب البيانات، وتحرر الذاكرة منه.

**كتابة Write:**

هذه الوسيلة تكتب في الملفّ، البيانات التي ترسلها لها كمعامل.. ولهذا الدالة صيغ كثيرة، وذلك حتّى تقبل أيّ نوع من أنواع البيانات كمعامل.. مع ملاحظة أنّ الكتابة تكون في الموضع الحاليّ في الملفّ.. هذا الموضع يكون في بداية الملف عند فتحه.. ولو قمت بالكتابة في هذه اللحظة، فسيتمّ استبدال ما هو موجود سابقا في الملف.

أمّا لو شئت التحرّك لموضع معيّن، فيمكنك استخدام الوسيلة [Seek](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\1-%20كائن%20بيانات%20الملفّ.htm).. مع ملاحظة أنّ الكتابة في أيّ موضع تلغي محتوياته السابقة.

أمّا لو شئت إضافة البيانات في نهاية الملف، دون إلغاء محتوياته السابقة، فعليك أن تفتح الملفّ للإضافة، قبل الكتابة فيه مباشرة، كالتالي:

**Dim Sw As StreamWriter = File.AppendText("C:\1.txt")**

**Sw.Write("هذه الكلمات ستضاف بعد المحتويات السابقة للملف")**

**Sw.Close()**

**كتابة سطر WriteLine:**

مماثلة للطريقة السابقة، ولكنّها تكتب البيانات المرسلة في سطر جديد.

**مثال:**

الكود التالي يريك كيف تكتب أنواع البيانات المختلفة في ملفّ:

**Dim FS As New FileStream("C:\TextData.txt", FileMode.Create)**

**Dim SW As New StreamWriter(FS)**

**SW.WriteLine(9.009)**

**SW.WriteLine(1 / 3)**

**SW.Write("تاريخ اليوم هو ")**

**SW.Write(Now())**

**SW.WriteLine() ' ستكتب هذه الوسيلة سطرا جديدا فارغا**

**SW.WriteLine(True)**

**' لاحظ كيف ستتم كتابة الكائن بكلّ عناصره، وكيف سيظهر ذلك في الملفّ**

**SW.WriteLine(New Rectangle(1, 1, 100, 200))**

**SW.WriteLine(Color.YellowGreen)**

**SW.Close()**

**FS.Close()**

ولو فتحت الملفّ الناتج فستجد فيه ما يلي:

9.009

0.333333333333333

تاريخ اليوم هو 07/11/2002 11:52:54 م

True

{X=1,Y=1,Width=100,Height=200}

Color [YellowGreen]

**قارئ البيانات النصّيّة StreamReader:**

يمدّك هذا الكائن بالوسائل اللازمة للقراءة من ملفّ.. هذه هي صيغ تعريفه، ولا جَرَمَ أنّها غنيّة عن الشرح:

**Dim SR As New StreamReader(مسار الملفّ)**

**Dim SR As New StreamReader(مسار الملفّ, الترميز)**

**Dim SR As New StreamReader(مسار الملفّ, الترميز, حجم المخزن)**

**Dim SR As New StreamReader(FStream)**

**Dim SR As New StreamReader(FStream, الترميز)**

**Dim SR As New StreamReader(FStream, الترميز, حجم المخزن)**

**الوسائل:**

**القمّة Peek:**

يمكنك بهذه الوسيلة أن تقرأ الحرف التالي للموضع الحاليّ في الملفّ، بدون أن يتغيّر الموضع الحالي.. فإذا لم يكن هناك حرف تالٍ فإنّ هذه الوسيلة تُرجع -1.

**قراءة Read:**

يمكنك بهذه الوسيلة أن تقرأ الحرف التالي في الملفّ، مع زيادة الموضع الحالي بواحد.. فإذا لم يكن هناك حرف تالٍ فإنّ هذه الوسيلة ترجع -1.

**charsRead = Read()**

وهناك صيغة أخرى من هذه الوسيلة، تقرأ العدد محدّدا من الحروف من الملفّ، وتعيد عدد ما قرأته من الحروف، والذي في الغالب سيكون نفس العدد الذي طلبت قراءته، إلا إذا كان ما بقي في الملفّ من الحروف أقلّ من العدد الذي تريد قراءته، ففي هذه الحالة يكون الناتج -1، ممّا يعني أنّك قد وصلت لنهاية الملفّ.. انظر للصيغة التالية:

**charsRead = Read(مصفوفة حروف, رقم خانة البداية في مصفوفة الحروف, عدد الحروف المطلوب قراءتها)**

واضح أنّ ناتج هذه الوسيلة يأتي عبر المعامل الأوّل، وهو أيّ مصفوفة حروف ترسلها لاستقبال الحروف المقروءة.

**قراءة مقطع ReadBlock:**

مماثلة للصيغة الثانية من الوسيلة السابقة!

**قراءة سطر ReadLine:**

تقرأ سطرا من الملفّ (بدون علامتي نهاية السطر وبدايته)، فإذا كان الموضع الحالي هو نهاية الملفّ، فإنّها ترجع القيمة Null.

**Dim txtLine As String = SR.ReadLine()**

**اقرأ لنهاية الملفّ ReadToEnd:**

تقرأ من الموضع الحالي إلى نهاية الملفّ.. فإذا كان الموضع الحاليّ هو بداية الملفّ، فإنّها تقرأ الملفّ كلّه.

**Dim allText As String = SR.ReadToEnd()**

**مثال:**

افترض أنّ لديك ملفّا يحتوي على سطور، وتريد قراءة كل سطر ليصبح عنصرا في قائمة:

**Dim FS As New FileStream("C:\Items.txt", FileMode.Open)**

**Dim SR As New StreamReader(FS)**

**Dim itm As Object**

**Do**

**itm = SR.ReadLine()**

**If itm = Nothing Then Exit Do**

**ListBox1.Items.Add(itm)**

**Loop**

**SR.Close()**

**FS.Close()**

**مشروع تخزين السجلات RecordSave:**

الآن ترى كيف يمكن حفظ سجلات الأسعار في ملفّ ثنائيّ، ثمّ تحميلها مرّة أخرى.

سيتكوّن كلّ سجلّ من الحقول التالية:

- نصّ يمثّل تعريف المنتج ID.

- نصّ يمثّل وصف المنتج.

- عدد صحيح يمثّل السعر.

- قيمة منطقيّة Boolean تدّل إذا ما كان المنتج متوفّرا أم لا.

- عدد صحيح يمثّل أقلّ كمّيّة للطلبيّة.

وفي هذا البرنامج، سنحفظ كلّ قيمة بطريقة مستقلّة، فكما قلنا، ليس لديك ما يحفظ سجلا كاملا مرة واحدة.



ابدأ بتعريف السجلّ التالي:

**Structure Product**

**Dim ProdID As String**

**Dim prodDescription As String**

**Dim listPrice As Single**

**Dim available As Boolean**

**Dim minStock As Integer**

**End Structure**

ها هو الكود الذي يحفظ مجموعة من السجلات في ملفّ ثنائي:

**Private Sub Button1\_Click(ByVal sender As Objesct, \_**

**ByVal e As EventArgs) Handles Button1.Click**

**Dim FS As New FileStream("Records.bin", FileMode.OpenOrCreate, \_**

**FileAccess.Write)**

**Dim BW As New BinaryWriter(FS)**

**' اجعل الموضع الحالي هو بداية الملفّ**

**BW.BaseStream.Seek(0, SeekOrigin.Begin)**

**' الجملة السابقة مكافئة للجملة التالية:**

**' FS.Seek(0, SeekOrigin.Begin)**

**Dim p As New Product()**

**' حفظ أوّل سجلّ**

**p.ProdID = "100-A39"**

**p.prodDescription = "Cellular Phone with built-In TV"**

**p.listPrice = 497.99**

**p.available = True**

**p.minStock = 40**

**SaveRecord(BW, p)**

**' حفظ السجلّ الثاني**

**p = New Product()**

**p.ProdID = "100-U300"**

**p.prodDescription = "Wireless Handheld"**

**p.listPrice = 315.5**

**p.available = False**

**p.minStock = 12**

**SaveRecord(BW, p)**

**' حفظ السجلّ الثالث**

**p = New Product()**

**p.ProdID = "ZZZ"**

**p.prodDescription = "Last Gadget"**

**p.listPrice = .99**

**p.available = True**

**p.minStock = 1000**

**SaveRecord(BW, p)**

**BW.Close()**

**FS.Close()**

**End Sub**

بقي أن نكتب كود الإجراء SaveRecord:

**Sub SaveRecord(ByVal writer As BinaryWriter, ByVal record As Product)**

**writer.Write(record.ProdID)**

**writer.Write(record.prodDescription)**

**writer.Write(record.listPrice)**

**writer.Write(record.available)**

**writer.Write(record.minStock)**

**End Sub**

ولقراءة السجلات المحفوظة في الملفّ، استخدم الكود التالي:

**Private Sub Button2\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Button2.Click**

**Dim BR As BinaryReader**

**Dim FS As FileStream**

**FS = New System.IO.FileStream("Binarydata2.bin", FileMode.Open, \_**

**FileAccess.Read)**

**BR = New System.IO.BinaryReader(FS)**

**BR.BaseStream.Seek(0, SeekOrigin.Begin)**

**Dim p As New Product()**

**TextBox1.Clear()**

**' نظرا لأنّك لا تعرف عدد السجلات في الملفّ**

**' فاستمر في القراءة من الملفّ طالما أنّ موضعك الحاليّ أصغر من طول الملفّ**

**While FS.Position < FS.Length**

**p = Nothing**

**' اقرإ حقول السجلّ بنفس الترتيب الذي أخلتها به، مع مراعاة أنواع البيانات**

**p.ProdID = BR.ReadString**

**p.prodDescription = BR.ReadString**

**p.listPrice = BR.ReadSingle**

**p.available = BR.ReadBoolean**

**p.minStock = BR.ReadInt32**

**' اعرض السجلّ في مربّع النصّ**

**ShowRecord(p)**

**End While**

**BR.Close()**

**FS.Close()**

**End Sub**

ولا داعي لذكر كود الإجراء ShowRecord، فكلّ ما يفعله هو إضافة حقول السجلّ إلى محتويات مربّع النصّ.

ممّا سبق تجدّ أنّ تلك الطريقة لتخزين السجلات غير مرضيّة، لهذا فمن الأفضل أن نتعرّف على وسيلةٍ أيسر وأكفأ للتعامل مع مجموعة ضخمة من البيانات.

**خليّة التسلسل Serialization Class**

سنتعرّف هنا على إمكانيّة رائعة، تستطيع بها حفظ المصفوفات والمجموعات Collections بما تحويه من عناصر في ملف، بحيث يمكن تحميلها مرّة أخرى في أيّ وقت تشاء.. هذه الطريقة هي استخدام الوسيلة "تسلسل" Serialize الخاصّة بخليّة التسلسل، وذلك لحفظ المجموعة في ملفّ، والوسيلة "فك التسلسل" Deserialize لتحميل المجموعة من الملفّ.

* + - [**سلسلة الكائنات المنفردة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\8-%20خليّة%20التسلسل\1-%20سلسلة%20الكائنات%20المنفردة.htm)
    - [**سلسلة مجموعة من الكائنات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\8-%20خليّة%20التسلسل\2-%20سلسلة%20مجموعة%20من%20الكائنات.htm)
    - [**فكّ تسلسل الكائنات Deserializing Objects:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\8-%20خليّة%20التسلسل\3-%20فكّ%20تسلسل%20الكائنات.htm)

**سلسلة الكائنات المنفردة:**

في البداية عرّف متغيّرا من خليّة "المنسّق الثنائي" BinaryFormatter كالتالي:

**Dim BFormatter As New \_**

**System.Runtime.Serialization.Formatters.Binary.BinaryFormatter()**

وحتّى لا تُضطرّ لكتابة كل هذا المسار الطويل في كلّ مرّة، اكتب جملة الاستيراد التالية في أقصى بداية الملفّ، قبل تعريف النموذج أو الخليّة:

**Imoprts System.Runtime.Serialization.Formatters.Binary**

إنّ المنسّق الثنائيّ يحفظ الكائنات في تنسيق ثنائي ـ طبعا!

ويمكنك أيضا أن تحفظ الكائنات في تنسيق نصّيّ عن طريق الخليّة SoapFormatter.. وليس المقصود بكلمة Soap هنا رشوة أو تملّق أو صابون، فليس من المنطقيّ أن تظنّ أنّ اسم هذه الخليّة "المنسّق الصابونيّ"!!

إنّ كلمة (SOAP) هي اختصار لمصطلح "البروتوكول البسيط للتعامل مع الكائنات" Simple Object Access Protocol.. لا مجال هنا إذن لأبلة نظيفة!!

ويحفظ هذا المنسّق الكائن في تنسيق XML، التي تمتاز بالحشو وكبر الحجم مقارنة بالملفّات الثنائيّة، ولكن رغم ذلك تعتبر الآن قياسيّةً لتبادل المعلومات بين العمليات والبرامج وأنظمة التشغيل المختلفة وعبر الإنترنت.

ولاستخدام هذا المنسّق النصّيّ، يجب أولا أن تضيف مرجعا له في البرنامج.. اضغط بزر الفأرة الأيمن على كلمة "مراجع" References في متصفّح المشاريع، ومن القائمة الموضعيّة اضغط "إضافة مرجع" Add Reference.. في مربّع الحوار الذي سيظهر لك اضغط شريط .NET، ومن القائمة ابحث عن المرجع التالي ثم انقره بالفأرة مرّتين، حتّى ينتقل القائمة الموجودة في الجزء السفليّ من مربّع الحوار:

System.Runtime.Serialization.Formatters.Soap

بعد إغلاق مربّع الحوار، أضف جملة الاستيراد التالية لبداية الملفّ:

**Imports System.Runtime.Serialization.Formatters.Soap**

الآن يمكنك تعريف متغيّر من هذه الخليّة:

**Dim formatter As Soap.SoapFormatter**

ولحسن حظّك أنّ للمنسّق النصّيّ نفس وسائل المنسّق الثنائيّ، لهذا فسنكتفي هنا بشرح ثانيهما، وعليك أنت باختبار الأوّل.

عامّةً إنّ ما يعنينا هنا هو وسيلة السلسلة، وهي بالصيغة التالية:

**BFormatter.Serialize(stream, object)**

والمثال التالي يريك كيف نحفظ كائنين من النوع مستطيل Rectangle في الملفّ "C:\SHAPES.BIN".. ولا غبار عليك في اختيار اسم الملفّ أو امتداده أن تجعلهما كما تهوى:

**Dim R1 As New Rectangle(1,1,10,20)**

**Dim R2 As New Rectangle(10,10,100,200)**

**Dim saveFile As FileStream = File.Create("C:\SHAPES.BIN")**

**Dim formatter As New BinaryFormatter**

**formatter.Serialize(saveFile, R1)**

**formatter.Serialize(saveFile, R2)**

**saveFile.Close() ' يجب إغلاق الملفّ بعد انتهائك من الكتابة فيه**

**سلسلة مجموعة من الكائنات:**

لا جديد.. كلّ ما ستفعله هو أن تضيف كائناتك إلى مجموعة Collection، حيث تقوم بسلسلتها دفعة واحدة.. يمكنك كذلك أن تفعل المثل مع المصفوفات من أيّ نوع.

وفي المثال التالي نعرّف مستطيلين، ونضيفهما لمصفوفة قائمة ArrayList وكذلك بعض الألوان، ثمّ نقوم بسلسلتها:

**Dim R1 As New Rectangle(1,1,10,20)**

**Dim R2 As New Rectangle(10,10,100,200)**

**Dim shapes As New ArrayList()**

**shapes.Add(R1)**

**shapes.Add(R2)**

**shapes.Add(Color.Chartreuse)**

**shapes.Add(Color.DarkKhaki.GetBrightness)**

**shapes.Add(Color.DarkKhaki.GetHue)**

**shapes.Add(Color.DarkKhaki.GetSaturation)**

**Dim saveFile As FileStream**

**saveFile = File.OpenWrite("C:\ShapesColors.bin")**

**' استخدم الجملة التالية للكتابة في نهاية الملفّ، بحيث لا تمحو محتوياته السابقة**

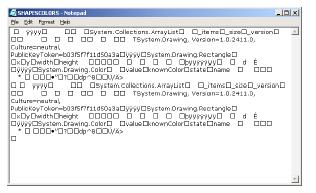
**saveFile.Seek(0, SeekOrigin.End)**

**Dim formatter As BinaryFormatter = New BinaryFormatter()**

**formatter.Serialize(saveFile, shapes)**

**saveFile.Close()**

ولو فتحت الملفّ "C:\ShapesColors.bin" بأيّ محرّر نصوص، فستجد أنّه ملئ بالرموز غير المقروءة، وذلك لأنّه ملفّ ثنائي وليس نصّيا، ولكن رغم هذا، ستجدّ أنّك ما زلت تستطيع قراءة أسماء الخصائص التي حفظناها في هذا الملفّ.



**فكّ تسلسل الكائنات Deserializing Objects:**

لاستعادة الكائنات المحفوظة، فإنّنا ببساطة نستخدم الوسيلة Deserialize كالتالي:

**كائن = Bformatter.Deserialize(FileStream)**

انظر كيف سنقرأ البيانات التي حفظناها في المثال السابق:

**' افتح الملفّ وافرأ محتوياته**

**Dim readFile As FileStream**

**readFile = File.OpenRead("C:\ShapesColors.bin")**

**Dim BFormatter As New BinaryFormatter()**

**Dim Shapes As New ArrayList()**

**Shapes = CType(BFormatter.Deserialize(readFile), ArrayList)**

ولمزيد من التدريب، عليك بالمشروع Serialization في مجلّد برامج هذا الفصل.

كما يمكنك أيضا فحص الكود المكتوب في القائمة الرئيسيّة المسمّاة Frequency Table في تطبيق WordFrequencies، في مجلّد برامج الفصل التاسع، حيث يقوم بحفظ الجدول المختلط HashTable إلى ملفّ ثنائيّ أو نصّيّ وإعادة فتحه.. وكما ذكرنا من قبل، لا يوجد اختلاف في الكود بين استخدام المنسّق النصّيّ والمنسّق الثنائيّ لحفظ أو قراءة الكائنات.

**استخدام دوال VB**

رغم الإمكانيات التي تمنحها خلايا إطار العمل، إلا إنّ مبرمجي VB6 قد يغمرهم الحنين للتعامل مع الملفّات بالطرق التي ألفوها على مدار ست إصدارات متتابعة.. ولحسن الحظّ ما زالت هذه الطرق متاحة هنا.

* [**أنواع الملفّات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20دوال\1-%20أنواع%20الملفّات.htm)
* [**فتح الملفّ وإغلاقه**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20دوال\2-%20فتح%20الملفّ%20وإغلاقه.htm)
* [**دوال الكتابة والقراءة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\2-%20استخدام%20دوال\3-%20دوال%20الكتابة%20والقراءة.htm)

**أنواع الملفّات:**

عندما استخدمنا خلايا إطار العمل Framework رأينا أنّها تتعامل مع نوعين من الملفّات: الملفّات النصّيّة Text Files، والملفّات الثنائيّة Binary Files.. إنّ هذا التقسيم سيختلف ها هنا، فـ VB يقسّم الملفّات التي يتعامل معها إلى الأنواع التالية:

|  |  |
| --- | --- |
| Sequential Files | ملفّات الوصول المتتابع.. وهي تعتبر ملفّات نصيّة (حتّى الأرقام تحفظ كنصوص)، وإن كانت العبرة ليست بما تكتبه فيها، ولكن بطريقة الكتابة، ففي هذه الملفّات يتمّ كتابة البيانات وقراءتها على التوالي (من البداية للنهاية).. ولا يمكن فتح الملفّات للقراءة والكتابة معا في نفس اللحظة بهذه الطريقة.. يجب فتح الملفّ للقراءة ثمّ إغلاقه أولا، ثمّ إعادة فتحه للكتابة.. وتناسبك هذه الطريقة للتعامل مع الملفّ عندما تريد نقل محتويات ملفّ إلى مربّع نصّ، أو العكس. |
| Random Files | ملفّات الوصول العشوائي.. وهي عكس الطريقة السابقة لفتح الملفّات، إذ تمكّنك من القراءة من أيّ موضع في الملفّ أو الكتابة فيه مباشرةً، دون أن تضطرّ للمرور على محتويات الملفّ من بدايتها لنهايتها.. وفي هذا النوع من الملفّات، يتمّ حفظ النصوص كحروف (كلّ حرف في Byte)، ولكنّ الأرقام تُحفظ بأنواعها (أعداد صحيحة Integers، أعداد طويلة Long... إلخ).. وتستخدم هذه الملفّات في حفظ البيانات التي تتكوّن من سجلات Records ذات أطوال ثابتة، لهذا تستخدم المصفوفات والسجلات Structures في كتابة البيانات في هذا النوع.. إنّ تساوي أطوال وحدات الملفّ يجعل التعامل معه شبيها بالتعامل مع المصفوفات، حيث يمكن الوصول إلى أيّ وحدة في الملفّ عن طريق معرفة رقمها Index.. ويمكنك فتح هذه الملفّات للقراءة والكتابة في وقت واحد. |
| Binary Files | ملفّات ثنائيّة.. وهي مشابهة لملفّات الوصول المتتابع، مع خلاف واحد، هو أنّ كلّ البيانات توضع فيها في صورة وحدات ذاكرة Bytes تحتوي على أرقام (سواء أكانت تمثّل حروفا أو أرقاما أو غير ذلك)، كما تعرّفنا على ذلك سابقا. |

وسنتعرّف على كيفيّة التعامل مع هذه الأنواع من الملفّات في المقاطع التالية.

**فتح الملفّ وإغلاقه:**

عند فتح الملفّ، يتمّ حجز مساحة في الذاكرة لنقل بعض محتويات الملفّ إليها.. وإذا لم يكن الملفّ موجودا، يتمّ إنشاؤه أولا ثمّ فتحه.

وتستخدم الدالة FileOpen() لفتح الملفّ.. هذه الدالة لها الصيغة التالية:

**FileOpen(رقم الملفّ, مساره, طور الفتح[, طور التعامل][, طور المشاركة][, طول السجلّ])**

**ملحوظة:**

اختفت جمل التعامل مع الملفّات وتحوّلت إلى دوالّ يبدأ معظمها بمقطع File.. فمثلا، تحوّلت جملة Open إلى الدالة FileOpen.

ولكن ما هي حكاية رقم الملفّ هذا؟

نوع من الاختصار.. فبدلا من أن تتعامل مع الملفّ المفتوح طوال الوقت باستخدام اسمه الكامل، أعطِه رقما عند فتحه (1 أو 2 أو .... )، واستخدم هذا الرقم في التعامل مع الملفّ بعد ذلك.

لاحظ أنّه من غير المسموح فتحُ ملفّين بنفس الرقم.. فإذا كنت بصدد فتح العديد من الملفّات، فيمكنك أن تستخدم الدالة FreeFile لتحصل على رقم جديد غير مستعمل.. انظر للمثال التالي:

**Dim No As Integer = FreeFile()**

**FileOpen(No, "C:\1.txt")**

**ملحوظة:**

استدعاء الدالة FreeFileأكثر من مرّة يعيد نفس الرقم، طالما لم يتمّ فتح ملفّ بهذا الرقم.

ويأخذ المعامل الثالث (طور الفتح) إحدى قيم المرقّم OpenMode التالية:

|  |  |
| --- | --- |
| Input | فتح الملفّ للقراءة منه (وصول متتابع). |
| Output | فتح الملفّ للكتابة فيه (وصول متتابع).. لاحظ فتح الملفّ بهذه الطريقة يُلغي محتوياته السابقة، حتّى ولو لم تكتب فيه! |
| Append | فتح الملفّ للإضافة إلى محتوياته السابقة (وصول متتابع). |
| Random | فتح الملفّ للقراءة أو الكتابة فيه في موضع أيّ سجلّ تختاره (وصول عشوائي). |
| Binary | فتح الملفّ كملفّ ثنائيّ. |

ويأخذ المعامل الرابع (طور التعامل) إحدى قيم المرقّم OpenAccess التالية:

|  |  |
| --- | --- |
| Read | فتح الملفّ للقراءة فقط. |
| Write | فتح الملفّ للكتابة فقط. |
| ReadWrite | فتح الملفّ للقراءة والكتابة. |

لاحظ أنّ هذا المعامل لن يُفيدك عند التعامل مع ملفات الوصول المتتابع، وإنّما سيفيدك عند التعامل مع ملفّات الوصول العشوائيّ والملفّات الثنائيّة.

أمّا المعامل الخامس (طور المشاركة)، فيأخذ إحدى قيم المرقّم OpenShare التالية:

|  |  |
| --- | --- |
| Shared | يمكن للتطبيقات الأخرى فتح الملفّ أثناء فتحك له.. ويجب أن تكون حذرا عند استخدام هذه الطريقة، حتّى لا تتسبّب التطبيقات الأخرى في تغيير محتويات الملفّ بينما تستخدمه أنت. |
| LockRead | لا يُسمح للتطبيقات الأخرى بالقراءة من الملفّ أثناء فتحك له. |
| LockWrite | لا يُسمح للتطبيقات الأخرى بالكتابة في الملفّ أثناء فتحك له. |
| LockReadWrite | لا يُسمح للتطبيقات الأخرى بالقراءة من الملفّ أو الكتابة فيه أثناء فتحك له. |

وأخيرا، لدينا المعامل السادس (طول السجلّ)، وهو يستخدم في تحديد عدد الوحدات Bytes التي تكتبها أو تقرأها في كلّ مرّة أثناء تعاملك مع ملفات الوصول العشوائيّ.

والمثال التالي يريك كيف تفتح ملفّا في طور الوصول المتتابع:

**Dim No As Integer = FreeFile()**

**FileOpen(No, "c:\samples\vb\cust.dat", OpenMode.Output, OpenAccess.ReadWrite )**

وبعد أن تُنهي تعاملك مع الملفّ يجب أن تقوم بإغلاقه، حتّى يتمّ تحرير المساحة التي يستهلكها من الذاكرة.. ولإغلاق الملفّ لديك الدالة FileClose، التي تستقبل كمعامل رقم الملفّ:

**FileClose(No)**

ولو شئت إغلاق كلّ الملفّات المفتوحة بجملة واحدة، فاستخدم الدالة Reset كالتالي:

**Reset()**

**دوال الكتابة والقراءة**

* [**دوالّ عامّة لجميع أنواع الملفّات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20دوال%20الكتابة%20والقراءة\1-%20دوالّ%20عامّة%20لجميع%20أنواع%20الملفّات.htm)
* [**ملفّات الوصول المتتابع Sequential Files:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20دوال%20الكتابة%20والقراءة\2-%20ملفّات%20الوصول%20المتتابع.htm)
* [**الملفّات الثنائيّة Binary Files:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20دوال%20الكتابة%20والقراءة\3-%20الملفّات%20الثنائيّة.htm)
* [**ملفّات الوصول العشوائيّ Random Files:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20دوال%20الكتابة%20والقراءة\4-%20ملفّات%20الوصول%20العشوائيّ.htm)

**دوالّ عامّة لجميع أنواع الملفّات:**

**نهاية الملفّ EOF:**

اسم هذه الدالة هو اختصار لتعبير End Of File، وهي تستقبل رقم أحد الملفّات المفتوحة، وتعيد True إذا كان موضع الكتابة أو القراءة الحاليّ قد وصل لنهاية هذا الملفّ.

**طول الملفّ LOF:**

اسم هذه الدالة هو اختصار لتعبير Length Of File، وهي تستقبل رقم أحد الملفّات المفتوحة، وتعيد عدد الوحدات Bytes التي يحتويها هذا الملفّ.

**الموضع الحالي Loc:**

اسم هذه الدالة هو اختصار للكلمة الإنجليزيّة Location، وهي تستقبل رقم ملفّ مفتوح، وتعيد رقم البايت الذي يمثّل موضع القراءة أو الكتابة الحاليّ في الملفّ.

**بحث Seek:**

تغيّر هذه الدالة موضع القراءة أو الكتابة الحاليّ في الملفّ إلى الموضع الذي تريد أن تتمّ فيه عمليّة الكتابة أو القراءة التالية.. ولهذه الدالة صيغتان:

**Seek(رقم الملفّ[, الموضع الجديد])**

**الموضع الحالي = Seek(رقم الملفّ)**

جليّ طبعا أنّ الصيغة الثانية تقوم بوظيفة الدالة Loc.

**إغلاق التعامل Lock:**

استخدم هذه الدالة لحماية أيّ جزء من الملفّ ـ أو الملفّ كلّه ـ ضد تعامل أيّ تطبيق آخر معه.. وطبعا ستتمكّن أنت من خلال تطبيقك من التعامل مع هذا الجزء المحميّ.. ولهذه الدالة الصيغ التالية:

**Lock(رقم الملفّ) ' لحماية الملفّ كلّه**

**Lock(رقم الملفّ, رقم السجلّ) ' لحماية سجلّ واحد**

**Lock(رقم الملفّ, رقم أوّل سجلّ, رقم آخر سجلّ) ' لحماية مجموعة من السجلات**

لاحظّ أنّ محاولة أيّ تطبيق آخر للتعامل مع السجلات المحميّة سيدفع VB لإطلاق استثناء في تطبيقك لتحذيرك.

لاحظ كذلك أن استخدام هذه الدالة مع ملفّات الوصول المتتابع يؤدّي لحماية الملفّ كلّه، حتّى لو حدّدت أنت مجموعة من السجلات فقط.. لا تسلني لماذا.. اسأل (ميكروسوفت)!!

**إعادة التعامل Unlock:**

بَدَهِيٌّ أن تكون هناك دالة تقوم بعكس ما تقوم به الدالة السابقة، لتتيح لك إزالة الحماية المضروبة على بعض أو كلّ سجلات الملفّ.. ولهذه الدالة نفس صيغ الدالة السابقة.

**سمات الملفّ FileAttr:**

أرسل لهذه الدالة رقم الملفّ المفتوح، لتعيد لك إحدى قيم المرقّم OpenMode الذي تعرّفنا عليه من قبل، والتي تمثّل طور فتح الملفّ (للقراءة، للكتابة، للإضافة... إلخ).

لا ترتبك بين هذه الدالة والدالة GetAttr، التي تعيد سمات الملفّ (خفيّ، للقراءة فقط، نظام... إلخ) بدون أن يشترط أن يكون الملف مفتوحا.

**ملفّات الوصول المتتابع Sequential Files:**

**طباعة Print، كتابة Write:**

استخدم هاتين الدالتين للكتابة في ملفّات الوصول المتتابع Sequential Files، على الصيغة التالية:

**Print (رقم الملفّ, البيانات)**

**Write(رقم الملفّ, البيانات)**

لاحظ أنّ المعامل الثاني هو مصفوفة معاملات ParamArray، ممّا يعني أنّ بإمكانك إرسال أيّ عدد من المتغيّرات لكتابة محتوياتها في الملفّ.. انظر للمثال التالي:

**Print(1, var1, var2, "نصّ تجريبي", 333.333)**

ولكن ما هو الفارق بينَ الدالتين Print **و** Write؟

الفارق الوحيد هو تنسيق المخرجات.. فبينما تضيف الدالة Printقيم مصفوفة المعاملات متجاورة، تقوم الدالة Writeبوضع كلّ قيمة بينَ علامتي تنصيص، وتفصل بين القيم بفاصلة "،".

**طباعة سطر PrintLine وكتابة سطر WriteLine:**

هاتان الدالتان مماثلتان للدالتين السابقتين، ولكنّ القيم المرسلة في مصفوفة المعاملات تتمّ طباعتها في سطر جديد.

ويمكنك استخدام الدالة TAB، لتحديد عدد علامات الجدولة المضافة بين قيمة وأخرى، وذلك لتنسيق المخرجات.. انظر للمثال التالي:

**On Error Resume Next**

**Kill("c:\test.txt")**

**FileOpen(1, "c:\test.txt", OpenMode.Output)**

**PrintLine(1, "محمد", TAB(12), "حمدي", TAB(25), "غانم", TAB(45), 33)**

**PrintLine(1, "سامح", TAB(12), "حسن", TAB(25), "النجار", TAB(45), 28)**

**PrintLine(1, "شريف", TAB(12), "أبو", TAB(25), "الحيل", TAB(45), 41)**

**PrintLine(1, "محمد", TAB(12), "جلال")**

**PrintLine(1, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**FileClose(1)**

ضع هذا الكود في حدث تحميل النموذج ونفّذه، ثمّ افتح الملفّ الناتج وانظر ماذا ترى.. عامة هذا هو ما سيحتويه الملفّ:

محمد حمدي غانم 33

سامح حسن النجار 28

شريف أبو الحيل41

محمد جلال

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ولو جرّبت أن تستبدل الدالة WriteLine بالدالة PrintLine في المثال السابق، فستجد أنّ الملفّ يحتوي على السطور التالية:

"محمد", ,"حمدي", ,"غانم", ,33

"سامح", ,"حسن", ,"النجار", ,28

"شريف", ,"أبو", ,"الحيل", ,41

"محمد", ,"جلال"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

**قراءة Input:**

استخدم هذه الدالة للقراءة من الموضع الحاليّ في ملفّ الوصول المتتابع، على الصيغة التالية:

**Input(رقم الملف, متغير)**

حيث يتمّ قراءة بيانات تتناسب مع المتغيّر.. فمثلا، لو كان المتغيّر عدد صحيحا، فسيتمّ قراءة 4 وحدات 4 Bytes من الملفّ ووضعها في المتغيّر.. وطبعا سيتغيّر الموضع الحالي في الملفّ، حيث سينتقل للبايت التالي لآخر بايت تمّت قراءته.. مثال:

**Dim numVal As Long, DateVal As Date**

**Input(1, numVal)**

**Input(1, DateVal)**

**قراءة سطر LineInput:**

أرسل لهذه الدالة رقم ملفّ مفتوح، لتعيد لك سطرا كاملا (بدون علامتي نهاية السطر و بدايته)بدءا من الموضع الحاليّ في الملفّ.. مثال:

**Dim Line1 As String = LineInput(1)**

**قراءة نصّ InputString:**

تقرأ نصوصا من ملفّ وصول متتابع.. وهي على الصيغة التالية:

**النصّ المقروء = InputString(رقم الملفّ, عدد حروف النصّ)**

**العرض Width:**

استخدم هذه الدالة مع ملفّات الوصول المتتابع، لتحديد أقصى عدد من الحروف يمكن كتابته في كلّ سطر من سطور الملفّ، وذلك على الصيغة التالية:

**Width(رقم الملفّ, طول السطر)**

لاحظ أنّ السطور القصيرة لن تتأثّر، ولكن لو زاد عدد الحروف في أحد السطور عن الطول المحدّد، فسيقوم VB بإضافة سطر جديد تلقائيّا، وكتابة الحروف الزائدة به.

**الملفّات الثنائيّة Binary Files:**

يمكنك التعامل مع الملفّات الثنائيّة بنفس دوالّ ملفّات الوصول المتتابع، ما عدا بالطبع تلك التي تقرأ أو تكتب سطورا، فالملفّ الثنائيّ لا يحتوي على سطور.

ويجب تلزم الحذر عند استخدام الدالة InputString مع الملفّات الثنائيّة، ففي هذه الحالة يجب أن تبتعد عن استخدام الدالة EOF لمعرفة إذا ما كانت عمليّة القراءة قد وصلت لنهاية الملفّ أم لا، ففي الغالب سيؤدّي استخدام هذه الدجالة لحدوث خطإ.

إذن فما هو البديل؟

البديل هو أن تقارن الموضع الذي تعيده الدالة Loc، مع طول الملفّ LOF، وعلى أساس هذا تحدّد استمرار عمليّة القراءة أو توقّفها.

**ملفّات الوصول العشوائيّ Random Files:**

**كتابة في الملفّ FilePut:**

استخدم هذه الدالة لكتابة السجلات في ملفّات الوصول العشوائيّ، على الصيغة التالية:

**FilePut(رقم الملفّ, قيمة السجلّ[, رقم السجلّ])**

ولو لم تحدّد رقم السجلّ، فسيتمّ التابة في الموضع الحاليّ.. ولو كانت هناك بيانات في الموضع المحدّد، فسيتمّ الكتابة فوقها.. وبعد كتابة أيّ سجلّ، يصبح السجلّ التالي له هو الموضع الحاليّ في الملفّ.

تعالَ نرى مثالا:

افترض أنّنا نريد حفظ معلومات عن بعض المنتجات.. هذه المعلومات يعبّر عنها السجلّ التالي:

**Structure ProductRecord**

**Dim ProductID As String**

**Dim Description As String**

**Dim Price As Decimal**

**End Structure**

تعال نرى كيف نحفظ معلومات بعض المنتجات في ملفّ وصول عشوائيّ:

**' تعريف متغيّر من السجلّ**

**Dim PRec As ProductRecord**

**' ملء المتغيّر ببيانات المنتج الأوّل**

**PRec.ProductID = "TV00180-A"**

**PRec.Description = "SONY Trinitron TV"**

**PRec.Price = 799.99**

**' إنشاء وفتح ملفّ وصول عشوائي**

**FileOpen(1, "c:\products.txt",OpenMode.Random)**

**' حفظ السجلّ في الملفّ**

**' لن نحدد موضع الحفظ، لهذا سيستخدم الموضع الحاليّ من الملفّ (وهو هنا بداية الملفّ)**

**FilePut(1, Prec)**

**' يمكن ملء السجلّ ببيانات منتجات أخرى وحفظها**

**PRec = New ProductRecord()**

**PRec.ProductID = "TV810X"**

**'……………**

**FilePut(1, Prec)**

**' بعد ذلك يجب إغلاق السجلّ**

**FileClose(1)**

لاحظ ما يلي: عند فتح الملفّ لم نحدّد طول السجلّ، وذلك لأنّه يحتوي على نصوص، وهي كما تعلم متغيّرة الطول.. في الواقع إنّ هذا سيؤدّي لحدوث مأساة.. فقديما كان مبرمج VB6 يستطيع تعريف متغيرات نصيّة ذات أطوال ثابتة.. الآن لم يعد هذا متاحا بطريقة مباشرة.. للأسف، هذا معناه أنّ ملفّات الوصول العشوائيّ لا معنى لها، ففكرتها كلّها تقوم على افتراض أنّ الملفّ يتكوّن من وحدات متساوية الطول، معلوم للمبرمج طولها بالتحديد!!.. إذن فهل يعني هذا أنّ الموضوع قد انتهى؟.. نلغي ملفّات الوصول العشوائيّ من حساباتنا؟.. إطلاقا.. إنّ هناك وسيلة لتعريف نصوص ذات أطوال محدّدة، ولكن عن طريق استخدام السمات Attributes في تعريف السجلات.. في هذه الحالة، يجب أن تعرّف النصّ بأطول طول يمكن أن تحتاج إليه.. تعال نفترض أنّ اسم المنتج لن يزيد طوله عن 10 حروف، ووصف لن يزيد عن 20 حرفا.. هذا هو التعريف الجديد للسجلّ:

**Structure ProductRecord**

**<VBFixedString(10)> Dim ProductID As String**

**<VBFixedString(20)> Dim Description As String**

**Dim Price As Decimal**

**End Structure**

الآن صار للسجلّ طول ثابت، وبهذا يمكن تحديد هذا الطول عند فتح الملفّ كالتالي:

**FileOpen(1, "c:\products.txt", OpenMode.Random, , , Len(PRec))**

لاحظ أنّ المعاملات التي لم نرسلها هي معاملات اختياريّة، وسيتمّ استخدام القيم الافتراضيّة لها.. لاحظ كذلك أنّ التعبير Len(PRec) يعيد طول السجل (مجموع أطوال حقوله).. ويمكن أن تكتب الطول بنفسك لو أردت، وذلك عن طريق حساب طول السجلّ (10 + 20 + 16 = 46).

هل يقفُ بنا الأمرُ عندَ هذا الحدّ؟

لا.. ما زالت هناك إمكانيّة طريفة، وهي قدرتك على تخزين مصفوفة مرّةً واحدة بهذه الدالة.. هذه المصفوفة قد تكون مصفوفة حروف أو مصفوفة عدديّة، أو مصفوفة نصوص ثابتة الحجم، أو حتّى مصفوفة سجلات!

تعال نرى ذلك:

**Dim PRec(1) As ProductRecord**

**' ملء السجلّ الأوّل**

**PRec(0).ProductID = "TV00180-A"**

**PRec(0).Description = "SONY Trinitron TV"**

**PRec(0).Price = 799.99**

**' ملء السجلّ الثاني**

**PRec(1).ProductID = "TEL00180-A"**

**PRec(1).Description = "SONY Trinitron TEL"**

**PRec(1).Price = 99.99**

**' حساب الطول الكلّيّ للمصفوفة**

**Dim L As Integer = PRec.Length \* Len(PRec(0))**

**' إنشاء وفتح ملفّ الوصول العشوائي**

**FileOpen(1, "c:\products.txt", OpenMode.Random, , , L**)

**' كتابة المصفوفة في الملفّ**

**FilePut(1, PRec)**

**FileClose(1)**

**قراءة من الملفّ FileGet:**

هذه الدالة قرينُة الدالة السابقة، حيث تستخدم في قراءة السجلات من ملفات الوصول العشوائيّ، ولها نفس الصيغة، مع الفارق أنّ المعامل الثاني هو معامل مرجعيّ ByRef، بعد استدعاء الدالة يحتوي على القيمة العائدة من الملفّ:

**FileGet(رقم الملفّ, قيمة السجلّ[, رقم السجلّ])**

تعال نقرأ المنتجات التي أدخلناها في المثال السابق:

**Dim PRec As New ProductRecord**

**FileOpen(1, "c:\products.txt", OpenMode.Random, , , Len(PRec))**

**Do While EOF(1) = False**

**FileGet(1, PRec)**

**TextBox1.AppendText(PRec.ProductID & " \_ ")**

**TextBox1.AppendText(PRec.Description & " \_ ")**

**TextBox1.AppendText(PRec.Price & vbCrLf)**

**Loop**

**FileClose(1)**

ولو أردت الاختصار، فيمكنك قراءة السجلات في مصفوفة دفعةً واحدة.. في هذه الحالة يجب أن تعرف عدد السجلات التي يحتويها الملفّ، وذلك عن طريق قسمة طول الملفّ على طول السجل.. انظر للكود التالي:

**FileOpen(1, "c:\products.txt", OpenMode.Random)**

**Dim PRec(0) As ProductRecord**

**' حساب عدد السجلات**

**Dim L As Integer = LOF(1) / Len(PRec(0)) – 1**

**' تعديل طول المصفوفة**

**ReDim PRec(L)**

**' قراءة الملفّ كلّه في المصفوفة**

**FileGet(1, PRec)**

**FileClose(1)**

الآن يمكنك التعامل مع المصفوفة بأيّ طريقة تشاء، كأن تعرضها في مربّع نصّ كالتالي:

**Dim C As Integer**

**For C = 0 To L**

**TextBox1.AppendText(PRec(C).ProductID & " \_ ")**

**TextBox1.AppendText(PRec(C).Description & " \_ ")**

**TextBox1.AppendText(PRec(C).Price & vbCrLf)**

**Next**

وعليك أن تنتبه هنا لشيء هامّ، هو أنّ هذه الطريقة لن تكون مناسبةً عند التعامل مع ملفّ ضخمِ الحجم، لأنّ هذا معناه حجز مساحة ضخمة للمصفوفة في الذاكرة.. الأفضل إذن أن تقرأ سجلا تلو سجلّ من الملفّ.

**أداة مُراقبة نظام الملفّات FileSystemWatcher**

* + [**تعريفها:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20مُراقب%20نظام%20الملفّات\1-%20تعريفها.htm)
  + [**خصائصها:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20مُراقب%20نظام%20الملفّات\2-%20خصائصها.htm)
  + [**مشروع FileSystemWatcher:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\3-%20مُراقب%20نظام%20الملفّات\3-%20مُراقب%20نظام%20الملفّات.htm)

**تعريفها**:

تسمح لك هذه الأداة التي لا تمتلك واجهة استخدام، بمراقبة الملفّات والمجلّدات لمعرفة ما يحدث لها من تغيير (إنشاء، حذف، تغيير الاسم، تعديل محتويات ملفّ)، سواء على جهازك أو عبر شبكة أو عبر الإنترنت.. وعند حدوث التغيير المحدّد في الملفّ الذي تراقبه، تطلق هذه الأداة أحد الأحداث لإخطارك بهذا التغيير.. وهذه الأحداث هي:

محتويات الملفّ تغيّرت **Changed** ـ إنشاء مجلّد أو ملفّ **Created** ـ حذف مجلّد أو ملفّ **Deleted** ـ تغيير اسم مجلّد أو ملفّ **Renamed**.

هذا بالإضافة إلى الحدث **Error**، الذي ينطلق عند حدوث تغييرات كثيرة بدرجة تعجز معها هذه الأداة عن ملاحقتها.

ولاستخدام هذه الأداة في مشروعك، اضغط شريط المكوّنات Components Tab في صندوق الأدوات Toolbox، وانقر مرّتين بالفأرة على الأيقونة FileSystemWatcher.. ستجد أنّ نسخة من هذه الأداة قد أضيفت إلى صينية الأدوات في مشروعك.

**خصائصها:**

يمكنك أن تستخدم نافذة الخصائص لاستعراض الخصائص التالية:

**مرشّح التنبيه NotifyFilter:**

استخدم هذه الخاصيّة لتحديد نوعيّة التغييرات التي تريد مراقبتها.. ويمكن اختيار قيم هذه الخاصيّة من المرقّم IO.NotifyFilters، وهي كالتالي:

|  |  |
| --- | --- |
| Attributes | مراقبة تغيّر سمات الملفّ أو المجلّد. |
| CreationTime | مراقبة تغيّر تاريخ إنشاء الملفّ أو المجلّد. |
| DirectoryName | مراقبة تغيّر اسم المجلّد. |
| FileName | مراقبة تغيّر اسم الملفّ. |
| LastAccess | مراقبة تغيّر تاريخ آخر قراءة للملفّ أو المجلّد. |
| LastWrite | مراقبة تغيّر تاريخ آخر كتابة في الملفّ أو المجلّد. |
| Security | مراقبة تغيّر إعدادات حماية الملفّ أو المجلّد. |
| Size | مراقبة تغيّر حجم الملفّ أو المجلّد. |

وطبعا في نافذة الخصائص لا تستطيع اختيار أكثر من قيمة واحدة من هذه القيم، ولكن في الكود، يمكنك دمج قيمتين معا، باستخدام العمليّة Or:

**FSW1.NotifyFilter = IO.NotifyFilters.LastWrite Or IO.NotifyFilters.LastAccess**

**المسار Path:**

استخدم هذه الخاصيّة لتحديد مسار المجلّد الذي تريد مراقبة ما به من ملفّات.

**متضمنا المجلّدات الفرعيّة IncludeSubdirectories:**

اجعل هذه الخاصيّة True لمراقبة المجلّدات الفرعيّة التي يحتويها المجلّد الذي تراقبه.

**المرشّح Filter:**

تمكّنك هذه الخاصيّة من تحديد نوعيّة الملفات التي تريد مراقبتها، فمثلا: يمكنك مراقبة الملفّات النصّيّة بجعل قيمة هذه الخاصيّة "\*.txt" .. والقيمة الافتراضيّة لهذه الخاصيّة هي "\*.\*"، وذلك لمراقبة كلّ الملفّات من كلّ الأنواع.. فإذا أردت أن تراقب الملفات التي لا امتداد لها، فاجعل قيمة هذه الخاصيّة نصّا فارغا "".. وللأسف: لا يمكنك أن تحدّد أكثر من امتداد بهذه الخاصيّة، فإذا كان هذا حتميّا، فاستخدم نسختين من هذه الأداة، كل واحدة لمراقبة نوع ما!

**تفعيل إطلاق الأحداث EnableRaisingEvents:**

لبدء مراقبة الملفات، اجعل هذه الخاصيّة True.

**حجم مساحة التخزين InternalBufferSize:**

توضع التغييرات التي تحدث للملفّات في مخزن في الذاكرة قبل أن يتمّ إطلاق الأحداث، فإذا حدثت تغييرات أكثر وأسرع من اللازم، فإنّها تتجاوز مساحة التخزين، ممّا يؤدّى لإطلاق الحدث Error.. لهذا يمكنك باستخدام هذه الخاصيّة أن تزيد من مساحة التخزين في الذاكرة، وإن كان ذلك قد يؤثّر على أداء النظام، إذا اقتطعت مساحة ضخمة من الذاكرة.. والأفضل أن تتجنّب حدوث أيّ أخطاء، بالتقليل قدر المستطاع من عدد الملفّات والمجلّدات المراقبة، باستخدام خاصيتي المسار والمرشّح.

ويمكنك مضاعفة حجم مساحة التخزين في حدث الخطأ Error بجملة كالتالية:

**FSW1.InternalBufferSize = 2 \* FSW1.InternalBufferSize**

**مشروع FileSystemWatcher:**

في هذه المشروع، سنراقب الملفّات النصّيّة الموجود على المحرّك "\C:".



تعال نعدّ مراقب الملفّات في حدث تحميل النموذج كالآتي:

**Private Sub Form1\_Load(sender As Object, e As EventArgs) \_**

**Handles MyBase.Load**

**FileSystemWatcher1.Path = "c:\"**

**FileSystemWatcher1.IncludeSubdirectories = False**

**FileSystemWatcher1.Filter = "\*.txt"**

**FileSystemWatcher1.NotifyFilter = IO.NotifyFilters.CreationTime Or \_**

**IO.NotifyFilters.LastWrite Or IO.NotifyFilters.LastAccess Or \_**

**IO.NotifyFilters.FileName**

**FileSystemWatcher1.EnableRaisingEvents = False**

**End Sub**

ويمكن للمستخدم بدء مراقبة الملفّات بضغط زر المراقبة.. ولو ضغطه ثانية يتمّ إيقاف المراقبة.. وسنغيّر عنوانه في كلّ مرّة ليدلّ على وظيفته.. هذا هو كود الزرّ:

**Private Sub Button1\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Button1.Click**

**If Button1.Text = "بداية المراقبة" Then**

**FileSystemWatcher1.EnableRaisingEvents = True**

**Button1.Text = "إنهاء المراقبة"**

**Else**

**FileSystemWatcher1.EnableRaisingEvents = False**

**Button1.Text = "بداية المراقبة"**

**End If**

**End Sub**

ونظرا لأنّ الأحداث Changed و Created و Deleted تشترك في نفس نوع المعاملات، فإنّ بإمكانك كتابة إجراء واحد يستجيب لها كلّها معا كالتالي:

**Private Sub WatcherHandler(sender As Object, e \_**

**As IO.FileSystemEventArgs) Handles \_**

**FileSystemWatcher1.Changed, FileSystemWatcher1.Created, \_**

**FileSystemWatcher1.Deleted**

**ListBox1.Items.Add(e.ChangeType & vbTab & e.FullPath)**

**End Sub**

لاحظ أنّ المعلومات التي تخصّ الملفّ أو المجلّد ستجدها في خواصّ المعامل e، مثل نوع التغيير الذي حدث للملفّ أو المجلّد e.ChangeType، والمسار الكامل e.FullPath.. إلخ.

أمّا حدث تغيّر الاسم، فلا بدّ أن يكون له إجراء مستقلّ، وذلك لأنّ معامله e له نوع مختلف، حيث يمنحك خاصيّة OldFullPath التي تمكّنك من معرفة الاسم القديم للملفّ أو المجلّد قبل تغييره.

**Private Sub FileSystemWatcher1\_Renamed(ByVal sender As Object, \_**

**ByVal e As System.IO.RenamedEventArgs) \_**

**Handles FileSystemWatcher1.Renamed**

**ListBox1.Items.Add(e.ChangeType & vbTab & e.OldFullPath & \_**

**" TO " & e.FullPath)**

**End Sub**

وهذا هو حدث الخطأ، وفيه نوقف المراقبة حتّى نتمكّن من زيادة حجم مساحة التخزين Buffer، ثمّ نعود لمراقبة الملفات مرّة أخرى:

**Private Sub FileSystemWatcher1\_Error(ByVal sender As Object, \_**

**e As IO.ErrorEventArgs) Handles FileSystemWatcher1.Error**

**Dim status As Boolean**

**status = FileSystemWatcher1.EnableRaisingEvents**

**FileSystemWatcher1.EnableRaisingEvents = False**

**FileSystemWatcher1.InternalBufferSize = \_**

**2 \* FileSystemWatcher1.InternalBufferSize**

**FileSystemWatcher1.EnableRaisingEvents = status**

**End Sub**

أعتقد الآن أنّك قد صرت ضليعا في استخدام الملفّات.

**مربعا حوار فتح ملفّ وحفظ ملفّ**

* + [**استخدامهما:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\4-%20مربعا%20حوار%20فتح%20ملفّ%20وحفظ%20ملفّ\1-%20استخدامهما.htm)
  + [**خصائصهما:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\4-%20مربعا%20حوار%20فتح%20ملفّ%20وحفظ%20ملفّ\2-%20خصائصهما.htm)
  + [**مشروع OpenMethod:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\4-%20مربعا%20حوار%20فتح%20ملفّ%20وحفظ%20ملفّ\3-%20مشروع%20وسيلة%20الفتح.htm)
  + [**مشروع MultipleFiles:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\10-%20الملفّات\4-%20مربعا%20حوار%20فتح%20ملفّ%20وحفظ%20ملفّ\4-%20مشروع%20الملفات%20المتعددة.htm)

**استخدامهما:**

ستجد هاتين الأداتين في صندق الأدوات Tool Box باسمي OpenFileDialog و SaveFileDialog، وهما متماثلتان في معظم خصائصهما، لهذا سنتناولهما بالشرح معا..

إنّ كلتا الأداتين تمكّن المستخدم من اختيار اسم الملفّ وامتداده والمسار الذي سيُقرأ منه أو يحفظ فيه، كما تمنحان المبرمج القدرة على تحديد الامتدادات التي يُسمح للمستخدم بالتعامل معها.

انتبه إلى أنّ هاتين الأداتين لا تقومان بفتح الملفّ أو حفظه.. كلّ ما تفعلانه هو منح المستخدم واجهة مريحة، وعند إغلاقهما يمكنك قراءة اسم الملفّ الذي اختاره المستخدم، وعليك أنت أن تقوم بفتحه أو حفظه كما يحلو لك، بمساعدة الخلايا التي تعرفنا عليها في هذا الفصل.. وللتسهيل، تمنحك الأداة OpenFileDialog الوسيلة OpenFile لفتح الملفّ للقراءة فقط، وتمنحك الأداة SaveFileDialog الوسيلة SaveFile لحفظ الملفّ.

ولعرض أيّ من مربّعي الحوار، استخدم الوسيلة ShowDialog.

**خصائصهما:**

**الامتداد الافتراضيّ DefaultExtension:**

هذا هو الامتداد الذي سيظهر للمستخدم عند فتح مربّع الحوار، والذي سيتمّ حفظ الملفّ به إذا لم يتمّ اختيار غيره.. وهذه الخاصيّة تأخذ قيمة نصّيّة على غرار ".txt" أو ".bin".. إلخ.

**إضافة الامتداد AddExtension:**

إذا جعلت هذه الخاصيّة True، فإنّ الامتداد الافتراضيّ الذي حدّدته بالخاصيّة السابقة ستتمّ إضافته تلقائيّا لاسم الملفّ الذي اختاره أو كتبه المستخدم، ما لم يختر امتدادا آخر.

**تحقّ من وجود الملفّ CheckFileExists:**

إذا جعلت هذه الخاصيّة True، فإنّ مربّع الحوار يتحقّق قبل إغلاقه من أنّ الاسم الذي كتبه المستخدم هو اسم ملفّ موجود بالفعل على الجهاز، وإلا أطلق رسالة تحذير.

**تحقّق من وجود المسار CheckPathExists:**

إذا جعلت هذه الخاصيّة True، فإنّ مربّع الحوار يتحقّق قبل إغلاقه من أنّ المسار الذي كتبه المستخدم في اسم الملفّ، هو مسار موجود بالفعل على الجهاز، وإلا أطلق رسالة تحذير.

**مرجعيّة الارتباطات DereferenceLinks:**

ماذا تتوقّع أن يحدث عندما يختار المستخدم اختصارا Shortcut يشير إلى ملفّ آخر في موضع آخر على الجهاز؟.. إذا أردت أن يعيد لك مربّع الحوار اسم الاختصار فاجعل هذه الخاصيّة False، أمّا إذا أردت أن يعيد لك اسم ومسار الملفّ الذي يشير إليه الاختصار، فاجعل هذه الخاصيّة True.

**اسم الملفّ FileName:**

يمكنك أن تضع في هذه الخاصيّة اسم الملفّ الذي تريد عرضه للمستخدم عند فتح مربع الحوار، كما يمكنك أن تحصل منها على اسم الملفّ الذي اختاره المستخدم بعد إغلاق مربّع الحوار.

**المرشّح Filter:**

يمكنك بهذه الخاصيّة أن تحدّد نوعيّة الملفّات التي تظهر في قائمة نوع الملفّات في مربّع الحوار.. فمثلا: لعرض ملفّات النصوص فقط، استخدم الجملة التالية:

**Dialog.Filter = "Text files|\*.txt"**

لاحظ أنّ العلامة "|" تفصل بين امتداد الملفّ ووصفه (ذلك الذي يظهر للمستخدم في قائمة أنواع الملفّات).

ومن الممكن أن تعرض أكثر من امتداد لنفس النوع، فمثلا: يمكن أن يكون ملفّ النصوص من النوع وثيقة ".doc" بالإضافة إلى النوع ".txt".. في هذه الحالة يمكنك أن تستخدم جملة كالتالية:

**Dialog.Filter = "Text files|\*.txt;\*.doc"**

لاحظ أنّ باستطاعتك إضافة أيّ عدد من الامتدادات مع الفصل بين كلّ منها بفاصلة منقوطة ";".

وفي الغالب تحتاج لعرض أنواع مختلفة من الملفّات، مثل ملفّات النصوص والصور.. في هذه الحالة استخدم الجملة التالية:

**Dialog.Filter = "Text files|\*.txt;\*.doc|Images|\*.BMP;\*.GIF;\*.JPG"**

حيث استخدمنا العلامة "|" أيضا للفصل بين النوعين.. لاحظ أنّنا لم نضع أيّ مسافات في هذا النصّ، وإلا فإنّها ستظهر للمستخدم أو سيعتبرها مربّع الحوار جزءا من الامتداد.

وأحيانا تضع للمستخدم اختيارا لعرض كلّ الملفّات.. افترض أنّنا نريد أن نعرض للمستخدم في قائمة أنواع الملفّات العناصر التالية:

Text files

Images

Texts & Images

All files

في هذه الحالة استخدم الجملة التالية:

**Dialog.Filter = "Text files|\*.txt;\*.doc" & \_**

**"|Images|\*.BMP;\*.GIF;\*.JPG" & \_**

**"|Texts & Images|\*.txt;\*.doc;\*.BMP;\*.GIF;\*.JPG" & \_**

**"|All files|\*.\*"**

ولكن في معظم الأحيان، تلاحظ أنّ قائمة أنواع الملفّات تعرض نوع الملفّ وبجواره امتداداته، مثل:

Text files(\*.txt, \*.doc)

Images(\*.BMP, \*.GIF, \*.JPG)

Texts & Images(\*.txt, \*.doc, \*.BMP, \*.GIF, \*.JPG)

All files(\*.\*)

في هذه الحالة كلّ ما عليك هو تطوير الوصف لإضافة الامتداد بجواره كالتالي:

**Dialog.Filter = "Text files(\*.txt, \*.doc)|\*.txt;\*.doc" & \_**

**"|Images(\*.BMP, \*.GIF, \*.JPG)|\*.BMP;\*.GIF;\*.JPG" & \_**

**"|Texts & Images(\*.txt, \*.doc, \*.BMP, \*.GIF, \*.JPG)" & \_**

**"|\*.txt;\*.doc;\*.BMP;\*.GIF;\*.JPG|All files(\*.\*)|\*.\*"**

**رقم المرشّح FilterIndex:**

إذا أردت أن تختار نوعا من أنواع الملفّات بحيث يكون معروضا للمستخدم عند فتح مربّع الحوار، فاستخدم هذه الخاصيّة.. فمثلا لو افترضنا أنّ قائمة الأنواع تحتوي على الأنواع الأربعة السابقة، فيمكنك أن تعرض للمستخدم ملفّات الصور Images عن طريق جعل هذه الخاصيّة = 2.

**المجلّد المبدئيّ InitialDirectory:**

استخدم هذه الخاصيّة لتحديد المجلّد الذي تريد عرض محتوياته للمستخدم عندما يفتح مربّع الحوار.

فمثلا لجعل هذا المجلّد هو مجلّد البرنامج، استخدم الجملة التالية:

**OpenFileDialog1.InitialDirectory = Application.ExecutablePath**

**احتفظ بالمجلّد RestoreDirectory:**

لو جعلت هذه الخاصيّة True، فسيعرض مربّع الحوار آخر مجلّد كان به المستخدم في المرّة السابقة، ما لم يتمّ تحديد المجلّد المبدئيّ بالخاصيّة السابقة.

**تحقّق من الأسماء ValidateNames:**

إذا كانت هذه الخاصيّة True (وهي القيمة الافتراضيّة ولا يجب تغييرها)، فسيتحقّق مربّع الحوار من صلاحية الاسم الذي يدخله المستخدم، ومن خلوّه من الحروف غير المسموح بها، قبل إغلاق مربّع الحوار.

بعض الخصائص الخاصّة فقط بمربّع حوار "فتح ملفّ":

**متعدّد الاختيار MultiSelect:**

لو جعلت هذه الخاصيّة True، فسيتمكّن المستخدم من اختيار أكثر من ملفّ من مربعّ فتح ملفّ، عن طريق ضغط هذه الملفّات بالفأرة مع ضغط زر Ctrl من لوحة المفاتيح.. وطبعا يجب أن تكون هذه الملفّات المختارة كلّها في نفس المجلّد.

**أسماء الملفّات FileNames:**

ترجع هذه الخاصيّة مصفوفةً نصّيّة تحتوي على كلّ أسماء الملفّات التي اختارها المستخدم في مربّع فتح ملفّ.

**عرض للقراءة فقط ShowReadOnly:**

إذا جعلت هذه الخاصيّة True، فسيتمّ عرض مربّع اختيار للقراءة فقط على مربّع الحوار.

**اختيار للقراءة فقط ReadOnlyChecked:**

إذا جعلت هذه الخاصيّة True، فسيتّم وضع علامة (✓) على مربّع اختيار للقراءة فقط عند فتح مربّع الحوار.. كما يمكنك بعد إغلاق المستخدم لمربّع الحوار، أن تستخدم هذه الخاصيّة لمعرفة إذا ما كان المستخدم قد اختار فتح الملفّ للقراءة فقط أم لا، فإذا كان الأمر كذلك، فيجب عليك أن تفتح الملفّ للقراءة فقط، ولا تسمح بتغيير محتوياته.

**مشروع OpenMethod:**

هذه المشروع يريك كيف تستخدم الوسيلة OpenFile لفتح ملفّ يختاره المستخدم، وعرض محتوياته في مربّع نصّ.

**' اعرض مربّع حوار فتح ملفّ**

**If OpenFileDialog1.ShowDialog() = DialogResult.Cancel Then Exit Sub**

**' فتح الملفّ الذي اختاره المستخدم في مربّع الحوار**

**Dim str As System.IO.Stream = OpenFileDialog1.OpenFile**

**Dim L As Integer = str.Length ' طول الملفّ**

**Dim buffer(L) As Byte**

**' ستتم قراءة الملفّ على أنّه ثنائيّ**

**If str.Read(buffer, 0, L) = 0 Then Exit Sub**

**Dim txt As New System.Text.StringBuilder()**

**Dim i As Integer**

**' يجب تحويل كلّ رقم إلى الحرف المناظر له**

**For i = 0 To L**

**txt.Append(Chr(buffer(i)))**

**Next**

**str.Close()**

**TextBox1.Text = txt.ToString**

**مشروع MultipleFiles:**

في هذا المشروع نرى كيف يختار المستخدم أكثر من ملفّ، وكيف نتعامل مع اختياراته، حيث سنعرض أسماء الملفّات التي اختارها في قائمة ListBox.

لا تنسَ أن تجعل لخاصيّة MultiSelect في مربّع حوار فتح الملفّ القيمة True.. اكتب الكود التالي في حدث ضغط الزرّ:

**OpenFileDialog1.Multiselect = True**

**' عرض مربّع الحوار**

**If OpenFileDialog1.ShowDialog() = DialogResult.Cancel Then Exit Sub**

**' إفراغ القائمة من القيم السابقة**

**ListBox1.Items.Clear()**

**Dim filesEnum As IEnumerator = OpenFileDialog1.FileNames.GetEnumerator()**

**While filesEnum.MoveNext**

**ListBox1.Items.Add(filesEnum.Current)**

**End While**

ولمزيد من التدريب على استخدام هاتين الأداتين، ارجع لمشروع TextPad في مجلّد برامج الفصل السادس، وانظر كيف استخدمنا هاتين الأداتين للقيام بحفظ محتويات مربّع النصّ وتحميلها، باستخدام أوامر القائمة "ملفّ" File: حفظ Save وحفظ باسم Save As وفتح Open.