



Visualizza il contenuto delle cartelle remote tramite due programmi Client/Server (prima parte)

http://www.vbsimple.net/cliserv/clser_05.htm

Difficoltà:  4 / 5

Vedremo come scrivere una funzione che mostri in un programma client l'elenco delle cartelle e dei files presenti sul computer in cui è presente il server.

Passiamo subito alla costruzione della parte Server, ovvero la parte del progetto che si occupa di fornire i dati e trasmetterli alla parte Client. Nel nostro progetto vogliamo leggere la struttura delle cartelle presenti sul computer dove viene eseguito il Server. Tale struttura verrà inviata al programma client che si occuperà di visualizzarla nella maniera più adeguata.

Tutte le comunicazioni avverranno via TCP/IP sulla **porta 1500**.

Il server si compone di tre soli controlli posizionati sopra un form di nome **ServerForm**.

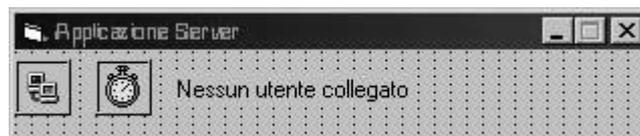


Figura 1

Abbiamo innanzitutto un controllo Winsock di nome **Server**. Impostiamo la sua proprietà *LocalPort* a 1500.

Inseriamo anche un controllo Timer  di nome **TimeOut**. Esso controllerà l'attività del server; se per 60 secondi non vengono inviati dati dal programma client, il controllo TimeOut provvederà a chiudere la connessione. Questo genere di controlli sono obbligatori, particolarmente in connessioni Dial-Up via modem. Se si dovesse disconnettere la linea, in alcuni casi, non riusciremmo più a connetterci al server perché la connessione risulterebbe già effettuata. In questo modo basterà aspettare fino a 60 secondi dall'ultimo comando inviato per potersi riconnettere al server. Impostiamo quindi la proprietà *Interval* a 60000 millisecondi, ovvero 60 secondi.

L'ultimo controllo presente sul nostro form è una Label di nome **StatoConnessione**. Essa visualizzerà se la connessione è stabilita oppure se non vi è nessun utente collegato al server.

Prima di passare al codice vero e proprio analizziamone il funzionamento.

All'avvio del programma il server si mette in ascolto sulla porta 1500.

Nel momento in cui avviene una richiesta di connessione scatta l'[evento](#)

ConnectionRequest dell'oggetto Server. All'arrivo della richiesta il server risponde con l'accettazione della chiamata. L'accettazione della chiamata comporta l'avvenuta

connessione, per cui all'interno di tale evento cambiamo anche la proprietà *Caption* della Label **StatoConnessione** ed avviamo il Timer **TimeOut**.

Il server autonomamente non invierà nessun dato al Client. Esso resterà fermo fintanto che non sarà inviato dal Client qualche comando. La ricezione di dati comporta l'esecuzione dell'evento *DataArrival*.

Tutti i dati inviati al server devono necessariamente terminare con un INVIO ([costante](#) `vbNewLine`). La motivazione è molto semplice: esistono svariati programmi di connessione - un esempio lampante è il Telnet di Windows - che inviano i dati carattere per carattere. Per cui al server non arriverebbero mai stringhe complete ma solo tante serie composte da un solo carattere. Per questo motivo tutti i dati ricevuti vengono immagazzinati in un [buffer](#) finché l'ultimo carattere ricevuto non è il tasto INVIO. Esso decreta la fine di una riga di comando.

Quando il comando si completa con l'INVIO esso sarà analizzato da una funzione che si occupa di effettuare le operazioni legate al comando. Subito dopo l'esecuzione del comando il buffer dei caratteri ricevuti deve essere svuotato per permettere l'esecuzione di un altro comando.

Inoltre ogni volta che viene inviato qualche dato al Server il Timer **TimeOut** deve ricominciare il suo conteggio.

Vediamo ora il codice:

```
1. Option Explicit
2. Private DATIINARRIVO As String
```

La variabile DATIINARRIVO è il buffer in cui vengono immagazzinati tutti i caratteri ricevuti fino a quando non viene inviato un INVIO.

```
4. Private Sub Form_Load()
5.     Server.Listen
6. End Sub
```

All'avvio del programma il server viene messo in ascolto sulla porta indicata dalla proprietà *LocalPort*, ovvero la 1500.

```
8. Private Sub Form_Unload(Cancel As Integer)
9.     Server.Close
10. End
11. End Sub
```

Nel momento della chiusura del server vengono chiuse tutte le connessioni e viene forzata la chiusura del programma.

```
13. Private Sub Server_Close()
14.     Server.Close
15.     Server.Listen
16.     StatoConnessione.Caption = "Nessun utente collegato"
17. End Sub
```

Questa routine indica la chiusura della connessione. La riga 14 è presente per il fatto che alcune funzioni più in basso fanno riferimento a quest'evento, ma senza tale riga la

chiamata ad esso non assicura la chiusura della connessione.
È soltanto una semplificazione scritturale.

Alla chiusura della connessione il server si rimette in ascolto per attendere un'ulteriore connessione e nel frattempo cambia il contenuto della Label **StatoConnessione**.

```
19. Private Sub Server_ConnectionRequest(ByVal requestID As Long)
20.     Server.Close
21.     Server.Accept requestID
22.     DoEvents
23.     Timeout.Enabled = True
24.     StatoConnessione.Caption = "Un utente collegato"
25. End Sub
```

Nel momento in cui il server riceve una richiesta di connessione scatta quest'evento. Per evitare alcuni errori di connessione già effettuata è stata inserita un'istruzione di chiusura della connessione alla riga 20.

In seguito a quella avviene l'accettazione della chiamata tramite il metodo *Accept*. Da questo momento in poi scatta il contatore dei 60 secondi prima della chiusura della connessione e viene aggiornato il contenuto della Label **StatoConnessione**.

```
27. Private Sub Server_DataArrival(ByVal bytesTotal As Long)
28.     Dim DATI As String
29.     Timeout.Enabled = False
30.     On Error Resume Next
31.     Server.GetData DATI, vbString, bytesTotal
32.     DATIINARRIVO = DATIINARRIVO & DATI
33.     If Right(DATIINARRIVO, 2) = vbNewLine Then
34.         GestisciComandi Left(DATIINARRIVO, Len(DATIINARRIVO) - 2)
35.         DATIINARRIVO = ""
36.     End If
37.     Timeout.Enabled = True
38. End Sub
```

Questo evento viene eseguito ogni volta che il server riceve dei dati. Per la motivazione vista prima, i dati vengono memorizzati in un buffer di nome **DATIINARRIVO** fino a quando l'ultimo carattere inviato non è un INVIO ovvero *vbNewLine*.

La ricezione di un dato comporta l'azzeramento del contatore **TimeOut** effettuato tramite la riga 29.

I dati inviati - siano essi singoli caratteri o parole complete - vengono ricevuti tramite l'istruzione *GetData* della riga 31. Tali dati saranno comunque accodati al contenuto del buffer.

Se l'ultimo carattere contenuto nel buffer è INVIO, allora il buffer dovrà essere analizzato per controllare se esso contiene dei comandi validi per il serve.

Alla riga 33 c'è scritto di estrarre gli ultimi **due** caratteri e non soltanto l'ultimo perché nei sistemi Windows il tasto INVIO è composto da due caratteri ASCII. La costante **vbNewLine** ha infatti la lunghezza di due bytes.

Se l'ultimo tasto inviato è INVIO verrà eseguita la funzione *GestisciComandi*, che analizza il contenuto del buffer.

In seguito all'analisi dei parametri il buffer viene azzerato per permettere la ricezione di nuovi comandi (riga 35).

Dopo quest'operazione viene riattivato il contatore **TimeOut** che permette la disconnessione automatica dopo 60 secondi di inattività.

```
40. Private Sub TimeOut_Timer()  
41.     Server_Close  
42.     TimeOut.Enabled = False  
43. End Sub
```

Infatti dopo 60 secondi di inattività scatta l'evento *Timer* del controllo **TimeOut**. Esso effettua la disconnessione tramite la funzione *Server_Close* (che al suo interno racchiude l'istruzione di disconnessione).

```
45. Private Sub GestisciComandi(ByVal COMANDO As String)  
46.     Dim VALORE As String  
47.     On Error Resume Next  
48.     If InStr(1, COMANDO, " ") <= 0 Then  
49.         Select Case UCase(COMANDO)  
50.             Case "QUIT": Server_Close  
51.             Case "BYE": Server_Close  
52.             Case "DIR": LeggiFilesDir CurDir, True, True  
53.             Case "DIRS": LeggiFilesDir CurDir, True, False  
54.             Case "DIRF": LeggiFilesDir CurDir, False, True  
55.             Case Else: Server.SendData " -ERR: Comando sconosciuto -> " & COMANDO &  
vbNewLine  
56.         End Select  
57.     Else  
58.         VALORE = Trim(Mid(COMANDO, InStr(1, COMANDO, " ") + 1))  
59.         If UCase(Left(COMANDO, InStr(1, COMANDO, " ") - 1)) = "CD" Then  
60.             ChDir VALORE  
61.             Server.SendData " +OK: La cartella di lavoro é: " & CurDir & vbNewLine  
62.         End If  
63.     End If  
64. End Sub
```

Questa funzione si occupa di smistare i dati ricevuti dal controllo *Server*. Essa innanzitutto controlla se all'interno del comando inviato ci sia uno spazio separatore. La sua presenza dovrebbe indicare che tale comando richiede dei parametri. Se non vi è spazio separatore viene eseguito un test tramite l'istruzione *Select Case*.

Seguono una serie di possibili comandi, come la chiusura della connessione o l'elencazione di tutti i files, dei soli files o delle sole cartelle contenute nella directory di lavoro. Se il comando non rientra in quelli elencati precedentemente viene inviato al client un messaggio di errore.

I messaggi che invia il server iniziano con " +OK:" se l'esecuzione del comando è esatta oppure con " -ERR:" se l'esecuzione non è andata a buon fine.

Per ragioni di compatibilità in ogni caso alla fine di ogni risposta da parte del server segue un carattere di INVIO (*vbNewLine*).

Se il comando passato alla funzione **GestisciComandi** contiene almeno uno spazio viene separato il comando vero (quello precedente allo spazio) dai parametri che seguono lo spazio. Il parametro viene memorizzato in una variabile di nome **Valore**.

Questo server gestisce soltanto un comando parametrico: il CD, ovvero la richiesta di cambio della cartella di lavoro. Viene per cui effettuato un controllo se il comando parametrico è CD. In caso di risposta affermativa viene cambiata la directory di lavoro tramite l'istruzione *ChDir* e viene inviato un messaggio di conferma con la nuova cartella.

```
Public Sub LeggiFilesDir(ByVal CARTELLA As String, ByVal CARTELLE
As Boolean, ByVal FILES As Boolean)
```

L'ultima funzione del programma Server è la **LeggiFilesDir** che effettua la scansione di una cartella e restituisce al programma client una riga contenente la lista dei files e/o cartelle contenuti in essa. La funzione richiede tre parametri: il primo (**CARTELLA**) è la cartella di cui effettuare la scansione; il secondo (**CARTELLE**) determina se devono essere restituite le cartelle. Se il suo valore è False tutte le cartelle saranno escluse dalla scansione. L'ultimo parametro (**FILES**) indica se devono essere restituiti i nomi dei files. Se ad esempio il parametro **CARTELLE** è True ed il parametro **FILES** è False, saranno restituite soltanto le cartelle contenute nella directory di lavoro. Quando, invece entrambi i parametri sono impostati a True saranno restituiti sia i files che le cartelle contenuti nella directory di lavoro.

```
66. Public Sub LeggiFilesDir(ByVal CARTELLA As String, ByVal CARTELLE As Boolean, ByVal
FILES As Boolean)
67.     Dim ELENCOFILESDIR As String
68.     Dim TEMPSTR As String
69.     Dim VALIDO As Boolean
70.     ELENCOFILESDIR = ""
71.     If Right(CARTELLA, 1) = "\" Then CARTELLA = Left(CARTELLA, Len(CARTELLA - 1))
72.     TEMPSTR = Dir(CARTELLA & "\*.*", vbArchive + vbDirectory + vbHidden + vbNormal
+ vbReadOnly + vbSystem)
73.     While TEMPSTR <> ""
74.         VALIDO = False
75.         If (CARTELLE = True) And (GetAttr(CARTELLA & "\" & TEMPSTR) And
vbDirectory) = vbDirectory Then VALIDO = True
76.         If (FILES = True) And (GetAttr(CARTELLA & "\" & TEMPSTR) And vbDirectory)
<> vbDirectory Then VALIDO = True
77.         If VALIDO = True Then ELENCOFILESDIR = ELENCOFILESDIR & vbNewLine & TEMPSTR
78.         TEMPSTR = Dir
79.     Wend
80.     Server.SendData " +OK: " & ELENCOFILESDIR & vbNewLine
81. End Sub
```

La variabile **ELENCOFILESDIR** conterrà l'elenco di tutti i files o le cartelle rilevate nella scansione, ognuna separata da INVIO (vbNewLine). Alla riga 70 tale variabile viene inizializzata con una stringa nulla.

Alla riga 71 viene effettuato un controllo: se l'ultimo carattere del parametro **CARTELLA** è "\" allora viene in automatico scartato prendendo tutti i caratteri di **CARTELLA** tranne l'ultimo.

Alla riga 72 inizia la vera e propria scansione.

Per il nostro scopo servono sia i files sia le cartelle. Saranno poi i parametri **CARTELLE** e **FILES** che decideranno se gli uni o gli altri debbano essere restituiti al client. Questa riga utilizza la funzione *Dir* per elencare tutti i files o cartelle (*.*) con qualunque attributo e memorizza il primo di essi nella variabile **TEMPSTR**.

Alla riga 73 viene azzerato il contenuto della variabile **VALIDO**. Essa servirà per determinare se l'elemento estratto dalla directory di lavoro deve essere aggiunto all'elenco.

Segue un ciclo che si ripeterà finché esistono files o cartelle nella directory di lavoro.

All'interno del ciclo vengono effettuati dei controlli: se il parametro CARTELLE è True **E** l'elemento estratto (contenuto in TEMPSTR) è una cartella, segna l'attributo **VALIDO** a True.

Altresì viene controllato se il parametro FILES è True **E** l'elemento estratto **non** è una cartella (ossia è un file - all'interno di una directory ci sono soltanto files e cartelle; tutto ciò che non è cartella è file) viene segnato l'attributo **VALIDO**.

Alla riga 77 viene controllato se l'attributo **VALIDO** è True; in caso positivo il file indicato nella variabile **TEMPSTR** viene aggiunto all'elenco degli elementi, separato dal precedente da un INVIO.

L'ultima riga prima della fine del ciclo (riga 78) cerca l'elemento successivo nella scansione.

Se ci sono altri elementi il ciclo si ripete, altrimenti il ciclo termina.

Subito dopo la fine del ciclo viene inviato al programma client l'elenco degli elementi della scansione nella solita forma: " +OK: " e l'INVIO alla fine della riga.

La parte server di questo programmino termina qui.

Nella prossima parte della richiesta vedremo come sviluppare la parte client che riceverà i dati inviati dal Server.

[Segue parte 2 >>](#)

[Fibia FBI](#)
4 Giugno 2001



[Torna all'indice Client / Server](#)
