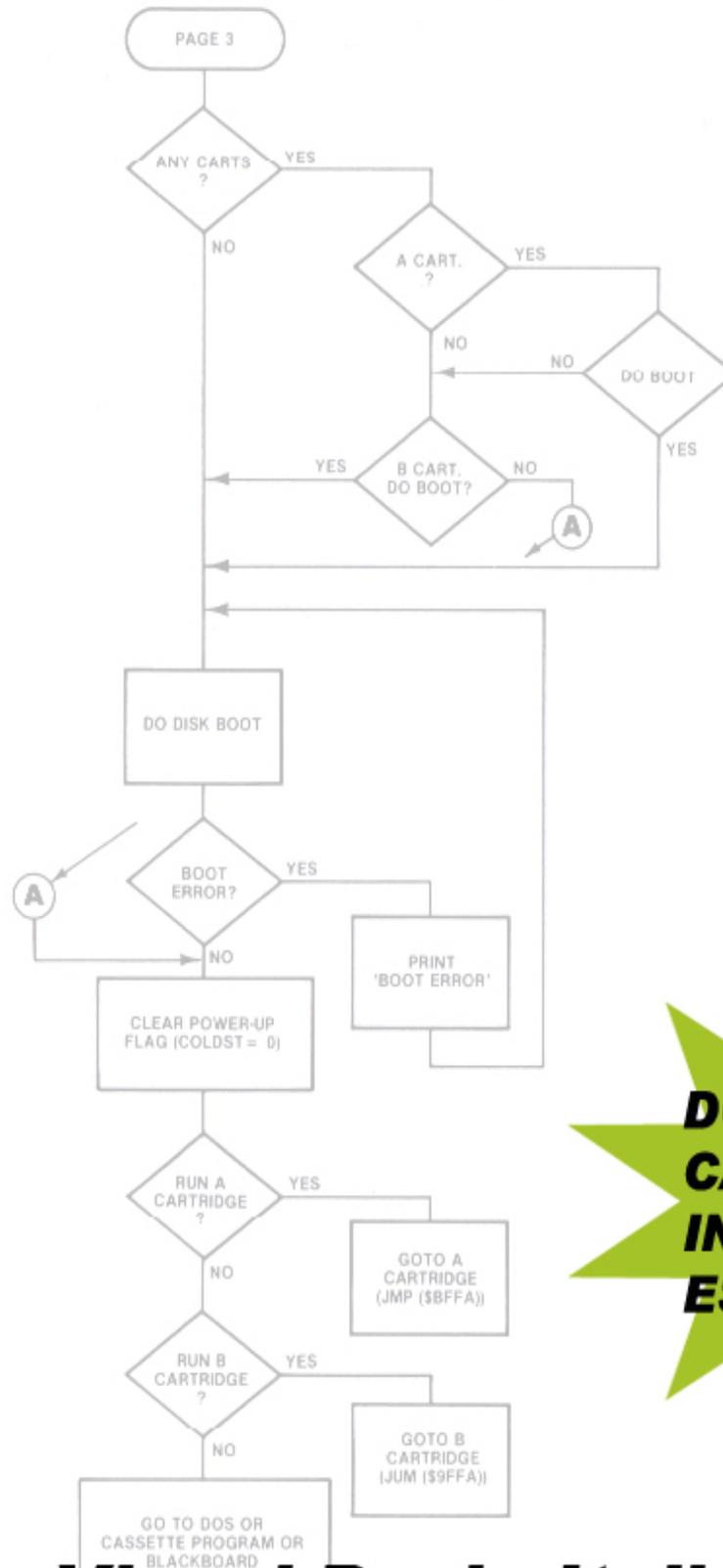


# GUIDA COMPLETA AI FILE .INI



**DUE  
CAPITOLI  
INEDITI E DUE  
ESEMPI**

**Visual Basic Italia**

## Indice

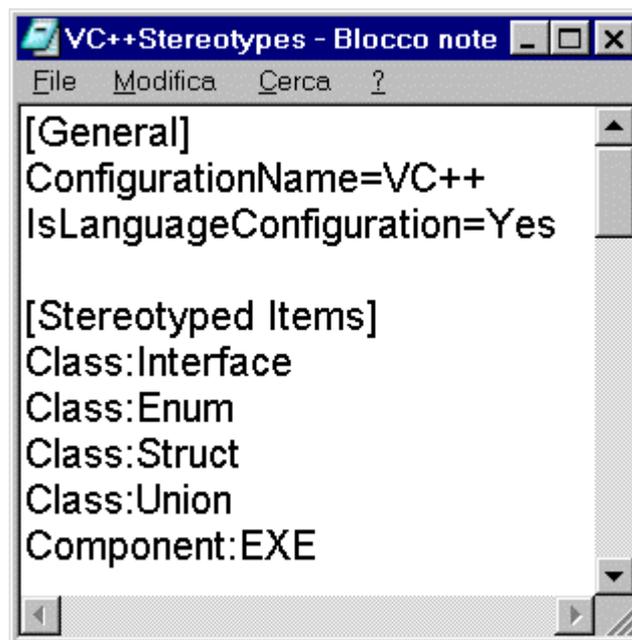
Struttura di un file .INI.....	3
Le funzioni per la lettura e la scrittura.....	4
Recuperare un valore.....	4
Conoscere la struttura di un file INI .....	8
Convertire un file INI in XML.....	8
I file INI e Visual Basic .NET.....	13

## Struttura di un file .INI

I file .INI, estensione che deriva dal termine **INI**tialization, sono sostanzialmente semplici file di testo che permettono di memorizzare informazioni utili sulle applicazioni accessibili all'utente. La loro struttura è tale da poter salvare e recuperare in modo semplice ed intuitivo le informazioni in essi contenuti.

L'utilità di tali files consiste proprio nel vantaggio di poter immagazzinare in essi tutte le informazioni sull'applicazione, evitando così di dover ricompilare il programma per modificarle. In poche parole dunque un file INI rappresenta un tramite tra il programmatore e l'applicazione sviluppata.

Una seconda caratteristica utile dei files INI è quella di essere files di testo, cosa che rende particolarmente agevole l'eventuale elaborazione del loro contenuto, visto che a tale scopo può essere utilizzato qualsiasi editor di testo (per esempio il classico ma sempre utile Microsoft Notepad). Qui di seguito mostriamo un semplice esempio di file .ini aperto per l'appunto con l'aiuto di Microsoft Notepad:



Un file di questo tipo si compone in sostanza di tre elementi: le sezioni, le chiavi ed i valori. Facendo riferimento all'immagine appena vista, analizziamo ciascuno di questi tre fattori:

**Sezione** è il nome incluso nella parentesi (nella figura soprastante [GENERALE]) che raggruppa una serie di valori e di chiavi. Rappresenta, semplificando molto il concetto, una sorta di titolo del paragrafo.

**Chiave** è una stringa univoca, almeno all'interno di ogni singola sezione, ossia continuando il nostro paragone semplificato, all'interno di ogni paragrafo. La chiave è sempre seguita dal segno "=" e da un valore.

**Valore** è il valore corrente di una particolare chiave in un file .INI. Sia le sezioni che le chiavi servono praticamente per poter definire un valore.

Come sono ordinati sezioni e chiavi all'interno del file non ha importanza perchè utilizzando nomi univoci non dovrebbero esserci ripetizioni nel documento ini. Inoltre il segno di punto e virgola ";" viene generalmente utilizzato per indicare un commento: come accade nel codice dei progetti sviluppati in Visual Basic in presenza del segno "'", una chiave o un valore che segue il punto e virgola viene del tutto ignorato.

## Le funzioni per la lettura e la scrittura

Prima di poter utilizzare un file INI comunque sarà necessario dichiarare due funzioni API e scrivere alcune funzioni cosiddette "di contorno" generate appunto intorno alle due funzioni:

La prima è l'API **GetPrivateProfileString** contenuta in kernel32.dll di cui abbiamo qui sotto la dichiarazione:

```
Declare Function GetPrivateProfileString Lib "kernel32" Alias _
"GetPrivateProfileStringA" (ByVal lpApplicationName _
As String, ByVal lpKeyName As Any, ByVal lpDefault _
As String, ByVal lpReturnedString As String, ByVal _
nSize As Long, ByVal lpFileName As String) As Long
```

La seconda è invece l'API **WritePrivateProfileString**:

```
Declare Function WritePrivateProfileString Lib "kernel32" Alias _
"WritePrivateProfileStringA" (ByVal lpApplicationName _
As String, ByVal lpKeyName As Any, ByVal lpString As Any, _
ByVal lpFileName As String) As Long
```

Per quello che riguarda le funzioni di contorno, vediamo una funzione **sGetINI** e una subroutine o sottoprocedura **writeINI** che ci servono rispettivamente per recuperare e per scrivere i dati di una nostra ipotetica applicazione da e su un file ini.

Ecco innanzitutto la funzione sGetINI, nella quale si può notare che i parametri richiesti sono il nome del file INI, la sezione, la chiave, il valore, il valore di default (che permette di operare anche se il file INI non esiste):

```
Public Function sGetINI(sINIFile As String, sSection As String, sKey _
As String, sDefault As String) As String
Dim sTemp As String * 256
Dim nLenght As Integer
sTemp = Space$(256)
nLenght = GetPrivateProfileString(sSection, sKey, sDefault, sTemp, _
255, sINIFile)
sGetINI = Left$(sTemp, nLenght)
End Function
```

Ed ecco adesso la subroutine writeINI:

```
Public Sub writeINI(sINIFile As String, sSection As String, sKey _
As String, sValue As String)
Dim n As Integer
Dim sTemp As String
sTemp = sValue
For n = 1 To Len(sValue)
If Mid$(sValue, n, 1) = vbCr Or Mid$(sValue, n, 1) = vbLf _
Then Mid$(sValue, n) = " "
Next n
n = WritePrivateProfileString(sSection, sKey, sTemp, sINIFile)
End Sub
```

## Recuperare un valore

La particolarità di un file INI risiede nella sua struttura, divisa come si è visto in Sezioni, Chiavi e Valori. Quello che c'interessa è il Valore, ossia il dato che andremo a recuperare da un file ini. Per poter far questo è però necessario indicare la Sezione e la Chiave che individuano tale Valore.

Il concetto è molto facile, quasi banale. Sicuramente alla fine dell'articolo sarete in grado di gestire un file di questo tipo senza problemi, soprattutto dopo aver preso confidenza con l'esempio riportato in fondo al capitolo.

Com'è facile da intuire, *GetPrivateProfileString* recupera i valori dai files INI, mentre *WritePrivateProfileString* scrive eventualmente la Sezione e sicuramente Chiavi e Valori.

Per rendersi conto di ciò basta la semplice applicazione che andremo adesso a sviluppare.

Sarà necessario aprire un progetto EXE Standard.

Trasciniamo quindi dalla sezione *Generale* della finestra di sviluppo di Visual Basic tre controlli TextBox e due CommandButton.

Per una maggiore comprensione del progetto è consigliabile organizzare la form come da figura (eventualmente inserendo sul piano anche le Label, non fondamentali però ai nostri fini):



Per riuscire ad intenderci meglio sui controlli impostare la proprietà *Name* della casella di testo più lunga (quella in alto, contrassegnata dalla Label "Sezione") a "txtSezione".

Facciamo la stessa cosa con la casella di testo a sinistra impostandone la proprietà *Name* a "txtChiave" ed infine impostiamo *Name* della casella più a destra a "txtValore". Allo stesso modo chiamiamo il pulsante a destra cmdCarica e quello a sinistra cmdSalva. Naturalmente la proprietà *Caption* dei controlli è lasciata alla fantasia di ognuno.

Volendo iniziare a lavorare sul codice sarà necessario innanzitutto inserire nella sezione generale del codice (ossia tra prime righe del codice) le dichiarazioni delle API appena viste:

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

Adesso nell'evento *Click* di cmdSalva, inseriamo la procedura da effettuare nel caso in cui si voglia salvare un file INI con tutti i dati al suo interno. Innanzitutto dichiariamo il valore di ritorno, un intero a 32 bit che chiamiamo ValoreDiRitorno e che ci indica se la funzione chiamata ha avuto effetto. Per entrambe le API viste prima infatti un valore di ritorno pari a 0 significa un errore di esecuzione. Dichiariamo inoltre il nome col quale vogliamo salvare il file INI. Ecco allora la prima parte del codice

```
Private Sub cmdSalva_Click()
Dim ValoreDiRitorno As Long
Dim NomeFile
NomeFile = "c:\test.ini"
```

Adesso chiamiamo la funzione *WritePrivateProfileString* per scrivere i dati nel file, associando il valore di ritorno alla variabile numerica che avevamo già dichiarato, ossia ValoreDiRitorno:

```
ValoreDiRitorno = WritePrivateProfileString(txtSezione, _ txtChiave.Text,
txtValore.Text, NomeFile)
```

Notare che i parametri richiesti dalla funzione sono i tre dati che compariranno nel file INI: Sezione, Chiave e Valore ed in più il nome del file che dev'essere salvato.

Come già illustrato in precedenza, un valore di ritorno della *WritePrivateProfileString* pari a 0 è determinato

da un errore. Siccome lavorando coi files un errore può essere difficile da individuare, gestiamo quest'eventualità con un messaggio di avviso.

Chiudiamo infine la sottoprocedura con *End Sub*:

```
If ValoreDiRitorno = 0 Then
MsgBox "Si è verificato un errore!", vbExclamation
End If
End Sub
```

Passando adesso al recupero delle informazioni contenute nel file INI, dichiariamo all'interno dell'evento *Click* di *cmdApri* il valore di ritorno della *GetPrivateProfileString* che chiameremo anche in questo caso *ValoreDiRitorno*, che ha lo stesso scopo del valore di ritorno visto poco fa relativamente all'API *WritePrivateProfileString*. In più dichiariamo il percorso del file da aprire ed una stringa di ritorno, che rappresenta la il Valore che andremo a recuperare dal file INI:

```
Private Sub cmdApri_Click()
Dim ValoreDiRitorno As Long
Dim NomeFile
NomeFile = "c:\test.ini"
Dim StringaDiRitorno As String * 50
```

Recuperiamo dunque il valore richiesto richiamando l'API *GetPrivateProfileString*:

```
ValoreDiRitorno = GetPrivateProfileString(txtSezione, _
txtChiave, NomeFile, StringaDiRitorno, Len(StringaDiRitorno), _
NomeFile)
```

I parametri qui richiesti sono la Sezione, la Chiave, una stringa di default, la variabile nella quale inserire il Valore che otterremo, le dimensioni del Valore da recuperare, pari alla lunghezza della stringa nella quale inseriremo il Valore, ed il percorso del file.

Anche qui sarà necessario gestire l'errore. Se infatti il valore di ritorno della funzione è pari a 0, visualizziamo un messaggio d'errore:

```
If ValoreDiRitorno = 0 Then
MsgBox "Si è verificato un errore!", vbExclamation
```

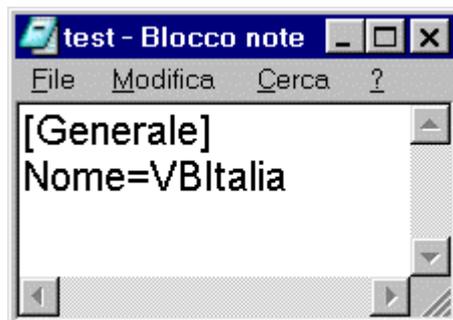
In caso contrario, la casella di testo *txtValore* riporterà il Valore ricercato. Chiudiamo anche in questo caso la sottoprocedura con un *End Sub*:

```
Else
txtValore.Text = Trim(StringaDiRitorno)
End If
End Sub
```

questo punto facciamo un piccolo test. Scriviamo nella casella *txtSezione* "Generale", nella casella *txtChiave* "Nome" e nella *txtValore* "VBItalia". Salviamo il file con la pressione del tasto "Salva".

Adesso andiamo a recuperare il file "c:\test.ini" nella directory C: del computer ed apriamolo con un editor di testo (per default Microsoft NotePad).

Ecco ciò che leggeremo:



Senza cancellare il file INI, scriviamo ancora nella casella txtSezione "Generale", nella casella txtChiave "Anno" e nella txtValore "2001". Salviamo nuovamente il file ed andiamo ad aprirlo per la seconda volta. La Sezione [Generale] si sarà completata con l'aggiunta della Chiave "Anno" e del valore rispettivo "2001":



Per finire cambiamo la Sezione, scrivendo questa volta nella casella txtSezione non più "Generale" ma ad esempio "Utente", nella casella txtChiave "Reg.Key" e nella txtValore "22390-32-15". Ci apparirà quanto segue:



Se volessimo recuperare un Valore in particolare dovremmo a questo punto specificare all'interno dell'applicazione che abbiamo sviluppato, la Sezione che contiene il valore e la Chiave posta in corrispondenza di tale valore.

Nel file ini che abbiamo appena salvato ad esempio, per recuperare il Valore 2001 dovremmo scrivere nella casella di testo txtSezione "Generale" e nella casella txtChiave "Anno":





## Conoscere la struttura di un file INI

Non è detto che a priori si conosca quale sia la composizione di un file .ini. E soprattutto può risultare necessario dover eseguire nella propria applicazione delle istruzioni per tutte le sezioni, le chiavi ed i valori di un file .ini.

Fortunatamente esistono un paio di funzioni che permettono di conoscere tali informazioni automatizzando così i processi di cui si parlava poco sopra.

In particolare la prima funzione in esame è la **GetPrivateProfileSectionNames** la cui dichiarazione corrisponde alla seguente:

```
Declare Function GetPrivateProfileSectionNames Lib "kernel32" Alias  
"GetPrivateProfileSectionNamesA" (ByVal lpReturnedString As String, ByVal nSize  
As Long, ByVal lpFileName As String) As Long
```

Questa funzione ha il compito di recuperare tutti i nomi delle sezioni presenti in un file .ini.

Dove *lpReturnedString* indica il puntatore alla porzione di memoria che riceverà il nome della sezione, *nSize* le dimensioni di tale buffer e *lpFileName* si riferisce al nome del file .ini da cui estrarre le informazioni.

La seconda funzione è denominata **GetPrivateProfileSection** ed ha invece il compito di recuperare le informazioni relative alle chiavi ed ai valori associati ad ogni singola sezione. La sua dichiarazione corrisponde a:

```
Declare Function GetPrivateProfileSection Lib "kernel32" Alias  
"GetPrivateProfileSectionA" (ByVal lpAppName As String, ByVal lpReturnedString  
As String, ByVal nSize As Long, ByVal lpFileName As String) As Long
```

Dove *lpAppName* corrisponde al nome della sezione, *lpReturnedString* al riferimento alla porzione di memoria che andrà a collezionare le informazioni, *nSize* alle dimensioni di tale buffer e, come visto per la funzione precedente *lpFileName* indica il nome del file .ini da cui estrarre le informazioni.

Sicuramente può risultare utile comprendere come queste due funzioni operano e come il buffer di cui si parlava poco sopra venga creato, dimensionato ed infine riempito.

Proprio per questo si analizzerà un esempio purtroppo anonimo preso dalla rete. Nel progetto che segue si vuole convertire un file .INI in un equivalente .XML mantenendone la struttura.

## Convertire un file .INI in .XML

Si supponga di avere un file esempio.ini con la seguente struttura:

```
[GENERALE]  
Chiave1=Valore_x  
chiave2=Valore2  
Chiave3=Valore3
```

```
[SEZIONE 2]  
Chiave1Sezione2 = Valore1Sezione2  
Chiave2Sezione2 = Valore2Sezione2  
chiave3Sezione2 = Valore3Sezione2
```

### [SEZIONE 3]

Chiave1Sezione3 = Valore1Sezione3  
Chiave2Sezione3 = Valore2Sezione3  
Chiave3Sezione3 = Valore3Sezione3

In un modulo .bas si andrà allora ad inserire le seguenti funzioni:

Option Explicit

```
' Win32 API functions used
Private Declare Function GetPrivateProfileSectionNames Lib "kernel32" Alias
"GetPrivateProfileSectionNamesA" (ByVal lpReturnedString As String, ByVal nSize
As Long, ByVal lpFileName As String) As Long
Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias
"GetPrivateProfileSectionA" (ByVal lpAppName As String, ByVal lpReturnedString
As String, ByVal nSize As Long, ByVal lpFileName As String) As Long

' Initial size of the buffer used when calling the Win32 api functions
Const INITIAL_BUFFER_SIZE As Long = 1024

'
'-----
' INI2XML
'
' Parameters:
' strINIFileName
'     File name of the INI file to convert
'
' strXMLFileName
'     File name of the XML file that is created
'
' Returns Boolean:
'     True if successfully, or False if a problem
'     If an exception is raised, it is passed on to the caller.
'
' Purpose:
' Converts an INI file into an XML file.
' Output XML file has the following structure...
' <?xml version="1.0"?>
' <configuration>
'     <section name="Main">
'         <setting name="Timeout" value="90"/>
'         <setting name="Mode" value="Live"/>
'     </section>
' </configuration>
'
' Example:
' If INI2XML(Me.txtIniFileName.Text, Me.txtXMLFileName.Text) Then
'     MsgBox "Successfully converted "" & Me.txtIniFileName.Text & "" to "" &
& Me.txtXMLFileName.Text & """, vbInformation, Me.Caption
' Else
'     MsgBox "Problem converting "" & Me.txtIniFileName.Text & "" to "" &
Me.txtXMLFileName.Text & """, vbExclamation, Me.Caption
' End If
'-----
Public Function INI2XML( _
    ByVal strINIFileName As String, _
    ByVal strXMLFileName As String) As Boolean

    Dim lpSections As String
    Dim nSize As Long
    Dim nMaxSize As Long
    Dim strSection As String
```

```

Dim intSection As Long
Dim intNameValue As Long
Dim strName As String
Dim strValue As String
Dim strNameValue As String
Dim lpNameValues As String
Dim iFile As Integer
Dim intPos As Long

INI2XML = False

If strXMLFileName = "" Then
    ' set the XML's file name based on the INI file name
    intPos = InStrRev(strINIFileName, ".ini", -1, vbTextCompare)
    If intPos > 0 Then
        strXMLFileName = Left(strINIFileName, intPos - 1) & ".xml"
    Else
        strXMLFileName = strINIFileName & ".xml"
    End If
End If

' Get all sections names
' Making sure allocate enough space for data returned
nMaxSize = INITIAL_BUFFER_SIZE / 2
Do
    nMaxSize = nMaxSize * 2
    lpSections = Space(nMaxSize)
    nSize = GetPrivateProfileSectionNames(lpSections, nMaxSize,
strINIFileName)
Loop Until nSize = 0 Or nSize < nMaxSize - 2

' Create XML File
iFile = FreeFile()
Open strXMLFileName For Output As iFile

' Write the opening xml
Print #iFile, "<?xml version=""1.0""?><configuration>"

' Loop through each section
intSection = 0
strSection = GetToken(lpSections, Chr(0), intSection)
Do While strSection <> ""
    ' Write a Node for the Section
    Print #iFile, "<section name="" & strSection & "">"

    ' Get all values in this section, making sure to allocate enough space
    nMaxSize = INITIAL_BUFFER_SIZE / 2
    Do
        nMaxSize = nMaxSize * 2
        lpNameValues = Space(nMaxSize)
        nSize = GetPrivateProfileSection(strSection, lpNameValues, nMaxSize,
strINIFileName)
    Loop Until nSize = 0 Or nSize < nMaxSize - 2

    ' Loop through each Name/Value pair
    intNameValue = 0
    strNameValue = GetToken(lpNameValues, Chr(0), intNameValue)
    Do While strNameValue <> ""
        ' Get the name and value from the entire null separated string of
name/value pairs
        ' Also escape out the special characters, (ie. &"<> )
        strName = XMLEncode(GetToken(strNameValue, "=", 0))
        strValue = XMLEncode(Right(strNameValue, Len(strNameValue) -
Len(strName) - 1))
    Loop
End Do

```

```

' Write the XML Name/Value Node to the xml file
    Print #iFile, "<setting name=""" & strName & """" value=""" &
strValue & """"/>"

    ' Get the next Name/Value pair
    intNameValue = intNameValue + 1
    strNameValue = GetToken(lpNameValues, Chr(0), intNameValue)
Loop

' Close the section node
Print #iFile, "</section>"

' Get the next section name
intSection = intSection + 1
strSection = GetToken(lpSections, Chr(0), intSection)
Loop

' Thats it
Print #iFile, "</configuration>"
Close #iFile

INI2XML = True
End Function

```

```

'-----
' XMLEncode
'
' Parameters:
' strText
'     Text that needs encoding
'
' Returns String:
'     Encoded text
'
' Purpose:
'     Encodes special characters that XML has problems with, ie.
'     Character      Encoded to
'     &              &amp;
'     "              &quot;
'     <              &lt;
'     >              &gt;
'
' Example:
' intSection = 0
' strSection = GetToken(lpSections, Chr(0), intSection)
'-----

```

```

Public Function XMLEncode(ByVal strText As String) As String
    Dim strTextRet As String

    strTextRet = strText

    strTextRet = Replace(strTextRet, "&", "&amp;")
    strTextRet = Replace(strTextRet, """, "&quot;")
    strTextRet = Replace(strTextRet, "<", "&lt;")
    strTextRet = Replace(strTextRet, ">", "&gt;")

    XMLEncode = strTextRet
End Function

```

```

'-----
' GetToken
'
' Parameters:
' strText

```

```

'      Text that is delimited
'
'      strDelimiter
'          The delimiter, eg. ",",
'
'      intIndex
'          The index of the token to return
'          NB. first token is index 0.
'
'      Returns String:
'          Returns the nth token from a string.
'
'      Purpose:
'          Get a token from a delimited string.
'
'      Example:
'      intSection = 0
'      strSection = GetToken(lpSections, Chr(0), intSection)
'-----

```

```

Private Function GetToken( _
    ByVal strText As String, _
    ByVal strDelimiter As String, _
    ByVal intIndex As Long) As String

    Dim i As Long
    Dim intLen As Long
    Dim intPos As Long
    Dim intStart As Long
    Dim intLastPos As Long
    Dim intPreviousPos As Long
    Dim intDelimiterLen As Long
    Dim strToken As String

    i = 0
    intPos = 0
    intLastPos = 0
    intPreviousPos = 0
    intDelimiterLen = Len(strDelimiter)
    For i = 0 To intIndex
        intPos = InStr(intLastPos + intDelimiterLen, strText, strDelimiter)
        If intPos > 0 Then
            ' found a delimiter
            intPreviousPos = intLastPos
            intLastPos = intPos

            intStart = intPreviousPos + intDelimiterLen
            intLen = intPos - intStart
        Else
            ' delimiter not found
            intPreviousPos = intLastPos
            If intLastPos > 0 Then
                intLen = Len(strText) - intLastPos
            Else
                ' must be no delimiters in the text
                intLen = Len(strText)
            End If
            intStart = intPreviousPos + intDelimiterLen
        End For
        Exit For
    End If
    Next

    If i >= intIndex Then
        ' means I've found the requested token
        strToken = Mid(strText, intStart, intLen)
    End If
End Function

```

```

Else
    ' Token not found, so return blank string
    strToken = ""
End If

GetToken = strToken
End Function

```

Ed infine nel modulo di codice di Form1:

```

Option Explicit
'
Private Sub btnConvert_Click()
    On Error GoTo HandleError

    If INI2XML(Me.txtIniFileName.Text, Me.txtXMLFileName.Text) Then
        MsgBox "Successfully converted "" & Me.txtIniFileName.Text & "" to "" & Me.txtXMLFileName.Text & """, vbInformation, Me.Caption
    Else
        MsgBox "Problem converting "" & Me.txtIniFileName.Text & "" to "" & Me.txtXMLFileName.Text & """, vbExclamation, Me.Caption
    End If
    Exit Sub

HandleError:
    MsgBox "Error# " & Err.Number & vbCrLf & Err.Description, vbExclamation, Err.Source
End Sub

Private Sub Close_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    Me.txtIniFileName.Text = App.Path & "\Sample.ini"
    Me.txtXMLFileName.Text = App.Path & "\Sample.xml"
End Sub

```

## I file .INI e Visual Basic .NET

Anche Visual Basic .NET permette l'accesso in lettura e scrittura ai file .INI, proprio come Visual Basic 6. In realtà però ci sono alcune piccole modifiche che devono essere apportate al codice. Ad esempio, se si analizza la dichiarazione delle due funzioni principali: la `GetPrivateProfileString` e la `WritePrivateProfileString`, si deve tenere a mente che è necessario dichiararle come Unicode e che i valori indicati "As Any" nella dichiarazione di VB6 sono sostituiti da una dichiarazione "As String". In conclusione ecco la dichiarazione delle due funzioni nel caso di Visual Basic .NET:

```

'la funzione GetPrivateProfileString dichiarata come Unicode
Private Declare Unicode Function GetPrivateProfileString Lib "kernel32" _
Alias "GetPrivateProfileStringW" (ByVal lpApplicationName As String, _
ByVal lpKeyName As String, ByVal lpDefault As String, _
ByVal lpReturnedString As String, ByVal nSize As Int32, _
ByVal lpFileName As String) As Int32

'la funzione WritePrivateProfileString dichiarata come Unicode
Private Declare Unicode Function WritePrivateProfileString Lib "kernel32" Alias
"WritePrivateProfileStringW" (ByVal lpApplicationName _

```

```
As String, ByVal lpKeyName As String, ByVal lpString As String, _  
ByVal lpFileName As String) As Long
```

Un esempio scaricabile di come sia possibile accedere ad un file .INI è presente sul sito [www.vbitalia.it](http://www.vbitalia.it).  
L'immagine sottostante mostra il risultato ottenuto utilizzando tale esempio:

Form1

C:\Documents and Settings\Administrator\Documenti\Progetti di Visual Studio\INI\bin\Impostazioni.ini

Scritto:  
[GENERALE]  
Chiave1=Valore\_X

Scrivi Leggi

SEZIONE GENERALE

CHIAVE Chiave1

NUOVO VALORE Valore\_X

Ad ogni modo, tralasciando l'implementazione dell'interfaccia grafica se ne può analizzare il codice che corrisponde al seguente:

```
'la funzione GetPrivateProfileString dichiarata come Unicode  
Private Declare Unicode Function GetPrivateProfileString Lib "kernel32" _  
Alias "GetPrivateProfileStringW" (ByVal lpApplicationName As String, _  
ByVal lpKeyName As String, ByVal lpDefault As String, _  
ByVal lpReturnedString As String, ByVal nSize As Int32, _  
ByVal lpFileName As String) As Int32
```

```
'la funzione WritePrivateProfileString dichiarata come Unicode  
Private Declare Unicode Function WritePrivateProfileString Lib "kernel32" Alias  
"WritePrivateProfileStringW" (ByVal lpApplicationName _  
As String, ByVal lpKeyName As String, ByVal lpString As String, _  
ByVal lpFileName As String) As Long
```

```
Private FileName As String
```

```
Public Function ReadIniData(ByVal Section As String, ByVal Key As String) As  
String  
Dim Valore As Long  
DimRetVal As String = Space(256)  
Valore = GetPrivateProfileString(Section, Key, "<Nessun valore>",RetVal,  
RetVal.Length, FileName)  
ReadIniData = Trim(RetVal.ToString)  
TextBox1.Text = ReadIniData  
End Function
```

```

Public Function WriteIniData(ByVal Section As String, ByVal Key As String) As
String
Dim Valore As Long
Dim RetVal As String = Space(256)
Valore = WritePrivateProfileString(Section, Key, TextBox4.Text, FileName)
WriteIniData = Trim(RetVal.ToString)
TextBox1.Text = "Scritto: " & vbCrLf & "[" & Section & "]" & vbCrLf & _
TextBox3.Text & "=" & TextBox4.Text
End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
ReadIniData(TextBox2.Text, TextBox3.Text)
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
FileName = System.AppDomain.CurrentDomain.BaseDirectory() & "Impostazioni.ini"
Label1.Text = FileName
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
WriteIniData(TextBox2.Text, TextBox3.Text)
End Sub

```