Leggere un File Txt con VBA

Ci sono vari modi di leggere un file Txt da VBA, vediamo un modo che credo sia facile da capire e modificare ma che richiede la conoscenza di un paio di concetti di VBA come la progettazione delle macro e la scelta dei metodi. Per meglio comprendere, vediamo quando, perché e come scegliere la funzione giusta, e una corretta gestione degli errori e delle decisioni da intraprendere a livello codice utilizzando una tecnica e metodi differenti. Cominciamo con l'aggiunta dei riferimenti necessari al progetto VBA, aprite un nuovo file Excel a dall'editor di VB seguite il percorso dal menu **Strumenti – Riferimenti**, scorrete la lista fino a trovare la libreria *Microsoft Scripting Runtime library* e spuntare la casella e confermate l'operazione cliccando sul tasto Ok. La Microsoft Scripting Runtime library ci permette di utilizzare:

FileSystemObject: Con questo oggetto si può accedere rapidamente a qualsiasi unità, cartella o file.
TextStream: E' un oggetto usato per leggere il contenuto di un file Txt

Creiamo ora un file Txt aprendo notepad o un altro editor di testo e copiate e incollate i dati di esempio qui sotto

```
Questa è la riga uno
questa invece è la seconda linea
è questa è la linea n ° 3
ecco la quarta riga del file
e finiamo con la Linea 5
```

Salvate il file col nome di *prova.txt* sul desktop e ritornate nell'editor di VBE per inserire un nuovo modulo, seguendo il percorso dal menu **Inserisci – Modulo** e copiate e incollate il codice sotto riportato

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
End Sub
```

Ora se si esegue la procedura *leggi_txt* si ottiene un messaggio di errore del tipo

Fig. 1

E cliccando sul pulsante *Debug* si evidenzia la riga che ha rimandato l'errore

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
End Sub
```

Fig. 2

L'errore rimandato è: *Impossibile trovare il file*. Un occhio attento si sarebbe accorto dell'errore prima di eseguire la procedura in quanto è evidente la differenza dei percorsi del file, ma è stato simulato questo errore per introdurre un modo per gestire questo errore. All'inizio è stato citato il metodo come semplice e ora vediamo di applicare questa logica.
Come si può vedere il metodo *OpenTextFile ()* accetta un parametro che è il percorso del file, ma la procedura non controlla se il file esiste o no, tocca al programmatore fornire un corretto percorso di ingresso alla funzione per evitare errori *Runtime*. Per essere sicuri che il file esista si possono usare vari metodi, ma il metodo più semplice sarebbe di pilotare il codice in modo che facesse questa valutazione: *se il file esiste puoi proseguire, se non non esiste esci*.

Quindi dobbiamo aggiungere qualcosa al codice per verificare se il file esiste, e lo possiamo fare in due modi, uno consiste nel creare una funzione che esegua il controllo e restituisca un valore Booleano True o False, oppure utilizzare un metodo già esistente nell'oggetto FileSystemObject che corrisponde al metodo *FileExists*. Utilizzando il metodo FileExists si può aggiungere un'istruzione *IF – Else* al codice attuale che prima di aprire il file Txt verifichi se il percorso è corretto. In pratica usando un'espressione del genere

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    If <pathisvalid> Then
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End Sub
```

Dove *<pathisvalid>* è solo un segnaposto per l'espressione che deve controllare se il file esiste oppure no, in altre parole, se il percorso è valido, pertanto sostituiamo il segnaposto con il comando alex.FileExist ("C\Test\prova.txt") e modifichiamo il listato in questo modo

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream

    If alex.FileExists("C:\Test\prova.txt") Then
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End Sub
```

Fig. 3

In alternativa, possiamo usare la funzione Dir per verificare se il file esiste in questo modo: *If Dir("C:\Test\prova.txt") <> vbNullString Then*. Se si vuole usare *Dir ()* vorrei suggerire di aggiungere una funzione di tipo booleano al codice, che restituisca un valore vero o falso se il percorso è corretta o meno. Converrete che ricevere un parametro di ritorno vero o falso sia quantomeno azzeccato all'operazione che si sta cercando di fare. Possiamo ora definire questa funzione

```
Function FileExist(filePath As String) As Boolean
    If Dir(filePath) <> vbNullString Then
        FileExist = True
    Else
        FileExist = False
    End If
End Function
```

Oppure con una sintassi diversa

```
Function fileExist(filePath As String) As Boolean
    fileExist = IIf(Dir(filePath) <> vbNullString, Vero, Falso)
End Function
```

Personalmente ritengo che non abbiamo assolutamente bisogno di quest'ultimo listato che utilizza il segnaposto (o placeholder) IIF, in quanto sappiamo che la funzione *FileExist* fa esattamente la stessa cosa, ma soprattutto vale la pena di sapere come gestire situazioni simili, creando le proprie funzioni e questo è un ottimo modo. E 'sempre una buona regola assegnare tutti i dati a variabili, se si memorizza il percorso in un'unica variabile nel codice, invece di ripeterlo in vari contesti, il codice diventerà più facile da gestire, in fase di test e di debug.

Cosa succede se il percorso o il nome del file cambia? Per non scorrere l'intero progetto alla ricerca di tutte le occorrenze di percorso sarebbe molto meglio e più facile da gestire se si usa una variabile che identifica il percorso del file e in caso di modifica si edita solo la variabile e il resto del codice rimane intatto.

Pertanto aggiungiamo una variabile per memorizzare il percorso, creando una variabile String denominata *filePath* a cui assegniamo il valore di "C:\Test\prova.txt"

```
Dim filePath As String
filePath = "C:\Test\prova.txt"
```

Da notare che la variabile filePath viene utilizzata nella Function *FileExist* e *OpenTextFile*, così ora il codice diventa:

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Test\prova.txt"
    If alex.FileExists(filePath) Then
   Set alex_1 = alex.OpenTextFile(filePath)
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End Sub
```

Ora abbiamo costruito la procedura che apre il file se il percorso è corretto, oppure visualizza un messaggio di errore se il percorso del file non è corretto. Però questa procedura andrebbe bene se sul pc che viene utilizzato ci sia un solo utente che si logga all'apertura di Windows, oppure, se si sposta il percorso di ricerca sul desktop, servono altri parametri, oppure se ci fossero più utenti che si loggano sul pc. In questo caso verrebbe rimandato un errore di percorso non trovato a meno che non modifichiamo il listato usando una funzione che restituirebbe il nome attualmente loggato in Windows. Si tratta di usare la funzione [i[Environ ()[/i] che per essere usata si deve modificare il percorso del file in questo modo:

```
filePath = "C:\Users\" & Environ$("Username") & "\Test\prova.txt"
```

Se si esegue questo codice la macro apre e chiude con successo il file prova.txt, ma non possiamo avere nessuna certezza, pertanto possiamo aggiungere una dichiarazione temporanea al codice per verificare se il file è effettivamente aperto.

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream

    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"

    If alex.FileExists(filePath) Then
   Set alex_1 = alex.OpenTextFile(filePath)
   Debug.Print IIf(alex_1 Is Nothing, "Il file è chiuso", "Il file è aperto")
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End SubFunction fileExist(filePath As String) As Boolean
    If Dir(filePath) <> vbNullString Then
       fileExist = True
    Else
       fileExist = False
    End If
End Function
```

Se il codice sopra esposto lo eseguiamo è possibile vedere se il file viene aperto oppure no, pertanto attiviamo la *Finestra Immediata* dal menu **Visualizza – Finestra immediata** oppure cliccando la combinazione di tasti **CTRL + G** e si dovrebbe vedere che il file viene aperto dopo aver eseguito la macro

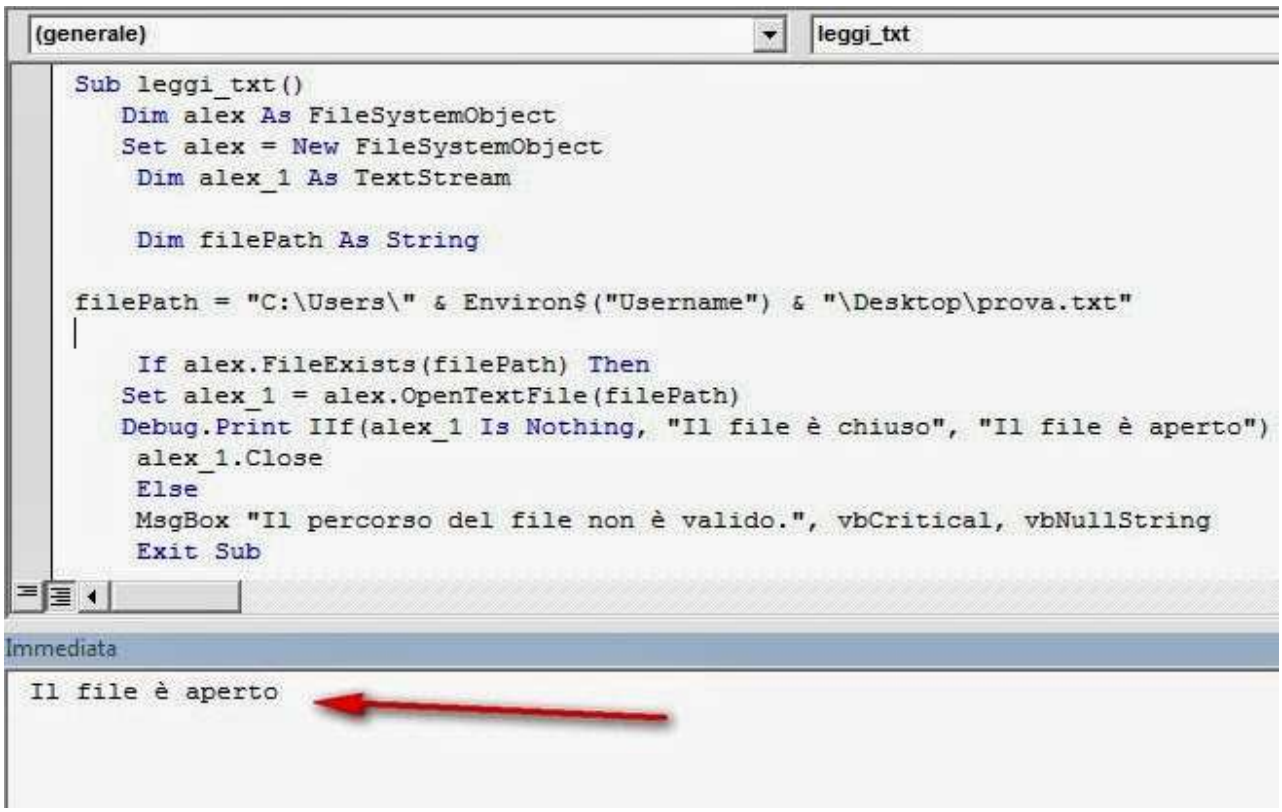


Fig. 4

Come si può vedere nella figura sopra riportata il codice sta funzionando benissimo, ma cosa accadrebbe se ci fosse un errore nel file prova.txt, oppure se fosse stato danneggiato, oppure se il metodo OpenTextFile non è riuscito a leggere il file. Certamente sarebbe apparso un messaggio minaccioso di errore di runtime.
Si dovrebbe sempre prendere in considerazione uno scenario simile se si lavora con i file, se qualcosa va storto nell'esecuzione del codice il programma si ferma immediatamente, così, tutti gli oggetti aperti, tutti i riferimenti sarebbe rimasti aperti e molto probabilmente non saremmo in grado di liberare la memoria a meno che il *garbage collection* (modalità automatica di gestione della memoria nei sistemi Windows) non ci fosse venuto in aiuto.

Questo è un caso in cui l'istruzione *On Error GoTo <label>* è molto utile, pertanto proviamo ad applicare il gestore degli errori al nostro esempio. Considerate le seguenti modifiche

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"

    If alex.FileExists(filePath) Then
    On Error GoTo Err
   Set alex_1 = alex.OpenTextFile(filePath)
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If

Err:
    MsgBox "Errore durante la lettura del file.", vbCritical, vbNullString
    alex_1.Close
    Exit Sub
End Sub
```

Se si verifica un errore durante la lettura del file l'istruzione *On goto errore Err* fermerà l'esecuzione del codice e salterà all'etichetta *Err* che mostrerà un messaggio di errore, per poi uscire dalla macro attualmente in esecuzione. Passiamo ora alla lettura del contenuto del file prova.txt. I tre approcci più comuni per leggere il contenuto di un file txt sono i metodi Read ReadAll e ReadLine applicabili all'oggetto TextStream.

Il metodo .ReadAll è molto facile da usare, se inseriamo una riga di codice dopo la chiamata OpenTextFile di questo tipo: *Range("A1") = alex_1.ReadAll*
Si recupera il contenuto del file prova.txt e lo si inserisce nella prima cella in alto a sinistra (A1) del foglio di calcolo attivo.

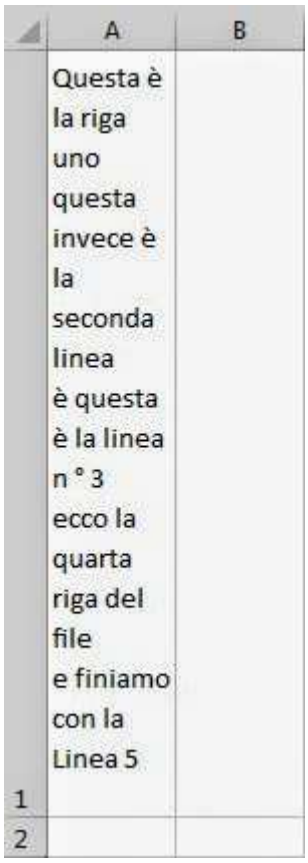


Fig. 5

Se si clicca sulla cella A1 e si osserva la barra della formula si sarà in grado di vedere meglio in quale formato avete ricevuto i vostri dati. Dovrebbe assomigliare a questa

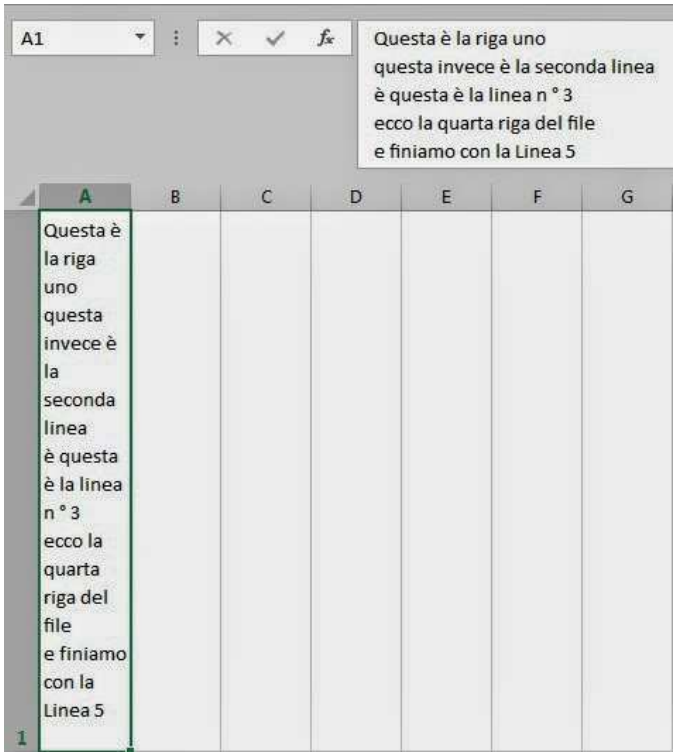


Fig. 6

Come si può vedere il metodo *ReadAll* recupera tutti i dati da un file di testo in una volta nello stesso formato che sono disposti nel file di testo. E' ovvio che ricevere dei dati in questo formato non è un buon approccio e non consentono una manipolazione degli stessi, ma possiamo usare la funzione *Read*, la quale accetta un parametro che corrisponde al numero di caratteri che si desidera leggere. Se modifichiamo il codice, sostituendo *alex_1.ReadAll* con*alex_1.Read(5)*, verranno restituiti i primi 5 caratteri dal file di testo nella cella A1. Potete cambiare il valore 5 in qualsiasi altro numero e se si supera il numero di caratteri nel file di testo verrà recuperato l'intero file.

Anche questo metodo non presenta le caratteristiche desiderate per importare dei dati e manipolarli, non possiamo sapere quanti caratteri dobbiamo importare per avere dei dati di senso compiuto, se si tratta di dati in formato stringa oppure avere dei valori numerici che non vengano troncati. Esiste un approccio diverso per leggere il file di testo, usando l'istruzione *ReadLine* e come suggerisce il nome stesso permette di leggere il file riga per riga e possiamo con le adeguate istruzioni depositare i dati in celle separate. Se modifichiamo il codice in questo modo

```
Set alex_1 = alex.OpenTextFile(filePath)
    Do While Not alex_1.AtEndOfStream
    Range("A" & Range("A" & Rows.Count).End(xlUp).Row + 1) = alex_1.ReadLine
    Loop
    alex_1.Close
```

Il risultato sarà come il seguente

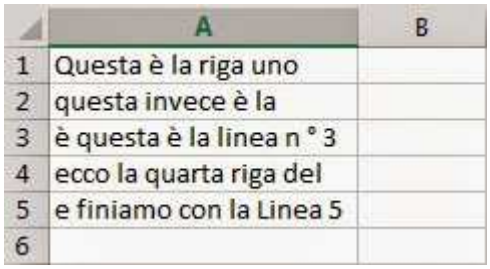
	A	B
1		
2	Questa è la riga uno	
3	questa invece è la	
4	è questa è la linea n ° 3	
5	ecco la quarta riga del	
6	e finiamo con la Linea 5	
7		

Fig. 7

Come si può notare la prima riga è stata lasciata vuota, che potrebbe ospitare le intestazioni dei campi, ma se si osserva l'istruzione *Range("A" & Range("A" & Rows.Count).End(xlUp).Row + 1)* , se si esegue il codice due volte, alla successiva esecuzione i dati verrebbero incollati sotto l'ultima riga utilizzata nella colonna A, in alternativa è possibile utilizzare una variabile per sovrascrivere i dati in questo modo:

```
Set alex_1 = alex.OpenTextFile(filePath)
   Dim i As Long
   i = 1
  Cells.ClearContents
    Do While Not alex_1.AtEndOfStream
   Range("A" & i) = alex_1.ReadLine
    i = i + 1
    Loop
    alex_1.Close
```

e si ottiene un risultato come il seguente:



	A	B
1	Questa è la riga uno	
2	questa invece è la	
3	è questa è la linea n ° 3	
4	ecco la quarta riga del	
5	e finiamo con la Linea 5	
6		

Fig. 8

Concludendo ormai si dovrebbe facilmente essere in grado di aprire qualsiasi file txt con codice VBA ed estrarre i dati da esso, vediamo il codice per intero nei due approcci presentati

```
Sub leggi_txt()
    Dim alex As FileSystemObject
    Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"
    If alex.FileExists(filePath) Then
        On Error GoTo Err
        Set alex_1 = alex.OpenTextFile(filePath)
   Dim i As Long
   i = 1
  Cells.ClearContents
    Do While Not alex_1.AtEndOfStream
   Range("A" & i) = alex_1.ReadLine
    i = i + 1
    Loop
    alex_1.Close
    Else
        MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
        Exit Sub
    End If
    Exit Sub
Err:
    MsgBox "Errore durante la lettura del file.", vbCritical, vbNullString
    alex_1.Close
    Exit SubEnd Sub

Function fileExist(filePath As String) As Boolean
    If Dir(filePath) <> vbNullString Then
        fileExist = True
    Else
        fileExist = False
    End If
End Function
```

Oppure

```
Sub leggi_txt()
    Dim alex As New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"
    If Not fileExist(filePath) Then GoTo FileDoesntExist
    On Error GoTo Err
    Set alex_1 = alex.OpenTextFile(filePath)
    With Sheets(1).Range("A1")
        .ClearContents
        .NumberFormat = "@"
        .Value = alex_1.ReadAll
    End With
    alex_1.Close
    Exit Sub

FileDoesntExist:
    MsgBox "Il File non esiste", vbCritical, "File not found!"
    Exit Sub

Err:
    MsgBox "Errore durante la lettura del file.", vbCritical, vbNullString
    alex_1.Close
    Exit Sub
End Sub

Function fileExist(path As String) As Boolean
    fileExist = IIf(Dir(path) <> vbNullString, True, False)
End Function
```