


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

Recuperare informazioni di rete tramite WSA

http://www.vbsimple.net/cliserv/clser_10.htm

Difficoltà:  3 / 5

In questo articolo vedremo come recuperare certe informazioni di rete sfruttando la classe *clsFBISocket* trattata [nell'articolo precedente](#). Più precisamente recuperemo il nome, i numeri IP sulla macchina macchina locale ed aggiungeremo alcune funzioni di conversione da IP a host e viceversa. Il progetto includerà la già citata classe, più una classe di nome **clsFBISocketInfo** che svilupperemo in questa sede e di cui vediamo subito la sezione dichiarazioni:

```

1. Option Explicit
2.
3. Private Const SOCKET_ERROR As Long = -1
4. Private Const AF_INET As Long = 2
5.
6. Private Type HOSTENT
7.     hName As Long
8.     hAliases As Long
9.     hAddrType As Integer
10.    hLen As Integer
11.    hAddrList As Long
12. End Type
13.

```

In queste poche righe si concentra tutta la complessità dell'intero articolo e solo una corretta comprensione della struttura *HOSTENT* limiterà gli insuccessi ed i probabili errori. Per utilizzarla è necessario avere chiara il concetto di [puntatore](#) ad una stringa; si tratta dell'indirizzo di memoria in cui si trova un determinato dato; [nel caso di stringhe il puntatore](#) indica l'indirizzo in cui si trova il primo carattere della stringa ([LPSTR](#)).

Il membro della struttura **hName** è un semplice puntatore al nome dell'host, cioè conterrà l'indirizzo in cui si trova la stringa del nome dell'host. Diverso è il caso dei membri **hAliases** (che non utilizzeremo in questo articolo) e **hAddrList**. Questi rappresentano un [array](#) di puntatori a stringa, cioè una sequenza di puntatori a stringa; ma poiché anche le matrici sono puntatori dove l'indirizzo di ogni elemento corrisponde a $\text{indirizzo} + (\text{indice} - 1) * 4$. Il numero 4 rappresenta 32 bit, cioè la lunghezza in bytes di un dato Long.

Data questa semplice operazione risulta chiaro che all'indirizzo corrisponde il primo elemento, all'indirizzo+4 il secondo, a +8 il terzo e così via. L'ultimo degli elementi sarà quello seguito da un elemento il cui puntatore è 0. Utilizzeremo il membro **hAddrList** che conterrà tutti gli indirizzi disponibili per l'host richiesto.

```

14. Private Declare Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" (pDst
    As Any, pSrc As Any, ByVal ByteLen As Long)
15. Private Declare Function lstrlenA Lib "KERNEL32" (ByVal Ptr As Any) As Long
16. Private Declare Function lstrcpy Lib "KERNEL32" Alias "lstrcpyA" (ByVal
    lpString As String, ByVal lPtr As Long) As Long
17. Private Declare Function gethostname Lib "WSOCK32" (ByVal szHost As String,

```

```

    ByVal dwHostLen As Long) As Long
18. Private Declare Function gethostbyname Lib "WSOCK32" (ByVal szHost As
    String) As Long
19. Private Declare Function gethostbyaddr Lib "WSOCK32" (haddr As Long, ByVal
    hhlen As Long, ByVal addrtype As Long) As Long
20.

```

Seguono le dichiarazioni delle funzioni esterne; *CopyMemory* già vista in tanti altri articoli è utilizzata per copiare una serie di bytes da un indirizzo ad un altro; *lstrlenA* e *lstrcpy* sono utilizzate per recuperare la lunghezza ed il contenuto di una stringa dato il suo puntatore; *gethostname* recupererà il nome dell'host locale, mentre *gethostbyname* e *gethostbyaddr* restituiscono un puntatore ad una struttura di tipo **HOSTENT**.

```

21. Private WithEvents fbiSocket As clsFBISocket
22. Private Host As HOSTENT
23. Private lpHost As Long
24.
25. Public Event Error(ByVal ErrorCode As Long, ByVal Description As String,
    ByVal Cancel As Boolean)
26.

```

La variabile **fbiSocket** rappresenterà l'istanza della classe *clsFBISocket*, che inizializza e mantiene attivo il sistema WSA per il processo; **HOST** è una variabile del tipo *HOSTENT* già visto in precedenza ed utilizzata da parecchie funzioni, mentre **lpHost** rappresenta il suo puntatore. Queste ultime due variabili sono utilizzate da quasi tutte le funzioni della classe.

È stato anche aggiunto un evento  **Error**, identico a quello della classe *clsFBISocket*; in tale maniera gli errori generati dall'oggetto **fbiSocket** saranno inviati direttamente all'evento **Error** di questa classe e basterà quindi gestire un solo evento per gli errori generati da entrambe le classi.

```

27. Private Sub Class_Initialize()
28.     Set fbiSocket = New clsFBISocket
29. End Sub
30.
31. Private Sub Class_Terminate()
32.     Set fbiSocket = Nothing
33. End Sub
34.
35. Private Sub fbiSocket_Error(ByVal ErrorCode As Long, ByVal Description As
    String, Cancel As Boolean)
36.     RaiseEvent Error(ErrorCode, Description, Cancel)
37. End Sub
38.
39. Public Property Get Socket() As clsFBISocket
40.     Set Socket = fbiSocket
41. End Property
42.

```

Naturalmente l'oggetto **fbiSocket** sarà creato contestualmente alla creazione dell'istanza della classe corrente e distrutto alla sua distruzione. Come già detto, tutti gli errori generati dall'oggetto **fbiSocket** saranno a loro volta inviati all'applicazione che utilizza questa classe. Sarà possibile accedere dall'esterno della classe al membro **fbiSocket** utilizzando la proprietà pubblica **Socket**.

Qui inizia il codice vero e proprio della classe, che è stato spezzettato in numerose tante funzioni per semplificarne l'uso e per evitare le ripetizioni di codice:

```

43. Public Function LongToIP(ByVal Host As Long) As String
44.     LongToIP = CStr(Host And &HFF) & "." & _
45.     CStr(Host \ &H100 And &HFF) & "." & _
46.     CStr(Host \ &H10000 And &HFF) & "." & _
47.     CStr(Host \ &H1000000 And &HFF)
48. End Function
49.

```

La prima funzione è **LongToIP** che converte un indirizzo IP in forma decimale nella forma puntata decimale (a.b.c.d); un *indirizzo decimale* sostanzialmente si compone di un valore a 32 bit che può essere rappresentato in maniera esadecimale con DDCCBBAA e può essere quindi scomposto nella maniera puntata con delle [semplici operazioni aritmetiche](#) estraendo di volta in volta un byte. Avremmo anche potuto convertire l'intero numero in esadecimale ed estrarre i singoli bytes utilizzando la funzione Mid ma sarebbe stato solo un'inutile spreco di tempi di elaborazione ed ottenendo gli stessi risultati.

```

50. Public Property Get LocalHost() As String
51.     LocalHost = Space$(256)
52.     If gethostname(LocalHost, Len(LocalHost)) = SOCKET_ERROR Then _
53.         Call fbiSocket.HandleError(11, "Impossibile recuperare l'host
        locale")
54.     LocalHost = Left$(LocalHost, InStr(1, LocalHost, vbNullChar) - 1)
55. End Property
56.

```

La proprietà **LocalHost** restituirà il nome dell'host locale che, nella quasi totalità dei casi, corrisponde al nome di rete del PC in uso. Il suo funzionamento è basato totalmente sulla funzione *gethostname* che effettua proprio questo scopo. Tutto il resto del codice è la preparazione del buffer sui cui la funzione lavorerà e l'estrazione del nome corretto al termine dell'elaborazione.

```

57. Public Function IPToHost(ByVal IP As String) As String
58.     lpHost = gethostbyaddr(HostToLong(IP, 0), 4, AF_INET)
59.     If lpHost <> 0 Then
60.         CopyMemory Host, ByVal lpHost, Len(Host)
61.         IPToHost = Space$(lstrlenA(ByVal Host.hName))
62.         Call lstrcpy(ByVal IPToHost, ByVal Host.hName)
63.     End If
64. End Function
65.

```

La funzione **IPToHost** recupera il nome dell'host corrispondente all'indirizzo IP specificato. La funzione, dopo aver convertito l'indirizzo IP punteggiato nel corrispondente decimale, richiama la funzione *gethostbyaddr* che richiede fra l'altro il tipo di indirizzo fornito, nel nostro caso **AF_INET** che indica un indirizzo Internet/Ethernet, e restituisce un puntatore ad una struttura di tipo HOSTENT.

Ottenuto il puntatore sarà possibile recuperare i dati dalla struttura e copiarli nella variabile **Host**; da quest'ultima estrarremo il nome dell'host, contenuto all'interno del campo hName ed indicato sotto forma di puntatore.

```

66. Public Property Get IPCount(ByVal HostName) As Long

```

```
67.     lpHost = gethostbyname(HostName)
68.     If lpHost <> 0 Then
69.         CopyMemory Host, ByVal lpHost, Len(Host)
70.         IPCount = -1
71.         Do
72.             IPCount = IPCount + 1
73.             CopyMemory lpHost, ByVal Host.hAddrList + 4 * IPCount, 4
74.             Loop Until lpHost = 0
75.         End If
76. End Property
77.
```

La proprietà **IPCount** restituisce il numero di indirizzi IP rilevati per l'host specificato; recuperate le informazioni sull'host mediante *gethostbyname* e, copiati i dati dal puntatore alla variabile **Host**, seguirà un controllo basato unicamente su un ciclo che verifica il valore di ogni elemento contenuto nell'array *hAddrList* (lo abbiamo già citato in cima all'articolo). Infatti l'ultimo elemento è quello seguito da un puntatore nullo.

```
78. Public Function HostToLong(ByVal HostName As String, ByVal Index As Long)
    As Long
79.     If Index >= IPCount(HostName) Then Exit Function
80.     lpHost = gethostbyname(HostName)
81.     If lpHost <> 0 Then
82.         CopyMemory Host, ByVal lpHost, Len(Host)
83.         CopyMemory lpHost, ByVal Host.hAddrList + Index * 4, 4
84.         If lpHost <> 0 Then CopyMemory lpHost, ByVal lpHost, 4
85.         HostToLong = lpHost
86.     End If
87. End Function
88.
```

Questa funzione recupera un indirizzo IP decimale dell'host indicato. L'argomento *Index* consente di specificare quale indirizzo IP restituire; naturalmente il valore di *Index* non può essere superiore al numero di IP rilevati dalla proprietà **IPCount**; inoltre poiché l'argomento *Index* è a base 0, ma il valore restituito da *IPCount* inizia da 1 è necessario che *Index* sia minore del valore restituito (riga 79).

Sarà quindi possibile recuperare il puntatore all'host tramite *gethostbyname*, ricopiare i dati all'interno di **Host** e quindi recuperare il puntatore all'indirizzo da noi richiesto semplicemente applicando il calcolo accennato all'inizio di questo articolo ($\text{indirizzo} + (\text{indice} - 1) * 4$). Recuperato l'indirizzo in cui si trova il puntatore del nostro indirizzo sarà possibile estrarre da questo il vero indirizzo. Lo ricordiamo, *hAddrList* conterrà un array di puntatori che indicano ciascuno la posizione in cui si trova l'indirizzo IP decimale.

```
89. Public Function HostToIP(ByVal HostName As String, ByVal Index As Integer)
    As String
90.     If Index >= IPCount(HostName) Then Exit Function
91.     HostToIP = LongToIP(HostToLong(HostName, Index))
92. End Function
```

L'ultima funzione della classe utilizza la funzione **HostToLong** e converte questo valore in forma punteggiata, utilizzando **LongToIP**. Quest'ultima funzione è stata aggiunta solo per comodità, sebbene avremmo potuto chiamare le due funzioni (*HostToLong* e *LongToIP*) direttamente.

Per testare il funzionamento di questa classe è stato preparato un semplicissimo form contenente alcuni controlli Label, una ListBox, due TextBox e quattro pulsanti. Tralasciamo i nomi dei controlli perché risulteranno subito chiari e passiamo al brevissimo codice:



```

1. Option Explicit
2.
3. Private socket As clsFBISocketInfo
4.
5. Private Sub Form_Load()
6.     Dim intLoop As Integer
7.     Dim strLocalHost As String
8.     Set socket = New clsFBISocketInfo
9.     With socket
10.        strLocalHost = .LocalHost
11.        lblServerLocaleVal.Caption = strLocalHost
12.        For intLoop = 0 To .IPCount(strLocalHost) - 1
13.            lbIPLocali.AddItem .HostToIP(strLocalHost, intLoop)
14.        Next intLoop
15.    End With
16. End Sub
17.
18. Private Sub Form_Unload(Cancel As Integer)
19.     Set socket = Nothing
20. End Sub
21.

```

All'avvio del form è creato l'oggetto **socket** di tipo *clsFBISocketInfo* visto in precedenza; alla riga 10 è recuperato il nome dell'host locale mediante proprietà **LocalHost** e quindi seguirà un ciclo per tutti gli indirizzi IP rilevati con **IPCount** e **HostIP**. Ogni indirizzo recuperato è inserito all'interno della ListBox *lbIPLocali*. Naturalmente alla chiusura del form sarà scaricato l'oggetto socket con tutto ciò che comporta, cioè la deallocazione della classe base *clsFBISocket* e quindi il rilascio di tutte le risorse occupate.

```

22. Private Sub lstIPLocali_Click()
23.     txtIndirizzo.Text = lbIPLocali.Text
24. End Sub
25.
26. Private Sub cmdHostToIP_Click()
27.     txtRisultato.Text = socket.HostToIP(txtIndirizzo.Text, 0)
28. End Sub
29.
30. Private Sub cmdHostToLong_Click()
31.     txtRisultato.Text = socket.HostToLong(txtIndirizzo.Text, 0)
32. End Sub
33.
34. Private Sub cmdIPToHost_Click()
35.     txtRisultato.Text = socket.IPToHost(txtIndirizzo.Text)
36. End Sub
37.
38. Private Sub cmdLongToIP_Click()
39.     txtRisultato.Text = socket.LongToIP(Val(txtIndirizzo.Text))
40. End Sub

```

Alla scelta di un indirizzo dall'elenco, questo sarà riportato all'interno della casella di testo `txtIndirizzo`; quindi premendo uno dei quattro pulsanti sarà richiamata la rispettiva funzione: **HostToIP** per recuperare il primo IP dell'host specificato in `txtIndirizzo`;

HostToLong per recuperare l'indirizzo IP decimale dell'host indicato; **IPToHost** per effettuare la ricerca inversa, dall'IP punteggiato al nome dell'host; infine **LongToIP** per recuperare l'IP punteggiato partendo da un IP decimale.

Il codice mostrato presenta una certa difficoltà d'approccio iniziale, ma una volta afferrati i concetti relativi ai puntatori il tutto si riduce a poche e semplici letture di questi valori. Questa classe sarà la base su cui verranno costruiti altri codici trattati in questa sezione.

[Fibia FBI](#)

27 Marzo 2004



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