الفصل السادس عشر
المزيد عن الملفات

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

بانتهاء هذا الفصل ستتعرف 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 والتى 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 هى:

- الملفات النصية Text Files التى تحتوى دائماً على نصوص قابلة للقراءة من قِبَل المستخدم العادى والتى يمكنك تعديلها من خلال أىٍ من برامج تحرير النصوص كبرنامج Notepad مثلاً.
- الملفات الثنائية Binary Files والتى تحتوى على البيانات المستخدمة من قِبَل 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. ففى 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 قِبَل 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 فى الذاكرة ووحـــدات التخـــزين على حدٍ سواء، كما يمكنك الوصول إليها واستخدامها من خلال الشبكة المتصـــل بهـــا حاسبك.

العمل مع الملفات النصية

يمكنك التعامل مع الملفات النصية داخل نطاق **.NET** سواءً بالقراءة أو الكتابة من خلال التصنيف **StreamReader** والتصنيف **StreamWriter** على الترتيـــب، حيـــث تم تصميم التصنيفين للتعامل مع هذه الملفات فى صورة مسلسلة **Sequential**.

يمكنك معالجة البيانات الموجودة داخل الملف التسلسلى سطر بسطر أو بقراءة عدد معـــين من الحروف. يوضح الكود التالى كيفية قراءة كل سطر داخل الملف النصى وطباعته على نافذة الإخراج:

```
Dim TestFile As StreamReader
Dim LineofText As 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 اســـتدعاء الوظيفـــة **()ReadLine** باستمرار حتى يتم قراءة محتويات الملف بالكامل.

يمكنك أيضاً استخدام الوظيفة **()WriteLine** لكتابة سطر من النصوص إلى أحد الملفات. يوضح الكود التالى كيفية قراءة كل سطر من الملف **myfile.txt** ثم كتابته بحروف كبيرة إلى ملف نصى آخر بعنوان **output.txt**.

```
Dim TestFile As StreamReader
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 النصى وكتابته، فقد قمنا أولاً بفتح الملف من خلال الوظيفة **File.AppendText()** بدلاً من الوظيفة **File.OpenText()** المستخدمة عند القراءة من الملف، حيث تعنى 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 فى هذه الحالة استخدام الوظيفة **CreateText()** كما فى الكود التالى:

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

أما الاختلاف الثانى فيكمن فى استخدام الوظيفة **Peek()** داخل الدوارة **While**، حيث تقوم هذه الوظيفة بإرجاع كود الحرف التالى داخل الملف دون أن تقوم بقراءته ومن ثمّ فهى تقوم بإرجاع القيمة **1-** بمجرد الوصول إلى نهاية الملف الذى تقوم بالقراءة منه.

العمل مع الملفات الثنائية

يقوم نظام التشغيل باعتبار أى ملف من الملفات مجموعة من الحروف التى يمثل كل منها بايت داخل الملف. وقد صُمم التصنيف **StreamReader** والتصنيف **StreamWriter** للتعامل مع هذه الحروف كبيانات نصية. ولكن ربما لا يمثل البايت الموجود بالملف دائماً حرف من الحروف وإنما يمثل رقم معين من الأرقام العددية. يحتوى نطاق **.NET** على التصنيف **BinaryReader** والتصنيف **BinaryWriter** التى تتيح لك الوصول المباشر إلى البيانات داخل الملفات بدلاً من الطريقة المتتابعة (المسلسلة) المستخدمة فى حالة الملفات

النصية. يوضح جدول ١٦-١ التالى أهم الوظائف الموجودة بالتصنيفين السابقين.

جدول ١٦-١ أهم وظائف التصنيفين BinaryReader وBinaryWriter

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 نصى باستخدام نوع البيانات Byte وقراءة ١٠٠ بايت فى المرة الواحدة، حيث يمكنك معالجة الملف من خلال بايت واحد فى كل مرة أو باختيار حجم معين، كما يتم تحويل كل بايت إلى الحرف ASCII المناظر كما فى الكود التالى:

```
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 الذى 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 فى الجزء الثالث من هذه السلسلة. فعلى الرغم من استخدام **XML** لتخزين البيانات المركبة، إلا أنه يمكنك استخدام هذا التنسيق فى كتابة الملفات النصية المستخدمة فى الأغراض المختلفة. ولعل أحد الأمثلة على استخدام ملفات **XML** يتمثل فى 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
```

فى هذا الكود، يقوم التصنيف Customer بوصف كل عميل من العملاء من خلال اسمه ورقمه وقائمة بالعناصر التى يتكون منها طلب الصرف الخاص بهذا العميل. كما يحتوى كل عنصر داخل طلب الصرف على اسم هذا العنصر وسعره وكميته. على فرض أن لـــدينا مصفوفة من العملاء ونرغب فى إنشاء ملف XML لتخزين كل هذه المعلومات. ســـنقوم بأداء ذلك من خلال التصنيف XMLDocument الذى يتعامـــل مـــع بيانـــات XML بالذاكرة والتصنيف XMLWriter الذى يقوم بكتابة البيانات إلى ملف XML النـــاتج. وذلك من خلال الكود التالى:

```
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 لصنف 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 = Customers(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 والتى يحتوى كل منها بدوره على عنصر أبوى Parent وعنصر وليد Child.

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("C:\temp\output.xml")
For Each tempnode In xmlDoc.DocumentElement.ChildNodes
        Debug.WriteLine("Customer " & tempnode.InnerText & "
        has " & TempNode.LastChild.ChildNodes.Count & "
         orders.")
Next
```

وفى 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 التى لم تزل مدعمة داخل .NET. على الرغم من وجود بعض التعديلات الطفيفة فى 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 يلى:

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

البحث عن الملفات

تعتبر الوظائف Dir من الوظائف المفيدة والتى 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 الذى يمثل كل الملفات ذات الامتداد BAT في الدليل الرئيسى للقرص C. والصيغة العامة للوظيفة Dir كما يلى:

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

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

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

```
Public Function bFileExists(sFile As String) As Boolean
        If Dir(sFile) <> " " Then
                bFileExists = True
        Else
                bFileExists =False
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، فلابد 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. والتى 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 اختيارى 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 ١٦-٢ التالى الخيارات المتاحة للمعامل الثانى داخل الدالة Dir().

جدول ١٦-٢ خيارات الصفة FileAttribute

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

دوال التعامل مع الملفات

تتشابه كثير من دوال التعامل مع الملفات في Visual Basic مع نظائرها في MS-DOS، كما يبين جدول ١٦-٣ التالي.

جدول ١٦-٣ دوال التعامل مع الملفات

العملية	الصيغة
نسخ ملف	**FileCopy *source, dest***
محو ملف أو أكثر	**Kill *path***
إنشاء مجلد جديد	**MkDir *pathname***
محو مجلد خالى	**RmDir *pathname***
تغيير الدليل الحالى	**ChDir *pathname***
تغيير القرص الحالى	**ChDrive *drive***

وفيما يلى نوضح استخدام كل دالة من هذه الدوال.

نسخ ملف

ينقص الدالة **FileCopy()** عن نظيرتها في **MS-DOS** عدم القدرة على استخدام الرمزين الشاملين وهما *، ؟ (**Wildcards**) لتحديد أكثر من ملف. يقوم المثال التالى بنسخ ملف النص **VbSample.TXT** إلى الملف **Sample.TXT** الموجود تحت المجلد **VB** فى 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 الرمزين الشاملين **Wildcards** مع الدالة **Kill()** كما في الأمر **Del** في **MS-DOS**. انظر المثال التالى:

```
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 يلى:

```
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() التى تقوم بدورها بربط الملف المطلوب برقم صحيح يمكن استخدامه فى التعامل مع هذا الملف من خلال السطور المتبقية من الكود كما يلى:

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

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

وقد قمنا فى 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() التى تقوم بإرجاع الرقم المتاح التالى كما فى الكود التالى:

```
Dim FileNum As Integer
FileNum = FreeFile()
FileOpen(FileNum, "C:\myfile.txt", OpenMode.Input)
```

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

```
Dim FileNum As 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 مستخدمى الإصدارات القديمة من Visual Basic، ستلاحظ تغيير واضح فى 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 2008.

وبعد أن تقوم بفتح الملف، يمكنك اختيار الوظيفة المناسبة لقراءة البيانات من هذا الملف. ففى 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، وفى هذه الحالة يمكنك استخدام الدالة Input() بدلاً من الدالة LineInput().

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

```
"Test",100,#2008-01-01#
```

يتم فى الكود التالى قراءة كل عنصر من هذه العناصر داخل متغير من نفس نوعه:

```
Dim TempString As String
Dim TempDate As DateTime
Dim TempInt As Integer

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

FileClose(FileNum)
```

بعد الانتهاء من استخدام الملف، يجب أن تقوم دائماً بإغلاقه باستخدام الدالة FileClose() وبالتالى يمكنك تحرير رقم الملف تمهيداً لاستخدامه مع ملف آخر. أما إذا

أردت إغلاق جميع الملفات المفتوحة فى وقتٍ 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 الحالى 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. أى يمكنك استخدام أيٍ من الصيغتين التاليتين:

- ويقوم بالإضافة إلى ملف موجود مسبقاً أو إنشاء ملف جديد **'Append Mode –**

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

- ويقوم دائماً بإنشاء ملف جديد مع حذف أى بيانات موجودة مسبقا **'Output Mode –**

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

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

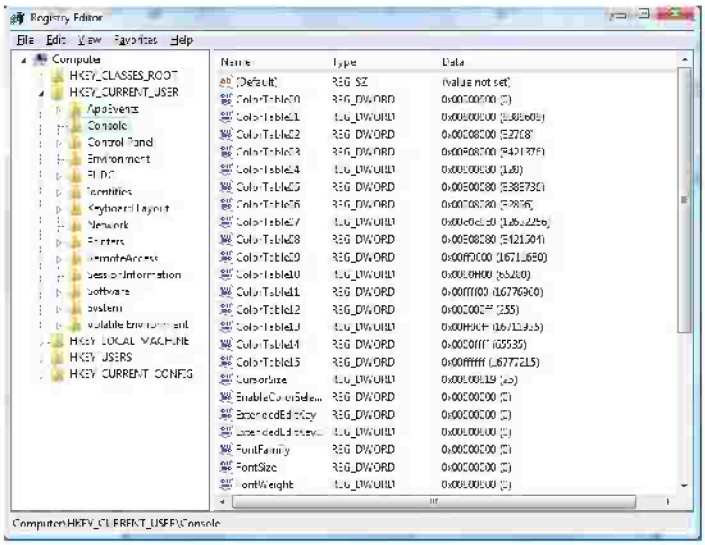
```
WriteLine(FileNum, "This is a write example", #01/01/2008#,100)
```

بكتابة السطر التالى داخل الملف:

```
This is a write example", #01/01/2008#,100
```

استخدام ملف التسجيل Registry

يتكون ملف تسجيل Windows أو Windows Registry من قاعدة بيانات هرمية تستخدم من قِبَل نظام التشغيل للاحتفاظ بالإعدادات المختلفة للنظام. فعلى سبيل المثال، إعدادات مستكشف Windows وآخر العناوين التى تم فتحها وكذلك مصادر بيانات ODBC التى تقوم بإنشائها، يتم تخزين كل ذلك داخل ملف التسجيل Registry. يوضح شكل ١٦-١ التالى نافذة ملف التسجيل التى يمكنك استخدامها فى عرض وتحديث بيانات هذا الملف. للحصول على هذه النافذة، قم بتشغيل البرنامج regedit.exe.



شكل ١٦-١ نافذة ملف التسجيل Registry

التعرف على تركيب ملف التسجيل

يظهر ملف التسجيل كما ترى من الشكل السابق فى صورة شجرة من المجلدات تشبه إلى حد كبير المجلدات الفرعية الموجودة على القرص الصلب المثبت بحاسبك عند عرضها داخل نافذة مستكشف Windows، حيث يطلق على كل مجلد داخل شجرة التسجيل مصطلح "مفتاح التسجيل" Register Key. كما يوجد مجموعة من المفاتيح الأساسية التى تبدأ دائماً بكلمة HKEY مثل HKEY_CURRENT_USER الذى يحتوى على

معلومات خاصة بالمستخدم الحالى. وقد تحتوى مفاتيح التسجيل على مفاتيح أخرى فرعية تسمى Sub keys 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.

التحكم فى 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 ثمَّ يمكنك نداء الوظائف المناسبة لتعيين القيم أو قراءتها.

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

تحتوى مفاتيح التسجيل على مسارات مفصولة بحرف الشرطة المائلة كما فى مسارات الملفات تماماً. فإذا أردت تعريف أحد المفاتيح، يجب أن تقوم بتعيين مسار مفتاح المستوى الأعلى والمفتاح الفرعى، حيث يحتوى كل مفتاح مستوى أعلى على الحقل الساكن المصاحب داخل التصنيف Registry كما فى جدول ١٦-٤ التالى.

جدول ١٦-٤ الحقول الساكنة داخل التصنيف Registry

الحقل	مفتاح التسجيل المصاحب
ClassesRoot	HKEY_CLASSES_ROOT
CurrentConfig	HKEY_CURRENT_CONFIG
CurrentUser	HKEY_CURRENT_USER
DynData	HKEY_DYN_DATA
LocalMachine	HKEY_LOCAL_MACHINE
PerformanceData	HKEY_PERFORMANCE_DATA
Users	HKEY_USERS

يمكنك الوصول إلى أى مفتاح داخل ملف التسجيل من خلال الوظيفة OpenSubKey() مع استخدام الحقل المناسب من الجدول السابق والمسار الكامل للمفتاح كما فى الكود التالى:

```
Const WinLogonPath As String = "Software\Microsoft\Windows
NT\CurrentVersion\WinLogo"
Dim LogonKey As RegisteryKey
LogonKey = Registry.LocalMachine.OpenSubkey(WinLogon,
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** المصاحبتان لكائن التصنيف **RegisterKey** كما فى الكود التالى الذى يقوم بتعيين قيم التسجيل الخاصة بالمفتاح الفرعى **WinLogon**:

```
LogonKey.SetValue("AutoAdminLogon","1")
LogonKey.SetValue("DefaultUserName","Administrator")
LogonKey.SetValue("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:

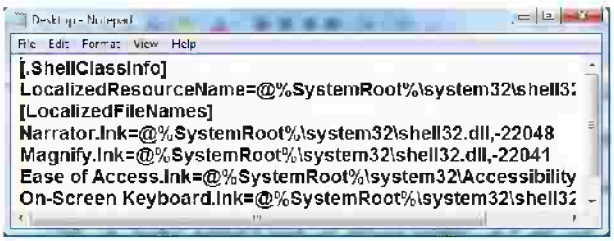
```
Dim strValue As String
strValue = LogonKey.GetValue("AutoAdminLogon",
"?").ToString
```

تقوم الوظيفة **()GetValue** بإرجاع كائن يمكن تحويله إلى قيمة نصية أو رقمية كما تحتوى هذه الوظيفة على معامل ثانى اختيارى يتيح لك تعيين القيمة الافتراضية المرتجعة من هذه الوظيفة. ففى 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**.

مفهوم ملفات INI

تستخدم ملفات **INI** (تنطق **any**) فى تخزين بيانات البرنامج وإعدادات المستخدم، وهى عبارة عن ملفات نصية تحتوى على تركيب معين يتيح لك تخزين واسترجاع أجزاء معينة من البيانات. ويتميز هذا النوع من الملفات ببساطة التركيب وإمكانية التعديل من خلال أى من محررات النصوص كبرنامج **Notepad** على سبيل المثال. يحتوى شكل ١٦-٢ التالى على نموذج بسيط لملف **INI** داخل نافذة برنامج **Notepad**.



شكل ١٦-٢ نموذج بسيط لملف INI

يحتوى ملف **INI** على ثلاثة عناصر أساسية هى الأقسام **Sections** والتى تسميها مايكروسوفت أيضاً بالتطبيق **Application** حيث يرجع ذلك إلى استخدام ملف **Win.ini** قديماً لتخزين بيانات التطبيقات، والمفاتيح **Keys** وأخيراً القيم **Values**. يوضح جدول ١٦-٥ التالى هذه العناصر.

جدول ١٦-٥ 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

العنصر	الوصف
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 فى الحسبان، لأن أسماء الأقسام والمفاتيح تشير فى جميع الأحوال إلى قيمة واحدة فقط.

يمكنك أيضاً كتابة التعليقات داخل ملف INI باستخدام رمز الفاصلة المنقوطة وذلك كما فى القسم التالى:

```
[Settings]
DBLocation = C:\Compuscience.mdb
;DBLocation = D:\Books\Compuscience.mdb
```

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

استخدام ملفات INI داخل Visual Basic

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 التالى إلى الجزء العلوى من تصنيف النموذج:

```
Declare Function GetPrivateProfileString Lib "kernel32" Alias
"GetPrivateProfileStringA" (ByVal lpApplicationName As String,
ByVal lpKeyName As String, ByVal lpDefault As String, ByVal
lpReturnedString As String, ByVal nSize As Integer, ByVal
lpFileName As String) As Integer
Declare Function WritePrivateProfileString Lib "kernel32" Alias
"WritePrivateProfileStringA" (ByVal lpApplicationName As
String, ByVal lpKeyName As String, ByVal lpString As String,
ByVal lpFileName As String) As Integer
```

```
Public Shared Function sGetINI(ByVal sINIFile As String, ByVal
sSection As String, ByVal sKey As String, ByVal sDefault 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(ByVal sINIFile As String, ByVal
sSection As String, ByVal sKey As String, ByVal sValue 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 الفرعى **writeINI()** للكتابة فى أى ملف **INI** أو القراءة منه.

يوضح الكود التالى كيفية استرجاع الإعدادات من ملف **Settings.INI** واستخدام هذه الإعدادات فى 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
```

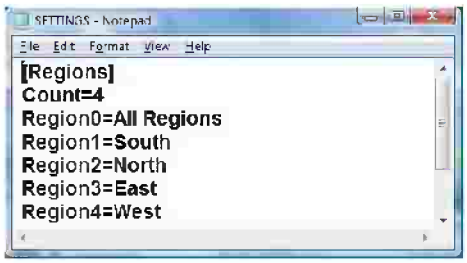
```
7.    sINIFile = Application.StartupPath & "\..\SETTINGS.INI"
8.    'Read the user name from the INI file
9.    sUserName = sGetINI(sINIFile, "Settings", "UserName",
      "?")
10.   If sUsername = "?" Then
11.      'No user name was present - ask for it and save for next
          time
12.      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 يلى:

- يتم فى السطر رقم ٧ تخزين عنوان الملف على وحدة التخزين داخل المتغير النصى **sINIFile**.
- يتم فى السطر رقم ٩ استدعاء الدالة **sGetINI()** لاسترجاع اسم المستخدم، حيث تحتوى هذه الدالة على أربعة معاملات، الأول هو اسم ملف **INI** والثانى اسم القسم والثالث المفتاح بينما يعبر الرابع عن القيمة الافتراضية التى تظهر فى حالة عدم وجود القيمة الأصلية داخل الملف.

- يتم فى السطور من ١٠ إلى ١٦ اختبار قيمة اسم المستخدم، فإذا تم إرجاع القيمة الافتراضية، يتم إظهار مربع إدخال للمستخدم لإدخال اسمه، ومن ثَمَّ يتم استخدام الإجراء **writeini()** لكتابة هذا الاسم إلى الملف، حيث يحتوى الإجراء على أربعة معاملات أيضاً، الأول هو اسم ملف **INI** والثانى اسم القسم والثالث المفتاح بينما يعبر الرابع عن القيمة التى ترغب فى كتابتها وهى اسم المستخدم فى هذه الحالة. أما إذا كان اسم المستخدم موجوداً بالفعل، فيتم تغيير عنوان النموذج ليحتوى علـــى رسالة ترحيب بالمستخدم.
- يتم فى السطور من ١٨ إلى ٢٧ استرجاع بيانات الملف **Settings.ini** وتخزينها داخل مربع سرد باسم **cmbRegion** وذلك من خلال الدالة **sGetINI()**.

يوضح شكل ١٦-٣ التالى محتويات الملف **Settings.ini**.



شكل ١٦-٣ محتويات الملف **Settings.ini**