

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

## Tutorial per l'utilizzo del Winsock con Visual Basic

[http://www.vbsimple.net/cliserv/clser\\_02.htm](http://www.vbsimple.net/cliserv/clser_02.htm)

Difficoltà:  2 / 5

La parte più affascinante della programmazione in quasi tutti i linguaggi è (secondo me) il Networking, cioè tutto quello che riguarda le reti, locali o remote. La soddisfazione di realizzare per la prima volta un programmino client/server è tanta, e grazie alla facilità d'uso del VB e del controllo [OCX Winsock](#)  alla fine di questo breve tutorial saremo in grado di costruire sia il client che il server e aggiungere tutte le funzionalità che ci verranno in mente, ma questo è lasciato alla fantasia e genialità del programmatore.

Quello che ci serve è solamente una discreta conoscenza dell'ambiente di sviluppo e dei fondamenti del VB. Apriamo quindi l'[IDE](#) di Visual Basic e partiamo! Nella casella degli strumenti che di solito si trova a sinistra ci sono tutti i [controlli standard](#), ma non quello che serve a noi e che dovremo aggiungere. Cliccando col destro all'interno della casella degli strumenti e scegliendo "Componenti...", oppure premendo **CTRL+T** si apre la finestra che ci offre altri controlli. Andremo a scegliere da essi il *Microsoft Winsock Control*.

Aggiungiamolo subito al nostro form ancora vuoto. Adesso possiamo cominciare a capire cosa ci permette di fare questo controllo. La prima cosa che succede quando interagiamo con un server è la richiesta di una connessione e l'attesa della risposta con l'esito della nostra richiesta. I primi due [eventi](#)  che vediamo sono quindi "**Connect**" e "**DataArrival**" e si può intuire dal nome cosa fanno.

### CONNECT

Questo evento ci connette ad un host e una porta specificati. Aggiungiamo un *CommandButton*  al form e chiamiamolo **Connetti**. Rinominiamo il controllo Winsock1 in **wskClient** (o il nome che a voi piace di più o sembra più facile da digitare). Associamo quindi alla pressione del CommandButton il [metodo](#)  *wskClient.Connect*.

Grazie all'autocompletamento ([Intellisense](#)) ci vengono chiesti due parametri: l'host e la porta. Per i nostri scopi usiamo come host quello del nostro provider o di un altro server di posta che conosciamo e come porta la 25. Ci connettiamo cioè al servizio di posta del server che abbiamo scelto. Non avendo ancora implementato l'evento  **DataArrival**, non saremo in grado di vedere quello che il server ci risponde. Più avanti nel tutorial vedremo come controllare comunque lo stato del winsock per capire se è in errore, in stato di "connessione avvenuta" o di "connessione in corso".

Quello che abbiamo fatto, la connessione che abbiamo stabilito, è praticamente uguale a

quella che potremmo fare con *Telnet*. Quello che otteniamo è una routine simile a questa:

```
1. Private Sub Connetti_Click()
2.     WskClient.Connect host, porta
3. End Sub
```

## DATA ARRIVAL

Aggiungiamo subito l'evento  *DataArrival* per sentire quello che il server ha da dirci. Per poterlo visualizzare aggiungiamo un controllo TextBox  che chiameremo **txtReply**. All'interno della Sub *wskClient\_DataArrival* aggiungiamo il metodo  *GetData* che ci chiede come parametri una variabile nella quale mettere i dati che arrivano e che dovrà essere di tipo stringa, dichiarata all'inizio.

Ci vengono anche chiesti il tipo di dati e la lunghezza. Gli ultimi due parametri, visto che non abbiamo particolari esigenze, li lasciamo vuoti. Avremo quindi una Sub simile a questa:

```
4. Private Sub wskClient_DataArrival(ByVal bytesTotal As Long)
5.     wskClient.GetData a
6.     txtReply = a & vbCrLf
7. End Sub
```

Con queste due semplici routine siamo già in grado di connetterci ad un server di posta e di ricevere il suo messaggio di benvenuto. Direi che è già un ottimo risultato.

Per sfizio si può provare a cambiare porta per vedere le stringhe che ci arrivano quando ci connettiamo ad altri servizi, come Ftp, Telnet, Finger e anche la risposta dell'Http Server sulla porta 80. Con in mano queste pochissime nozioni e usando un po di fantasia e di creatività, si possono costruire degli scanner per trovare server di posta che montano SmdMail, Ftp Server che vogliamo, demoni che ascoltano sulla porta 51 per il servizio di DNS o sulla porta 79 per il servizio di Finger.

Basta [parserizzare](#) la stringa che ci arriva e cercare nella risposta quello che ci interessa. Fatto questo si dà in pasto al programmino un range di IP tra i quali cercare. Per uno scanner di questo tipo che sia anche minimamente performante, occorre conoscere però gli Array di socket che studieremo un'altra volta.

Avete provato a connettervi a qualche server di posta ma non avete avuto successo? Prima di provare col nostro programmino, proviamo col Telnet, per verificare se il server scelto ci consente di connetterci.

La stringa 'a' che abbiamo inserito nel metodo **GetData** deve essere dichiarata tale all'inizio del listato della funzione.

Era una cosa che avevo omesso di dire, ma dovrebbe essere buona abitudine dichiarare qualsiasi tipo di variabile si usi, anche per chiarezza mentale o per chiarezza per chi legge il codice. Quindi:

```
4. Private Sub wskClient_DataArrival(ByVal bytesTotal As Long)
5.     Dim a As String
6.     wskClient.GetData a
7.     txtReply = a & vbCrLf
```

```
8. End Sub
```

Adesso però facciamo qualcosa di più utile, cioè inviamo qualcosa al server con quale siamo collegati. Senza neanche spremerci troppo potremmo immaginare che il metodo che ci permetterà di inviare una stringa sarà qualcosa tipo *wskClient.Send(data as String)*.

Ci siete andati vicini. Il metodo è **SendData** e richiede un parametro che sarà proprio la stringa che vogliamo inviare. Aggiungiamo un'altra TextBox  e un altro pulsante che chiameremo rispettivamente **txtOut** e **Invia**. Alla pressione del Button associamo il metodo creando così una routine del tipo:

```
9. Private Sub Invia_Click()  
10.     Dim data As String  
11.     data = txtOut  
12.     wskClient.SendData data & vbCrLf  
13. End Sub
```

Il **vbCrLf** è una [costante](#)  equivalente ad 'Invio'. Per chi ancora non lo sapesse significa **'Visual Basic Carriage Return Line Feed'**. Praticamente ritorno a capo con inizio di linea nuova. Nelle versioni precedenti di VB occorreva inviare i caratteri ASCII corrispondenti che sono il 10 e il 13.

L'evento *DataArrival*, una volta implementato correttamente da noi, lavora da solo e si attiva ogni volta che ci arrivano dei dati in risposta. Avrete già capito quindi che abbiamo costruito null'altro che un Telnet dal quale possiamo inviare stringhe e leggere le risposte del server. Abbiamo in mano quindi lo scheletro di un'applicazione Client e le cose che fa sono tre:

- Connettersi ad un Server;
- Ricevere i dati che gli arrivano da questi;
- Inviare stringhe

---

Sarebbe ora carino costruire un Server da far girare in locale e vedere le due applicazioni che dialogano secondo un protocollo di nostra invenzione. Pensiamo allora cosa deve fare un server. Per prima cosa deve stare in ascolto su una determinata porta e al momento di una richiesta accettarla o anche rifiutarla.

Dopo aver accettato la connessione il server non è più riconoscibile dal client a livello di metodi, perché dovrà inviare lui stesso delle stringhe e gestire quelle che gli arrivano con i metodi *SendData* e *GetData* in un Evento *DataArrival*. Di diverso dal client possiede solo un metodo che gli permette di ascoltare una determinata porta e uno che gli permette di accettare la connessione che gli viene richiesta o di rifiutarla (se ad esempio arrivasse da un host che vogliamo ignorare).

In poche parole svolge la funzione di demone e poi fa quello che abbiamo già visto fare al client. Apriamo un nuovo form e aggiungiamogli il controllo Winsock  che chiameremo **wskServer**. Vediamo allora questi due metodi nuovi.

## LISTEN

Per mettere il server in ascolto il metodo sarà - ci si sente quasi scemi a programmare in questo modo - **Listen**.

Quindi **wskServer.Listen** mette in ascolto il server.

Non manca nulla? Non gli abbiamo detto su che porta. Per specificarla utilizzeremo la proprietà  **LocalPort**. Visto che fino alla 1024 sono tutte riservate e non è buona cosa usarle, scegliamo la porta 2000.

Associamo la proprietà **LocalPort** e il metodo **Listen** ad un nuovo Button che chiameremo appunto **Listen**. Avremo quindi una routine del genere:

```
14. Private Sub Listen_Click()
15.     wskServer.LocalPort = 2000
16.     wskServer.Listen
17. End Sub
```

Se apriamo un Prompt del DOS  e scriviamo "[NETSTAT](#) -na" vediamo che il [localhost](#) (oppure l'host 127.0.0.1 o anche 0.0.0.0) ha la porta 2000 aperta in stato Listening.

## CONNECTION REQUEST

Ora dovremmo gestire la richiesta di connessione che presto faremo fare al nostro novello Client.

Per questo usiamo l'evento *ConnectionRequest* e gli associamo il metodo *Accept* che richiede come parametro il requestID che compare nella dichiarazione dell'evento.

Avremo quindi una routine del genere:

```
18. Private Sub wskServer_ConnectionRequest(ByVal requestID As Long)
19.     If (wskServer.State <> sckClosed) Then wskServer.Close
20.     wskServer.LocalPort = 0
21.     wskServer.Accept requestID
22. End Sub
```

Ho aggiunto un IF nel quale compaiono la proprietà *State* ed il metodo *Close*.

La ragione è semplice. Se la connessione fosse già attivata otterremmo un errore. In tal modo siamo sicuri di non richiedere una cosa che già è avvenuta (vedi le informazioni sullo [stato TIME\\_WAIT](#)).

Per avere anche la possibilità di rifiutare una connessione potremmo usare la proprietà *RemoteHostIP* per sapere se il client che sta richiedendo una connessione ci piace o meno.

Gli eventi *DataArrival* e i metodi *SendData* e *GetData* sappiamo già come vanno introdotti.

Abbiamo ottenuto quindi anche un abbozzo di server, anche se è un pò pretenzioso chiamarlo così. ;) Possiamo vedere i nostri due mostriciattoli in azione dicendo al client di collegarsi all'host 127.0.0.1 su una porta di vostra scelta (avevamo deciso la 2000).

Richiedere una connessione al server dopo che questo è stato messo in ascolto (sulla stessa porta ovviamente) e cominciare a scambiare stringhe di testo tra uno e l'altro. Sarebbe comodo qualche abbellimento come una TextBox nel quale si possono specificare host e porta e magari una label che ci dice che è stata richiesta una connessione o ancora meglio che ci dice in che stato è il controllo. Questo è gestibile con la proprietà *State* che abbiamo visto prima.

Sulla documentazione dell'MSDN si trovano tutti i valori che la proprietà *State* può assumere e gli stati al quale corrispondono. Quelli fondamentali sono:

- 0 = SocketClosed
- 1 = Open
- 2 = Listening
- 7 = Connected
- 9 = Error

Adesso libero sfogo alla fantasia! Possiamo costruirci un client di posta, una messenger simile ad Icq, un client per le news e tutto quello che vi viene in mente di malizioso. Se ad esempio mettessimo il form del server invisibile e mettessimo il winsock in Listen sul Form\_Load, magari scrivendo nel registro di partire in automatico all'avvio di Windows, cosa avremmo ottenuto...? ;)

Buon divertimento.

[Alberto Alagna](#)  
12 Marzo 2001



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

---