

[Home Page](#) [Informazioni](#) [Aiuto](#) 

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

http://www.vbsimple.net/cliserv/clser_05_2.htm

Difficoltà:  4 / 5

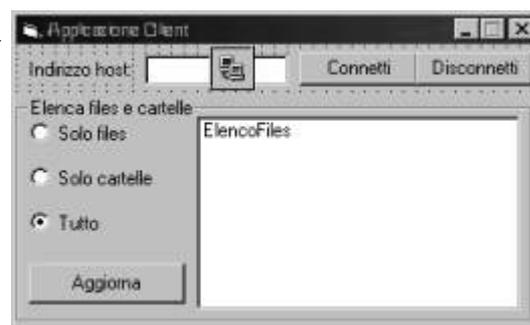
[<< Continua dalla parte 1](#)

In questa seconda parte vedremo come sviluppare il programma client che riceve i dati dal programma server visto nella prima parte.

Mentre la parte server conteneva molto codice e pochi elementi grafici, la parte server contiene più controlli sul form e poche righe di codice.

Le uniche funzioni che il client deve effettuare sono la connessione al computer remoto Server alla porta 1500, la richiesta della struttura directory e la sua visualizzazione.

Disponiamo sul nostro form **CientForm** una label la cui *Caption* sia "Indirizzo host" e alla destra di questa inseriamo una TextBox di nome **HostName**. Inseriamo anche due pulsanti standard di nome e *Caption* **Connetti** e **Disconnetti**.



Inseriamo ovviamente un controllo Winsock di nome **Client**. A causa di uno strano [bug](#) del Winsock è necessario creare una matrice di controlli per evitare errori. Per cui impostiamo anche la proprietà *Index* di questo controllo a 0.

Per dare un tocco grafico abbiamo inserito anche un frame la cui *Caption* è "**Elenca files e cartelle**".

Inseriamo all'interno del frame una matrice di OptionButton formata da 3 elementi. Denominiamo la matrice **SceltaDIR** ed associamo ai singoli pulsanti le proprietà *Index* a 0 a 1 e a 2. Il primo pulsante di opzione conterrà il testo "**Solo files**", il secondo conterrà "**Solo cartelle**", mentre il terzo avrà come *Caption* "**Tutto**".

Impostiamo anche la proprietà *Value* del terzo pulsante a True.

Sotto questi inseriamo un pulsante di comando di nome **AggiornaButton** la cui *Caption* sarà "**Aggiorna**". Nel momento in cui l'utente premerà questo verrà tentata la scansione del disco remoto in base ai pulsanti di opzione sopra.

Per ultimo elemento inseriamo una ListBox di nome **ElencoFiles**. Per semplificare un po' il tutto impostiamo la proprietà *Sorted* di essa a True. Così facendo, tutti gli elementi che verranno inseriti dentro essa saranno ordinati alfabeticamente.

La parte estetica del form è terminata.

Il funzionamento è molto semplice: l'utente immette l'indirizzo del computer a cui connettersi nella TextBox in alto, preme il pulsante Connetti e viene connesso al computer remoto. Per ottenere la struttura del disco remoto basterà scegliere la modalità di elencazione attraverso i pulsanti di opzione e premere il pulsante Aggiorna. Gli elementi verranno inseriti nella ListBox a fianco.

Prima di passare al codice è bene definire meglio il bug del controllo Winsock accennato prima. Dopo che ci si è collegati ad un indirizzo, ci si scollega e si ritenta la connessione, il controllo Winsock riporta un errore di runtime "*Indirizzo già in uso*" sebbene la connessione precedente fosse stata chiusa attraverso il metodo Close, via Client e via Server.

La soluzione più praticata consiste nel creare una matrice di controlli in [fase di progettazione](#) con un solo elemento. A [runtime](#), si alloca un secondo elemento di questa matrice; ad ogni tentativo di collegamento si dealloca l'elemento precedentemente allocato e lo si rialloca nuovamente. La deallocazione fa sì che il controllo Winsock *dimentichi* la precedente connessione, permettendo il normale collegamento.

Per una spiegazione su questo bug, consultare la sezione dedicata all'[utilizzo di NETSTAT](#).

Passiamo ora al codice:

```
1. Option Explicit
2. Private ELENCADIR As Boolean
```

Dichiariamo una variabile di nome **ELENCADIR**. Essa servirà da segnale nel momento in cui riceveremo l'elenco dal server.

```
4. Private Sub Form_Load()
5.     Load Client(1)
6. End Sub
```

All'avvio del programma allochiamo una nuova istanza del Winsock per risolvere il problema dell'indirizzo in uso.

```
8. Private Sub Connetti_Click()
9.     If Trim(HostName.Text) = "" Then Exit Sub
10.    Unload Client(1)
11.    Load Client(1)
12.    Client(1).Connect Trim(HostName.Text), 1500
13. End Sub
```

Quando l'utente preme il pulsante **Connetti**, innanzitutto viene controllato se il campo HostName contiene qualcosa. Se è vuoto non tenta nemmeno la connessione.

Supponiamo che il campo HostName contenga un indirizzo valido.

Prima di effettuare la connessione viene deallocato e riallocato l'[istanza](#) di Winsock per il bug già accennato prima.

Fatto questo si tenta la connessione all'host indicato dalla casella di testo, con porta 1500.

```
15. Private Sub Disconnetti_Click()
16.     Client(1).Close
17. End Sub
```

Nel momento in cui l'utente preme il pulsante **Disconnetti** viene chiusa la connessione via

client.

```
19. Private Sub AggiornaButton_Click()  
20.     If Client(1).State <> sckConnected Then Exit Sub  
21.     ELENCADIR = True  
22.     If SceltaDIR(0).Value Then Client(1).SendData "DIRF" & vbNewLine  
23.     If SceltaDIR(1).Value Then Client(1).SendData "DIRS" & vbNewLine  
24.     If SceltaDIR(2).Value Then Client(1).SendData "DIR" & vbNewLine  
25. End Sub
```

Questo è l'evento *Click* del pulsante **AggiornaButton**. Quando l'evento scatta, viene controllato se la connessione è stabilita, mediante la proprietà *State* del controllo Client. Se la connessione non è avvenuta, esce immediatamente dalla funzione senza far nulla.

Se la connessione è stabilita, imposta il flag **ELENCADIR**, indicando che i prossimi dati che arriveranno al controllo Client, saranno i dati della struttura della cartella remota.

Dopodichè segue il controllo del pulsante di opzione premuto.

Sul server sono presenti tre comandi di scansione: DIRF che recupera soltanto i files, DIRS che recupera soltanto le sottocartelle e DIR che recupera tutto dalla cartella di lavoro.

A seconda del pulsante di opzione scelto viene eseguito il comando corrispondente.

```
27. Private Sub Client_DataArrival(Index As Integer, ByVal bytesTotal As Long)  
28.     Dim DATI As String  
29.     Client(1).GetData DATI, vbString, bytesTotal  
30.     If ELENCADIR = True Then PreparaElenco DATI  
31. End Sub
```

L'evento *DataArrival* scatta nel momento in cui vengono inviati dati dal server verso il client.

Alla riga 28 viene dichiarata una variabile temporanea che riceverà i dati dal server tramite la riga successiva.

La riga 30 controlla se lo stato di **ELENCADIR** è True (ricordiamo che ELENCADIR viene impostato a True nel momento in cui viene premuto il pulsante Aggiorna, indicando che i dati che arriveranno saranno la struttura della cartella). In caso di controllo positivo esegue la funzione PreparaElenco passandogli come parametro i dati ricevuti dal server.

```
33. Private Sub ElencoFiles_DblClick()  
34.     Client(1).SendData "CD " & ElencoFiles.Text & vbNewLine  
35. End Sub
```

Nel momento in cui l'utente clicca due volte sull'elenco dei files viene inviato al server il comando CD che indica il cambio della cartella attiva.

```
37. Private Sub PreparaElenco(ByVal Elenco As String)  
38.     Dim INVIO As Integer  
39.     ELENCADIR = False  
40.     ElencoFiles.Clear  
41.     Elenco = Mid(Elenco, InStr(1, Elenco, vbNewLine))  
42.     INVIO = InStr(1, Elenco, vbNewLine)  
43.     If InStr(INVIO + 1, Elenco, vbNewLine) < 1 Then INVIO = 0  
44.     While INVIO <> 0  
45.         ElencoFiles.AddItem Mid(Elenco, INVIO + 2, InStr(INVIO + 1, Elenco,  
vbNewLine) - INVIO - 2)  
46.         INVIO = InStr(INVIO + 1, Elenco, vbNewLine)  
47.         If InStr(INVIO + 1, Elenco, vbNewLine) < 1 Then INVIO = 0
```

```
48.      Wend
49. End Sub
```

L'ultima funzione di questo programma è la `PreparaElenco` che legge la stringa ricevuta dal server ed inserisce gli elementi all'interno dell'elenco dei files.

Per prevenzione azzera subito il flag `ELENCADIR`. Azzera anche la `ListBox` **ElencoFiles**.

Prima di iniziare a lavorare elimina l'intestazione del pacchetto, ovvero la "+OK:", andando a cercarsi il primo `INVIO` (`vbNewLine`) dentro esso (riga 41).

Alla riga successiva memorizza la posizione del prossimo `INVIO` dentro la variabile **INVIO**.

La riga 43 è un controllo d'obbligo: se ci troviamo davanti ad un elenco vuoto, per esempio un'elenco di files di una cartella vuota. Per capire questo verifica se esiste un altro `INVIO` dopo quello indicato dalla variabile **INVIO**. Se non riesce a trovarlo, imposta quest'ultima a 0, forzando l'uscita del ciclo alla prossima riga.

La riga 44 inizia un ciclo che dura fintanto che la variabile **INVIO** conterrà un numero diverso da 0, ovvero fintanto che ci saranno caratteri di `INVIO` nell'elenco, quindi fintanto che ci saranno altri files o cartelle.

Questo ciclo contiene 3 istruzioni:

la prima estrae un elemento dall'elenco e lo inserisce all'interno di **ElencoFiles**;

la seconda aggiorna la posizione di **INVIO**, andandosi a cercare l'`INVIO` successivo all'elemento appena estratto;

la riga 47 effettua un'ultimo controllo: se non vi sono altri segni di `INVIO`, imposta la variabile **INVIO** a 0, per forzare l'uscita dal ciclo.

Il programma Client termina qui.

Quando lo si prova su connessioni Dial-Up con modem o su reti particolarmente lente, si raccomanda di non aver troppa fretta a premere un pulsante di seguito all'altro.

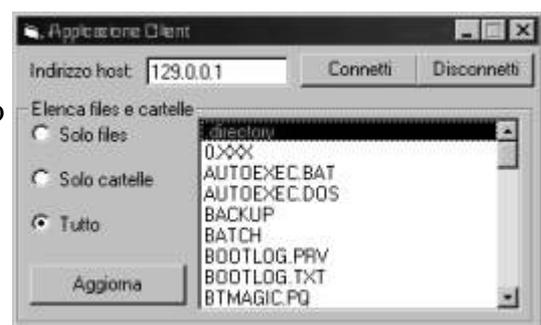
Infatti il programma non effettua nessun controllo se i dati sono arrivati correttamente al server.

È lo stesso motivo per il quale abbiamo evitato l'auto-aggiornamento della lista quando si effettua il cambio di cartella tramite doppio click.

Il codice è tutto da ottimizzare e da controllare, ma come esempio di base dovrebbe andare. È stato provato su una LAN a 10 MBit/s e non ha presentato nessuna difficoltà.

Lo sviluppo di architetture [Client/Server](#) presenta sempre numerosi rischi, e tra i principali spicca la perdita di un pacchetto durante il tragitto.

Il server per ogni richiesta ricevuta manda indietro una piccola intestazione indicante la correttezza dell'ultimo comando. Si consiglia di sfruttare le intestazioni per controllare il corretto scambio dei dati tra client e server.





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