

## الفصل الرابع عشر المزيج من الملفات

تعرفنا في الفصل السابق على كيفية التعامل مع نظام الملفات الموجود بالفعل وفي هذا الفصل سنتعرف على كيفية إنشاء الملفات والكتابة فيها أو القراءة منها إلى جانب استخدام الدوال التقليدية للتعامل مع الملفات.

بانتهاء هذا الفصل سنتعرف على:

- العمل مع جداول الملفات.
- العمل مع الملفات النصية.
- العمل مع الملفات الثنائية.
- العمل مع ملفات XML.
- دوال الملفات القياسية.
- استخدام ملف التسجيل Registry.
- مفهوم ملفات INI.

يعتبر هذا الفصل استكمالاً للفصل السابق، وفيه نتناول مفهوم جداول الملفات **File Streams** الذى يمكنك استخدامه فى التعامل مع الأنواع المختلفة من البيانات وكذلك استخدام الدوال التقليدية للتعامل مع الملفات والتي كانت موجودة بإصدارات **Visual Basic** القديمة. كما سنتعرف أيضاً على كيفية الوصول إلى ملف تسجيل نظام التشغيل **Windows Registry** الذى يحتوى على إعدادات التطبيقات وغيرها من البيانات الأخرى، هذا إلى جانب التعامل مع ملفات **INI**.

### العمل مع جداول الملفات

يحتوى كل ملف من الملفات المخزنة على حاسبك على سلسلة من الحروف، حيث يمكن تقسيم هذه الملفات إلى ثلاثة أنواع أساسية هي:

- الملفات النصية **Text Files** التى تحتوى دائماً على نصوص قابلة للقراءة من قبل المستخدم العادى والتي يمكنك تعديلها من خلال أي من برامج تحرير النصوص كبرنامج **Notepad** مثلاً.
  - الملفات الثنائية **Binary Files** والتي تحتوى على البيانات المستخدمة من قبل البرامج، أى لا يستطيع المستخدم العادى التعرف على محتوياتها. فإذا حاولت فتح أحد الملفات الثنائية (وليكن ملف تنفيذى بالامتداد **.EXE**) داخل برنامج **Notepad**، فلن تتمكن من التعرف على محتويات هذا الملف، حيث يحتوى كل حرف بداخله على بايت من البيانات المعروفة من قبل البرنامج الذى يقوم باستخدامه. ففي حالة الملفات التنفيذية، يكون نظام التشغيل هو المسئول الأول عن نداء هذا الملف واستخدامه وفهم محتوياته.
  - ملفات **XML** وتحتوى هذه الملفات على بيانات مركبة تركيباً محكماً وفى نفس الوقت تكون قابلة للقراءة والفهم والتعديل من قبل المستخدم العادى.
- يحتوى نطاق **.NET** على بعض التصنيفات الجديدة التى تم تصميمها خصيصاً للعمل مع جداول الملفات **File Streams** حيث يمكنك تصور الملفات المخزنة على حاسبك كما لو

كانت جداول أو سلاسل من البيانات قابلة للتخزين في الذاكرة ووحدات التخزين على حدٍ سواء، كما يمكنك الوصول إليها واستخدامها من خلال الشبكة المتصل بها حاسبك.

### العمل مع الملفات النصية

يمكنك التعامل مع الملفات النصية داخل نطاق .NET. سواءً بالقراءة أو الكتابة من خلال التصنيف `StreamReader` والتصنيف `StreamWriter` على الترتيب، حيث تم تصميم التصنيفين للتعامل مع هذه الملفات في صورة سلسلة `Sequential`. يمكنك معالجة البيانات الموجودة داخل الملف التسلسلي سطر بسطر أو بقراءة عدد معين من الحروف. يوضح الكود التالي كيفية قراءة كل سطر داخل الملف النصي وطباعته على النافذة الفورية `Immediate Window`:

```
Dim TestFileAsStreamReader
Dim LineofTextAs String
TestFile = File.OpenText(Directory.GetCurrentDirectory() &
"\myfile.txt")
Do
LineofText = TestFile.ReadLine()
Debug.WriteLine(LineofText)
Loop Until LineofText Is Nothing
TestFile.Close()
```

وكما ترى مدى سهولة هذا الكود، حيث نقوم أولاً بفتح الملف ثم استدعاء الوظيفة `ReadLine()` باستمرار حتى يتم قراءة محتويات الملف بالكامل. يمكنك أيضاً استخدام الوظيفة `WriteLine()` لكتابة سطر من النصوص إلى أحد الملفات. يوضح الكود التالي كيفية قراءة كل سطر من الملف `myfile.txt` ثم كتابته بحروف كبيرة إلى ملف نصي آخر بعنوان `output.txt`.

```
Dim TestFileAsStreamReader
Dim OutputFile As StreamWriter
Dim LineofText As String
TestFile = File.OpenText(Directory.GetCurrentDirectory() &
"\myfile.txt")
OutputFile = File.AppendText(Directory.GetCurrentDirectory() &
"\output.txt")
```

```
While TestFile.Peek() <> -1
    LineOfText = TestFile.ReadLine()
    OutputFile.WriteLine(LineOfText.ToUpper)
End While
TestFile.Close()
OutputFile.Close()
```

بالنظر إلى هذا الكود، تلاحظ وجود اختلافين أساسيين بين عملية قراءة الملف النصي وكتابته، فقد قمنا أولاً بفتح الملف من خلال الوظيفة `File.AppendText()` بدلاً من الوظيفة `File.OpenText()` المستخدمة عند القراءة من الملف، حيث تعني كلمة `Append` الإضافة إلى الملف، لذا يمكنك استخدام الوظيفة `AppendText()` لإضافة النصوص إلى ملف موجود بالفعل أو إنشاء ملف جديد من البداية. أما إذا أردت حذف محتويات الملف الموجودة مسبقاً قبل عملية الكتابة، فيمكنك في هذه الحالة استخدام الوظيفة `CreateText()` كما في الكود التالي:

```
TestFile = File.CreateText("c:\output.txt")
```

أما الاختلاف الثاني فيمكن في استخدام الوظيفة `Peek()` داخل الدوارة `While`، حيث تقوم هذه الوظيفة بإرجاع كود الحرف التالي داخل الملف دون أن تقوم بقراءته ومن ثم فهي تقوم بإرجاع القيمة -1 بمجرد الوصول إلى نهاية الملف الذي تقوم بالقراءة منه.

### العمل مع الملفات الثنائية

يقوم نظام التشغيل باعتبار أي ملف من الملفات مجموعة من الحروف التي يمثل كل منها بايت داخل الملف. وقد صُمم التصنيف `StreamReader` والتصنيف `StreamWriter` للتعامل مع هذه الحروف كبيانات نصية. ولكن ربما لا يمثل البايث الموجود بالملف دائماً حرف من الحروف وإنما يمثل رقم معين من الأرقام العديدة. يحتوي نطاق `.NET` على التصنيف `BinaryReader` والتصنيف `BinaryWriter` التي تتيح لك الوصول المباشر إلى البيانات داخل الملفات بدلاً من الطريقة المتتالية (المسلسلة) المستخدمة في حالة الملفات النصية. يوضح جدول ١٤-١ التالي أهم الوظائف الموجودة بالتصنيفين السابقين.

جدول ١٤-١ أهم وظائف التصنيفين BinaryWriter, BinaryReader

| اسم الوظيفة | نوع البيانات        |
|-------------|---------------------|
| Read        | Bytes-Characters    |
| ReadBoolean | Boolean             |
| ReadByte    | Byte                |
| ReadBytes   | Array of Bytes      |
| ReadChar    | Character           |
| ReadChars   | Array of Characters |
| ReadString  | String              |
| ReadDecimal | Decimal             |
| ReadInt16   | Short               |
| ReadInt32   | Integer             |
| ReadInt64   | Long                |

لتوضيح طريقة استخدام التصنيف BinaryReader، سنقوم بقراءة ملف نصي باستخدام نوع البيانات Byte وقراءة ١٠٠٠ بايت في المرة الواحدة، حيث يمكنك معالجة الملف من خلال بايت واحد في كل مرة أو باختيار حجم معين، كما يتم تحويل كل بايت إلى الحرف ASCII المناظر كما في الكود التالي:

```
Const BLOCK_SIZE As Integer = 100
```

```
Dim TestFile As BinaryReader  
Dim BytesRead As Integer  
Dim ByteValues(BLOCK_SIZE) As Byte  
Dim StringValue As String  
Dim ASCIIConverter As System.Text.ASCIIEncoding
```

فتح الملف للقراءة'

```
TestFile = New BinaryReader(File.Open("myfile.txt",  
IO.FileMode.Open, IO.FileAccess.Read))
```

قراءة الملف وإنشاء السلسلة النصية'

```
BytesRead = TestFile.Read(ByteValues, 0, BLOCK_SIZE)  
ASCIIConverter = New System.Text.ASCIIEncoding()  
While BytesRead > 0
```

```
StringValue = ASCIIConverter.GetString(ByteValues, 0,
BytesRead)
txtMessage.Text&= StringValue
Array.Clear(ByteValues, 0, BLOCK_SIZE)
BytesRead = TestFile.Read(ByteValues, 0, BLOCK_SIZE)
End While
```

```
TestFile.Close()
```

حيث `txtMessage` عبارة عن مربع النص الذي يتم من خلاله عرض محتويات الملف `.myFile.txt`

### العمل مع ملفات XML

يعتبر تنسيق XML من السمات الأساسية قلباً وقالباً داخل .NET. كما سنعرف تفصيلاً في الجزء الثالث من هذه السلسلة. فعلى الرغم من استخدام XML لتخزين البيانات المركبة، إلا أنه يمكنك استخدام هذا التنسيق في كتابة الملفات النصية المستخدمة في الأغراض المختلفة. ولعل أحد الأمثلة على استخدام ملفات XML يتمثل في الاحتفاظ بمتغيرات المرحلة `Session Variables` بين مجموعة من نماذج الويب أو ما نطلق عليه `Web Farm`، حيث يمكنك تخزين هذه المتغيرات داخل ملف واحد بتنسيق XML وتخزينه مع ملف أو ملفات قاعدة البيانات.

### حفظ البيانات بتنسيق XML

على فرض أننا بصدد تخزين بيانات عن العملاء باستخدام التصنيفات التالية:

```
Private Class OrderItem
    Public ItemName As String
    Public UnitPrice As Decimal
    Public Quantity As Integer
End class
```

```
Private Class Customer
    public Name As String
    Public ID As Integer
    Public OrderOtems As ArrayList
    Sub New()
```

```
OrderItems = New ArrayList()
End Sub
End Class
```

في هذا الكود، يقوم التصنيف **Customer** بوصف كل عميل من العملاء من خلال اسمه ورقمه وقائمة بالعناصر التي يتكون منها طلب الصرف الخاص بهذا العميل. كما يحتوي كل عنصر داخل طلب الصرف على اسم هذا العنصر وسعره وكميته. على فرض أن لدينا مصفوفة من العملاء ونرغب في إنشاء ملف **XML** لتخزين كل هذه المعلومات. سنقوم بأداء ذلك من خلال التصنيف **XMLDocument** الذي يتعامل مع بيانات **XML** بالذاكرة والتصنيف **XMLWriter** الذي يقوم بكتابة البيانات إلى ملف **XML** الناتج. وذلك من خلال الكود التالي:

```
Import System.XML
.....
Dim cusTemp As Customer ' عميل مؤقت
Dim xmlDoc As New XmlDocument()
Dim rootNode As XmlNode
Dim tempCustomer, tempCustomerNode As XmlNode
Dim tempOrder, tempOrderNode As XmlNode
Dim itemListNode As XmlNode
Dim i, j As Integer

إنشاء عقدة عميل خالية لاستخدامها فيما بعد
tempCustomer = xmlDoc.CreateNode(XmlNodeType.Element,
"Customer", "")
tempCustomer.Attributes.Append(xmlDoc.CreateAttribute("ID"))

إنشاء عقدة خالية لصف جديد لاستخدامها فيما بعد
tempOrder = xmlDoc.CreateNode(XmlNodeType.Element,
"OrderItem", "")
tempOrder.Attributes.Append(xmlDoc.CreateAttribute
("UnitPrice"))
tempOrder.Attributes.Append(xmlDoc.CreateAttribute
("Quantity"))
tempOrder.Attributes.Append(xmlDoc.CreateAttribute
```

("ItemName"))

إنشاء العقدة الرئيسية للملف'

Create the root node of the XML document

```
RootNode = xmlDoc.CreateNode(XmlNodeType.Element,  
"Customers", "")
```

```
xmlDoc.AppendChild(RootNode)
```

For i = 0 To 39

```
cusTemp = New Customer)
```

```
cusTemp.ID = i
```

إنشاء عقدة عميل وتعيين الرقم والاسم'

```
TempCustomerNode = TempCustomer.Clone()
```

```
TempCustomerNode.InnerText = cusTemp.Name
```

```
TempCustomerNode.Attributes("ID").Value =  
cusTemp.ID.ToString
```

إضافة أصناف الطلبات إن وجدت إلى عقدة العميل'

```
If cusTemp.OrderItems.Count > 0 Then
```

```
ItemListNode = xmlDoc.CreateNode(XmlNodeType.  
Element, "OrderItems", "")
```

```
For j = 0 To cusTemp.OrderItems.Count - 1
```

```
TempOrderNode = TempOrder.Clone()
```

```
TempOrderNode.Attributes("Quantity").Value =
```

```
CType(cusTemp.OrderItems(j), OrderItem).Quantity.  
ToString
```

```
TempOrderNode.Attributes("UnitPrice").Value =
```

```
CType(cusTemp.OrderItems(j), OrderItem).UnitPrice.  
ToString
```

```
TempOrderNode.Attributes("ItemName").Value =
```

```
CType(cusTemp.OrderItems(j), OrderItem).ItemName  
ItemListNode.AppendChild(TempOrderNode)
```

```
Next j
```

```
TempCustomerNode.AppendChild(ItemListNode)
```

```
End If
```

إضافة عقدة العميل إلى ملف XML'

```
xmlDoc.DocumentElement.AppendChild  
(TempCustomerNode)
```

```
Next
```

كتابة الكائن إلى الملف'

```
Dim writer As New XmlTextWriter(Path.Combine(Application.StartupPath(), "output.xml"), System.Text.ASCIIEncoding.ASCII)
writer.Formatting = Formatting.Indented
xmlDoc.WriteContentTo(writer)
writer.Close()
MessageBox.Show("XML File has been created.")
```

حيث يتكون ملف XML من مجموعة من العقد Nodes والتي يتوى كل منها بدوره على عنصر أبوى Parent وعنصر وليد Child. قم بتنفيذ هذا الكود تحصل على الملف output.xml الذى يمكنك عرض محتوياته بمستعرض الويب.

### قراءة محتويات ملف XML

يحتوى التصنيف XmlDocument على العديد من الوظائف المستخدمة لقراءة محتويات ملف XML. فعلى سبيل المثال، يمكنك قراءة محتويات الملف output.xml الذى أنشأناه منذ قليل باستخدام سطور الكود التالية:

```
Dim xmlDoc As New XmlDocument()
Dim tempnode As XmlNode
xmlDoc.Load(".\debug\output.xml")
For Each tempnode In xmlDoc.DocumentElement.ChildNodes
    Debug.WriteLine("Customer " & tempnode.InnerText & "
        has " & TempNode.LastChild.ChildNodes.Count & "
        orders.")
Next
```

وفي هذا الكود يتم استخدام الدوارة For Each للوصول إلى كل عقدة وليدة داخل العقدة الأصلية.

### دوال الملفات القياسية

تعرفنا حتى الآن على كيفية الوصول إلى نظام الملفات من خلال التصنيفات الجديدة التى يدعمها نطاق .NET. وعلى الرغم من ذلك هناك مجموعة من الدوال القياسية التى كانت موجودة بالإصدارات السابقة من Visual Basic. فإذا كنت كثير العمل مع كود هذه

الإصدارات، يمكنك استخدام هذه الدوال التي لم تزل مدعومة داخل .NET. على الرغم من وجود بعض التعديلات الطفيفة في صيغة هذه الدوال. ولكن، يُنصح باستخدام وظائف التصنيفات الجديدة الموجودة داخل نطاق .NET. إلا إذا كنت تعمل مع أجزاء من الكود أو التطبيقات القديمة. وكى تتمكن من استخدام هذه الدوال، قم أولاً بتضمين المسمى **FileSystem** من خلال عبارة **Imports** كما يلي:

```
Imports Microsoft.VisualBasic.FileSystem
```

### البحث عن الملفات

تعتبر الوظائف **Dir** من الوظائف المفيدة والتي تستخدم للبحث عن الملفات مثل الأمر **Dir** في **MS-DOS**. يمكن استخدام هذه الوظيفة لإرجاع قائمة بملف أو أكثر من ملفات النظام التي تحقق شروط معينة مثل المسار **C:\\*.BAT** الذي يمثل كل الملفات ذات الامتداد **BAT** في الدليل الرئيسى للقرص **C**. والصيغة العامة للوظيفة **Dir** كما يلي:

```
stringvar = Dir(path [,attributes])
```

وأحد استخدامات الوظيفة **Dir** يتمثل في تحديد وجود ملف معين من عدمه وذلك حتى نحاول قراءته مما يتسبب في طرح أحد الاستثناءات، وذلك كما في الكود التالي:

```
Public Function bFileExists(sFile As String) As Boolean
    If Dir(sFile) <> " " Then
        bFileExists = True
    Else
        bFileExists = False
    End If
End Function
```

تقوم الوظيفة **Dir** بإرجاع اسم الملف إذا كان موجوداً. فإذا لم يتم العثور عليه، تقوم الوظيفة بإرجاع القيمة "".

إذا أردت التأكد من وجود الملف دون استخدام الوظيفة **Dir**، يمكنك كتابة الكود اللازم لذلك بنفسك كما في الدالة التالية:

```
Public Function bFileExists(sFile As String) As Boolean
    If Dir(sFile) <> " " Then
        bFileExists = True
    Else
```

```
bFileExists =False
End If
End Function
```

### سرد الملفات والمجلدات

أحد الاستخدامات الأخرى للدالة (**Dir()**)، إرجاع قائمة بالملفات الموجودة داخل مسار معين. فعند استخدام الأمر **Dir** في **MS-DOS**، يتم عرض الملفات على الشاشة. ولأن الدالة (**Dir()**) معدة لإرجاع قيمة حرفية واحدة، فلا بد من تكرار استدعاء هذه الدالة للحصول على الملفات الموجودة في المسار المحدد واحداً وراء الآخر. لنفرض أن المجلد **C:\Data** به مجموعة من ملفات الصور ذات الامتداد **.bmp**. والتي يتم تنفيذها حينئذٍ بالمسار **C:\Data\*.bmp**. يمكنك استخدام الكود التالي للحصول على أسماء هذه الملفات ووضعها داخل مربع سرد بالنموذج.

```
Dim sNextFile As String
sNextFile = Dir("C:\Data*.bmp")
```

```
While sNextFile <> ""
    lstPictureList.Items.Add(sNextFile)
    sNextFile = Dir()
End While
```

وكما ترى، فقد تم ذكر المسار في أول استدعاء للدالة (**Dir()**)، أما عمليات الاستدعاء التالية فتم فيها ذكر الدالة (**Dir()**) فقط دون تضمين أية معاملات، مما يعني استخدام المسار السابق مع التقدم نحو الملف التالي في المجلد. وبمجرد انتهاء الملفات الموجودة في المجلد، تقوم الدالة (**Dir()**) بإرجاع القيمة الخالية "" وهذا هو الشرط الذي يتم به الخروج من الدوارة.

المعامل الثاني للدالة **Dir** معامل اختياري ويستخدم لتحديد الصفات **Attributes** المطلوبة في الملفات التي يتم إرجاع أسمائها. فعلى سبيل المثال، يمكنك استخدام الصفة **FileAttribute.Directory** لإرجاع أسماء المجلدات فقط، بينما تستخدم الصفة **FileAttribute.Volume** لإرجاع اسم القرص. فمثلاً يمكنك تعديل الكود السابق لإرجاع المجلدات الفرعية بالإضافة إلى الملفات ذات الامتداد **.bmp**. هكذا:

```
Dim sNextFile As String
```

```
sNextFile = Dir("C:\Data\*.bmp", FileAttribute.Directory)
While sNextFile <> ""
    lstPictureList.Items.Add(sNextFile)
    sNextFile = Dir()
End While
```

يوضح جدول ١٤-٢ التالي الخيارات المتاحة للمعامل الثاني داخل الدالة Dir().

جدول ١٤-٢ خيارات الصفة FileAttribute

| التأثير                                  | الصفة                   |
|--|-------------------------|
| القيمة الافتراضية، أى إرجاع جميع الملفات | FileAttribute.Normal    |
| إظهار الملفات المخفية                    | FileAttribute.Hidden    |
| إظهار الملفات الخاصة بالنظام             | FileAttribute.System    |
| إظهار اسم القرص                          | FileAttribute.Volume    |
| إظهار المجلدات الفرعية                   | FileAttribute.Directory |
| إظهار الملفات المعلمة للقراءة فقط        | FileAttribute.ReadOnly  |
| إظهار الملفات الأرشيفية                  | FileAttribute.Archive   |

دوال التعامل مع الملفات

تتشابه كثير من دوال التعامل مع الملفات في Visual Basic مع نظائرها في MS-DOS، كما يبين جدول ١٤-٣ التالي.

جدول ١٤-٣ دوال التعامل مع الملفات

| الصيغة                        | العملية             |
|-------------------------------|---------------------|
| FileCopsy <i>source, dest</i> | نسخ ملف             |
| Kill <i>path</i>              | محو ملف أو أكثر     |
| MkDir <i>pathname</i>         | إنشاء مجلد جديد     |
| Rmdir <i>pathname</i>         | محو مجلد خالي       |
| ChDir <i>pathname</i>         | تغيير الدليل الحالى |

| الصيغة  | العملية            |
|---------|--------------------|
| ChDrive | تغيير القرص الحالي |

وفيما يلي نوضح استخدام كل دالة من هذه الدوال.

*نسخ ملف*

ينقص الدالة (`FileCopy()`) عن نظيرتها في `MS-DOS` عدم القدرة على استخدام الرمز `VB` الشاملين وهما `*`، `?` (`Wildcards`) لتحديد أكثر من ملف. يقوم المثال التالي بنسخ ملف النص `VbSample.TXT` إلى الملف `Sample.TXT` الموجود تحت المجلد `VB` في نفس القرص:

```
FileCopy("C:\VBSample.txt", "C:\VB\Samples.txt")
```

تقوم الوظيفة السابقة بالكتابة فوق الملف إن وجد، ما لم يكن محمياً ضد الكتابة أو كان مفتوحاً من قِبَل تطبيق آخر.

إذا كنت تقوم بعمليات تستغرق وقتاً طويلاً مثل عملية النسخ، يستحسن أن تقوم بجذب انتباه المستخدم من خلال أي صورة متحركة، مثلما يفعل مستكشف `Windows` عند النسخ.



*محو ملف*

يمكنك استخدام الرمز الشاملين `Wildcards` مع الدالة (`Kill`) كما في الأمر `Del` في `MS-DOS`. انظر المثال التالي:

```
Kill("C:\Test\*.txt")
```

*إعادة تسمية ملف*

يمكنك استخدام الوظيفة (`File.Move()`)، الصيغة العامة لهذه الدالة كالتالي:

```
File.Move(oldname,newname)
```

كما يمكن أيضاً أن تستخدم هذه الوظيفة لنقل ملف مثل الوظيفة (`Move()`) في `MS-DOS` كما يلي (يجب استخدام العبارة `Imports System.IO` في بداية الكود):

```
MkDir("C:\Docs")
```

```
File.Move("C:\Test.doc" As "C:\Docs\Test.doc")
```

*تحديد الدليل الحالي*

لتسهيل التعامل مع الملفات وبدلاً من كتابة اسم المسار بالكامل، يمكن تحديد الدليل الحالي **Current Directory**، بحيث لا نحتاج لكتابة المسار الكامل في كل مرة، وذلك من خلال الدالة **ChDir()** مقرونةً بالدالة **ChDrive()** كما في المثال التالي:

```
chdir("c:\VB\")
chdrive("c: ")
File.Move("Sample.txt" , "Test.txt")
Chdrive("e: ")
Chdir("e:\Samples")
Kill ("data.bak")
```

تم استبدال الدوال **Name()** و **App.Path()** الموجودة بالإصدارات القديمة بالوظائف **File.Move()** و **Application.StartupPath()** على الترتيب.



### العمل مع الملفات النصية

يمكنك من خلال كود **Visual Basic** القراءة من الملفات النصية أو الكتابة فيها حيث تعمل هذه الملفات كما ذكرنا من قبل بالطريقة المتتابعة **Sequential**. ولكن قبل أن تقوم بعملية القراءة أو الكتابة، يجب أن تقوم أولاً بفتح الملف من خلال الدالة **FileOpen()** التي تقوم بدورها بربط الملف المطلوب برقم صحيح يمكن استخدامه في التعامل مع هذا الملف من خلال السطور المتبقية من الكود كما يلي:

```
FileOpen(1,"C:\myfile.txt",OpenMode.Input)
```

ففي هذا الكود يتم فتح الملف **C:\myfile.txt** مع تخصيصه بالرقم 1، ولأننا نرغب في القراءة من الملف، فقد قمنا بتخصيص القيمة **OpenMode.Input** للمعامل **Mode** (المعامل الثالث بالدالة).

وقد قمنا في الكود السابق بتعيين رقم صريح للملف على فرض أننا سنتعامل مع هذا الملف فقط من خلال الكود. أما إذا كنت تنوى فتح وإغلاق ملفات متعددة أثناء عملك، فمن الأفضل أن تترك تعيين هذا الرقم للدالة **FreeFile()** التي تقوم بإرجاع الرقم المتاح التالي كما في الكود التالي:

```
Dim FileNumAs Integer
FileNum = FreeFile()
```

```
FileOpen(FileNum, "C:\myfile.txt", OpenMode.Input)
```

يوضح الكود التالي استخدام الدالة `LineInput()` داخل الدوارة `While` لقراءة محتويات الملف بمقدار سطر واحد في الدورة الواحدة ثم كتابة محتويات هذا السطر إلى مربع سرد موجود بالنموذج:

```
Dim FileNumAs Integer  
Dim TempString As String
```

```
FileNum = FreeFile()  
FileOpen(FileNum, "C:\myfile.txt", OpenMode.Input)  
While Not EOF(FileNum)  
    TempString = LineInput(fileNum)  
    lstFiles.Items.Add(TempString)  
End While  
FileClose(FileNum)
```

ولأن محاولة قراءة بيانات غير موجودة بالملف ينتج عنها خطأ، فقد قمنا باختبار نهاية الملف من خلال الوظيفة `EOF` قبل أن نقوم بعملية القراءة.

إذا كنت من مستخدمي الإصدارات القديمة من **Visual Basic**، ستلاحظ تغيير واضح في أوامر التعامل مع الملفات. فعلى سبيل المثال، تم استبدال العبارة `Line Input` بالدالة `LineInput()` كما أن الصيغة التالية للدالة `Open`:  
`Open "C:\myfile.txt" for Input As #1`  
لم تعد تعمل داخل **Visual Basic 2012**.



وبعد أن تقوم بفتح الملف، يمكنك اختيار الوظيفة المناسبة لقراءة البيانات من هذا الملف. ففي المثال السابق، قمنا باستخدام الدالة `LineInput()` على اعتبار أن كل سطر بالملف يمثل جزءاً مستقلاً بذاته داخل هذا الملف. ولكن ربما أردت أحياناً قراءة جزء معين من السطر أو تخزين أكثر من جزء داخل نفس السطر، وفي هذه الحالة يمكنك استخدام الدالة `Input()` بدلاً من الدالة `LineInput()`.

تستخدم الدالة `Input()` لقراءة المعلومات المختلفة، فالسطر التالي على سبيل المثال يحتوي على ثلاثة أجزاء مختلفة من البيانات، نص ورقم وتاريخ مفصولةً بعلامة الفاصلة "،" حيث تقوم الدالة `Input()` بالبحث عن أي فاصل مثل الفاصلة أو علامة الاستفهام أو الرمز `#`:

"Test",100,#2014-01-01#

يتم في الكود التالي قراءة كل عنصر من هذه العناصر داخل متغير من نفس نوعه:

```
Dim FileNumAs Integer
Dim TempStringAs String
Dim TempDate As DateTime
Dim TempInt As Integer
```

```
FileNum = FreeFile()
FileOpen(FileNum,"C:\myfile.txt", OpenMode.Input)
Input(FileNum, TempString)
Input(FileNum, TempDate)
Input(FileNum, TempInt)
```

**FileClose(FileNum)**

بعد الانتهاء من استخدام الملف، يجب أن تقوم دائماً بإغلاقه باستخدام الدالة **FileClose()** وبالتالي يمكنك تحرير رقم الملف تمهيداً لاستخدامه مع ملف آخر. أما إذا أردت إغلاق جميع الملفات المفتوحة في وقت واحد، فقم باستدعاء الدالة **FileClose()** دون أن تقوم بتحديد رقم الملف هكذا:

**FileClose()**

### الكتابة إلى الملف

يمكنك الكتابة إلى الملف من خلال الدالة **FileOpen()** وذلك باختيار النمط **OpenMode.Append** إذا أردت الإضافة إلى الملف الحالي إذا كان موجوداً بالفعل أو إنشاء ملف جديد إذا لم يكن الملف موجوداً، أو اختيار النمط **OpenMode.Output** إذا أردت إنشاء ملف جديد مع حذف بياناته إذا كان الملف موجوداً من قبل. أي يمكنك استخدام أي من الصيغتين التاليتين:

- **Append Mode** ويقوم بالإضافة إلى ملف موجود مسبقاً أو إنشاء ملف جديد  
**FileOpen(FileNum,"C:\myfile.txt", OpenMode.Append)**

- **Output Mode** ويقوم دائماً بإنشاء ملف جديد مع حذف أى بيانات موجودة مسبقاً.

**FileOpen(FileNum,"C:\myfile.txt", OpenMode.Output)**

وبعد فتح الملف باستخدام أي من الصيغتين، يمكنك استخدام الوظيفة **Print()** أو الوظيفة **PrintLine()** لكتابة البيانات إلى هذا الملف، حيث تختلف الوظيفة الثانية في إضافة سطر جديد بعد كتابة البيانات. كما يمكنك استخدام الدالتين **Write()** و **WriteLine()** التي تقوم بإضافة الفواصل تلقائياً. فمثلاً تقوم العبارة التالية:

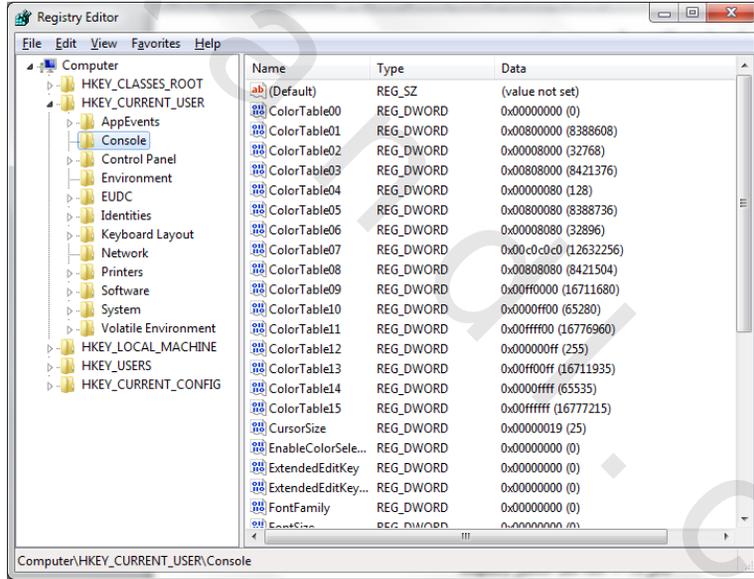
**WriteLine(FileNum, "This is a write example", #01/01/2014#,100)**

بكتابة السطر التالي داخل الملف:

**This is a write example", #01/01/2014#,100**

## استخدام ملف التسجيل Registry

يتكون ملف تسجيل Windows Registry أو Windows Registry من قاعدة بيانات هرمية تستخدم من قبل نظام التشغيل للاحتفاظ بالإعدادات المختلفة للنظام. فعلى سبيل المثال، إعدادات مستكشف Windows وآخر العناوين التي تم فتحها وكذلك مصادر بيانات ODBC التي تقوم بإنشائها، يتم تخزين كل ذلك داخل ملف التسجيل Registry. يوضح شكل ١٤-١ التالي نافذة ملف التسجيل التي يمكنك استخدامها في عرض وتحديث بيانات هذا الملف. للحصول على هذه النافذة، قم بتشغيل البرنامج regedit.exe من الخيار Run بقائمة Start.



شكل ١٤-١ نافذة ملف التسجيل Registry

### التعرف على تركيب ملف التسجيل

يظهر ملف التسجيل كما ترى من الشكل السابق في صورة شجرة من المجلدات تشبه إلى حد كبير المجلدات الفرعية الموجودة على القرص الصلب المثبت بجاسيك عند عرضها داخل نافذة مستكشف Windows، حيث يطلق على كل مجلد داخل شجرة التسجيل مصطلح

"مفتاح التسجيل" **Registry Key**. كما يوجد مجموعة من المفاتيح الأساسية التي تبدأ دائماً بكلمة **HKEY** مثل **HKEY\_CURRENT\_USER** الذى يحتوى على معلومات خاصة بالمستخدم الحالى. يوضح جدول ٤-١٤ المفاتيح الأساسية واستخدام كل منها. وقد تحتوى مفاتيح التسجيل على مفاتيح أخرى فرعية تسمى **Subkeys** بالإضافة إلى زوج من البيانات يحتوى على الاسم **Name** والقيمة **Value** التى قد تكون نص **String** أو قيمة عشرية **Decimal** أو قيمة ثنائية **Binary** أو حتى قيمة بالنظام السداسى عشر **Hexadecimal** طبقاً لاحتياجات التطبيق نفسه.

جدول ٤-١٤ المفاتيح الأساسية داخل ملف التسجيل

| المفتاح                    | الاستخدام   |
|----------------------------|---|
| <b>HKEY_CLASSES_ROOT</b>   | يحتوى على البيانات اللازمة لربط أنواع الملفات ببرامجها المصاحبة إلى جانب بيانات تهيئة المكونات <b>COM</b> |
| <b>HKEY_CURRENT_USER</b>   | تحتوى على بيانات التهيئة الخاصة بالمستخدم الحالى داخل <b>Windows</b>                                      |
| <b>HKEY_LOCAL_MACHINE</b>  | تحتوى على بيانات التهيئة الخاصة بالحاسب بغض النظر عن المستخدم الحالى داخل <b>Windows</b>                  |
| <b>HKEY_USERS</b>          | يحتوى على جميع بيانات المستخدمين المسجلين داخل <b>Windows</b>   |
| <b>HKEY_CURRENT_CONFIG</b> | يحتوى على بيانات التهيئة الخاصة بالمكونات المادية والى يقوم الكمبيوتر بإعدادها أثناء عملية التشغيل        |

## التحكم في قيم ملف التسجيل

يمكنك استخدام التصنيف **Registry** الموجود داخل المسمى **Microsoft.Win32** لتخزين واسترجاع بيانات ملف التسجيل. يمكنك ربط أحد المفاتيح بحالة من حالات التصنيف **Registry** ومن ثمّ يمكنك نداء الوظائف المناسبة لتعيين القيم أو قراءتها.

ينصح بتوخي الحذر عند التعامل مع ملفات التسجيل، فقد يؤثر تغيير إحدى القيم أو حذفها على برنامج بالكامل أو ربما على نظام التشغيل برمته.



تحتوي مفاتيح التسجيل على مسارات مفصولة بحرف الشرطة المائلة كما في مسارات الملفات تماماً. فإذا أردت تعريف أحد المفاتيح، يجب أن تقوم بتعيين مسار مفتاح المستوى الأعلى والمفتاح الفرعي، حيث يحتوي كل مفتاح مستوى أعلى على الحقل الساكن المصاحب داخل كائن التصنيف **My.Computer.Registry** كما في جدول ١٤-٥ التالي.

جدول ١٤-٥ الحقول الساكنة داخل التصنيف **My.Computer.Registry**

| مفتاح التسجيل المصاحب | الحقل         |
|-----------------------|---------------|
| HKEY_CLASSES_ROOT     | ClassesRoot   |
| HKEY_CURRENT_CONFIG   | CurrentConfig |
| HKEY_CURRENT_USER     | CurrentUser   |
| HKEY_LOCAL_MACHINE    | LocalMachine  |
| HKEY_USERS            | Users         |

يمكنك الوصول إلى أي مفتاح داخل ملف التسجيل من خلال الوظيفة (**OpenSubKey**) مع استخدام الحقل المناسب من الجدول السابق والمسار الكامل للمفتاح كما في الكود التالي:

**Imports Microsoft.Win32**

```
.....
Const WinLogonPath As String = "Software\Microsoft\Windows
NT\CurrentVersion\WinLogo"
Dim LogonKey as RegistryKey
LogonKey =
My.Computer.Registry.LocalMachine.OpenSubKey(WinLogonPath, True)
```

وقد قمنا باستخدام الحقل LocalMachine لتعريف مفتاح المستوى العلوى KEY\_LOCAL\_MACHINE الذى يحتوى على معلومات عن الحاسب. كما تستقبل الوظيفة OpenSubKey() بقية مسار المفتاح الفرعى المطلوب من خلال المعامل الأول للوظيفة، بينما يحتوى المعامل الثانى على قيمة منطقية تحدد إذا كان هذا المفتاح قابلاً للتعديل أم لا. فإذا كنت ترغب فقط فى قراءة البيانات من ملف التسجيل، قم بتخصيص القيمة False لهذه الخاصية، أما إذا كنت ترغب فى تحديث ملف التسجيل، فيمكنك فى هذه الحالة استخدام القيمة True.

يمكنك أيضاً التعامل مع القيم الموجودة داخل أحد ملفات التسجيل من خلال الوظيفتين GetValue() و SetValue() المصنعتان لكائناً التصنيف My.Computer.Registry كما فى الكود التالى الذى يقوم بتعيين قيم التسجيل الخاصة بالمفتاح الفرعى WinLogon:

```
My.Computer.Registry.SetValue(WinLogonPath,"AutoAdminLogon","1")
My.Computer.Registry.SetValue(WinLogonPath,"DefaultUserName","Administrator")
My.Computer.Registry.SetValue(WinLogonPath,"DefaultPassword","topsecret")
```

تقوم الوظيفة SetValue() بتعيين القيمة إذا لم تكن موجودة بالفعل. يمكنك أيضاً قراءة إحدى القيم من ملف التسجيل بسهولة تامة من خلال الوظيفة GetValue() كما فى الكود التالى:

```
Dim strValue As String
strValue = LogonKey.GetValue("AutoAdminLogon", "?").ToString
```

كما يمكنك استبدال الكائن LogonKey فى السطر الثانى بالعبارة My.Computer.Registry هكذا:

```
strValue = My.Computer.Registry.GetValue("AutoAdminLogon", "?").ToString
```

وتقوم الوظيفة (**GetValue()**) بإرجاع كائن يمكن تحويله إلى قيمة نصية أو رقمية كما تحتوي هذه الوظيفة على معامل ثنائي اختياري يتيح لك تعيين القيمة الافتراضية المرتجعة من هذه الوظيفة. ففي الكود السابق، إذا لم يكن هناك قيمة للمفتاح **AutoAdminLogon**، يتم إرجاع علامة الاستفهام "?". أما إذا لم تقم بتعيين قيمة افتراضية وصادف ذلك عدم وجود قيمة داخل المفتاح، فسيتم إرجاع القيمة **Null**.

### إنشاء مفاتيح التسجيل

يمكنك إنشاء مفاتيح تسجيل جديدة داخل ملف التسجيل باستخدام الوظيفة (**CreateSubKey()**) والتي تحتوي على معامل وحيد يمثل المفتاح الذي ترغب في إضافته. فعلى سبيل المثال، لإضافة المفتاح **Software\Compuscience\MyProject** إلى المفتاح الرئيسي **HKEY\_CURRENT\_USER**، يمكنك استخدام العبارة التالية:

```
My.Computer.Registry.CurrentUser.CreateSubKey("Software\Compuscience\MyProject")
```

### حذف مفاتيح التسجيل

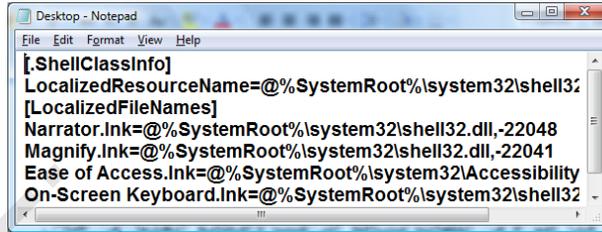
يمكنك حذف أي من مفاتيح التسجيل الموجودة داخل ملف التسجيل باستخدام الوظيفة (**DeleteSubKey()**) والتي تحتوي على معامل وحيد أيضاً يمثل المفتاح الذي ترغب في حذفه. فعلى سبيل المثال، لحذف المفتاح **Software\Compuscience\MyProject** الذي أضفناه منذ قليل إلى المفتاح الرئيسي **HKEY\_CURRENT\_USER**، يمكنك استخدام العبارة التالية:

```
My.Computer.Registry.CurrentUser.DeleteSubKey("Software\Compuscience\MyProject")
```

### مفهوم ملفات INI

تستخدم ملفات INI (تنطق any) في تخزين بيانات البرنامج وإعدادات المستخدم، وهي عبارة عن ملفات نصية تحتوي على تركيب معين يتيح لك تخزين واسترجاع أجزاء معينة من البيانات. ويتميز هذا النوع من الملفات ببساطة التركيب وإمكانية التعديل من خلال أي من

محررات النصوص كبرنامج Notepad على سبيل المثال. يحتوي شكل ١٤-٢ التالي على نموذج بسيط لملف INI داخل نافذة برنامج Notepad.



شكل ١٤-٢ نموذج بسيط لملف INI

يحتوي ملف INI على ثلاثة عناصر أساسية هي الأقسام Sections والتي تسميها مايكروسوفت أيضاً بالتطبيق Application حيث يرجع ذلك إلى استخدام ملف Win.ini قديماً لتخزين بيانات التطبيقات، والمفاتيح Keys وأخيراً القيم Values. يوضح جدول ١٤-٦ التالي هذه العناصر.

جدول ١٤-٦ عناصر الملف INI

| العنصر        | الوصف   |
|---------------|---|
| القسم Section | يتم تمثيله باسم موجود بين القوسين [ ]، ويقوم بتجميع مجموعة من المفاتيح والقيم مع بعضها البعض                  |
| المفتاح Key   | عبارة عن نص فريد متبوعاً بعلامة = ثم قيمة معينة، حيث يجب أن يكون هذا النص مميزاً فقط داخل القسم المعرف بداخله |
| القيمة Value  | يمثل القيمة الحقيقية لمفتاح معين داخل الملف، حيث يتم استخدام القسم والمفتاح معاً لقراءة القيمة أو كتابتها     |

لا يتم وضع ترتيب الأقسام داخل ملف INI في الحسبان، لأن أسماء الأقسام والمفاتيح تشير في جميع الأحوال إلى قيمة واحدة فقط. يمكنك أيضاً كتابة التعليقات داخل ملف INI باستخدام رمز الفاصلة المنقوطة وذلك كما في القسم التالي:

[Settings]

DBLocation = C:\Compuscience.mdb

;DBLocation = D:\Books\Compuscience.mdb

حيث يمكنك التنقل بين مصدر البيانات بسهولة من خلال تحويل السطر الذي ترغب فيه إلى تعليق.

### استخدام ملفات INI داخل VisualBasic

أحد أسباب سهولة استخدام ملفات INI هو عدم القلق بشأن إنشاء الملف أو فتحه أو إيجاد سطر معين بداخله. فكل ما تحتاجه هو استدعاء دالة واحدة فقط لقراءة قيمة أو كتابتها. وقبل أن تتمكن من استخدام ملفات INI، يجب أن تقوم بتعريف دالتين من دوال Windows API. وقد قمنا بأداء ذلك من أجلك، قم بإضافة الكود التالي إلى الجزء العلوي من تصنيف النموذج:

```
Declare Function GetPrivateProfileString Lib "kernel32" Alias
"GetPrivateProfileStringA" (ByValpApplicationName As String,
ByValpKeyName As String, ByValpDefault As String,
ByValpReturnedString As String, ByValnSize As Integer,
ByValpFileName As String) As Integer
Declare Function WritePrivateProfileString Lib "kernel32" Alias
"WritePrivateProfileStringA" (ByValpApplicationName As String,
ByValpKeyName As String, ByValpString As String,
ByValpFileName As String) As Integer
```

```
Public Shared Function sGetINI(ByValsINIFile As String,
ByValsSection As String, ByValsKey As String, ByValsDefault
As String) As String
```

```
Dim sTemp As String = Space(255)
Dim nLength As Integer
```

```
nLength = GetPrivateProfileString(sSection, sKey, sDefault,
sTemp, 255, sINIFile)
Return sTemp.Substring(0, nLength)
```

```
End Function
```

```
Public Shared Sub writeINI(ByValsINIFile As String,
ByValsSection As String, ByValsKey As String, ByValsValue As
String)
```

```
    'Remove CR/LF characters
    sValue = sValue.Replace(vbCr, vbNullChar)
    sValue = sValue.Replace(vbLf, vbNullChar)
```

```
    'Write information to INI file
    WritePrivateProfileString(sSection, sKey, sValue, sINIFile)
```

```
End Sub
```

وبذلك يمكنك استخدام الدالة `sGetINI()` والإجراء الفرعي `writeINI()` للكتابة في أي ملف INI أو القراءة منه.

يوضح الكود التالي كيفية استرجاع الإعدادات من ملف `Settings.INI` واستخدام هذه الإعدادات في بداية تشغيل التطبيق:

```
1. Private Sub InitProgram()
2. Dim sINIFile As String
3. Dim sUserName As String
4. Dim nCount As Integer
5. Dim i As Integer

6. 'Store the location of the INI file
7. sINIFile = Application.StartupPath & "\..\SETTINGS.INI"
8. 'Read the user name from the INI file
9. sUserName = sGetINI(sINIFile, "Settings", "UserName",
    "?")
10. If sUsername = "?" Then
11. 'No user name was present - ask for it and save for next
    time
12. sUserName = InputBox("Enter your name please:")
13. writeini(sINIFile, "Settings", "UserName", sUserName)
14. Else
15. Me.Text = "Welcome back " & sUserName
16. End If
17. 'Fill up combo box list from INI file
18. cmbRegion.Items.Clear()
19. nCount = Convert.ToInt32(sGetINI(sINIFile, "Regions",
```

```
"Count", "0"))
20. If nCount > 0 Then
21. For i = 0 To nCount
22. cmbRegion.Items.Add(sGetINI(sINIFile, "Regions",
"Region" & i, "?"))
23. Next i
24. 'select the user's last chosen item
25. cmbRegion.SelectedIndex =
26. Convert.ToInt32(sGetINI(sINIFile, "Regions",
"LastRegion", "0"))
27. End If
28. End Sub
```

وعن هذا الكود، نوضح ما يلي:

- يتم في السطر رقم ٧ تخزين عنوان الملف على وحدة التخزين داخل المتغير النصي **.sINIFile**.
  - يتم في السطر رقم ٩ استدعاء الدالة **sGetINI()** لاسترجاع اسم المستخدم، حيث تحتوي هذه الدالة على أربعة معاملات، الأول هو اسم ملف **INI** والثاني اسم القسم والثالث المفتاح بينما يعبر الرابع عن القيمة الافتراضية التي تظهر في حالة عدم وجود القيمة الأصلية داخل الملف.
  - يتم في السطور من ١٠ إلى ١٦ اختبار قيمة اسم المستخدم، فإذا تم إرجاع القيمة الافتراضية، يتم إظهار مربع إدخال للمستخدم لإدخال اسمه، ومن ثمّ يتم استخدام الإجراء **writeini()** لكتابة هذا الاسم إلى الملف، حيث يحتوي الإجراء على أربعة معاملات أيضاً، الأول هو اسم ملف **INI** والثاني اسم القسم والثالث المفتاح بينما يعبر الرابع عن القيمة التي ترغب في كتابتها وهي اسم المستخدم في هذه الحالة. أما إذا كان اسم المستخدم موجوداً بالفعل، فيتم تغيير عنوان النموذج ليحتوي على رسالة ترحيب بالمستخدم.
  - يتم في السطور من ١٨ إلى ٢٧ استرجاع بيانات الملف **Settings.ini** وتخزينها داخل مربع سرد باسم **cmbRegion** وذلك من خلال الدالة **sGetINI()**.
- يوضح شكل ١٤-٣ التالي محتويات الملف **Settings.ini**.

```
SETTINGS - Notepad
File Edit Format View Help
[Regions]
Count=4
Region0=All Regions
Region1=South
Region2=North
Region3=East
Region4=West
```

شكل ١٤-٣ محتويات الملف Settings.ini



obeykandi.com

## الباب الخامس

### مفاهيم متقدمة

١٥ . معالجة الاستثناءات

١٦ . تخزين التطبيقات وإعدادها للتوزيع