Home Page
Novità
Aiuto

Introduzione alle funzioni WSA

http://www.vbsimple.net/cliserv/clser_09.htm

Difficoltà: 3 / 5

Questo articolo è il primo della sezione che utilizzerà le funzioni ***WSA*** (*Winsock API*) anziché il controllo Winsock visto negli altri articoli. Esistono numerose ragioni per preferire le funzioni dell'API al controllo ActiveX; il primo sicuramente è dato dalla libertà non dover installare l'OCX sulla macchina in cui andrà eseguito il programma, ma in generale il controllo presenta enormi limitazioni ed aspetti sgradevoli. La scelta fatta in questo articolo invece, è data esclusivamente dalla fattibilità; alcune operazioni che vedremo in seguito non sono possibili usando soltanto il controllo ActiveX.

Essendo il primo progetto basato su WSA sarà occasione per spiegare in generale il funzionamento, peraltro davvero molto semplice.

Affinché sia possibile utilizzare qualsiasi funzione WSA è necessario che venga chiamata almeno una volta all'interno del processo, la funzione di inizializzazione **WSAStartup**, che vedremo fra poco, e che consente di specificare il numero di versione richiesta delle funzioni WSA; se il sistema è in grado di fornire la versione richiesta risponderà positivamente oppure in caso contrario restituirà un codice d'errore ed il numero di versione che il sistema è in grado di utilizzare. Ad ogni chiamata di WSAStartup deve corrispondere una chiamata a **WSACleanUp** che servirà a liberare le risorse occupate. Una singola applicazione può chiamare più volte la funzione WSAStartup ma deve richiamare altrettante volte la funzione WSACleanUp al termine dell'utilizzo delle risorse.

Vista la necessità di dover chiamare ogni volta queste funzioni e ricordarsi di effettuare inizializzazione e scaricamento si è preferito sviluppare una classe apposita che servirà unicamente ad inizializzare il supporto alle funzioni WSA e liberare le risorse al termine. La classe sarà utilizzata in tutti gli esempi che accedono a WSA: il suo nome è **clsFBISocket** e contiene il seguente codice:

```
 1. Option Explicit
 2.
 3. Private Const WSADESCRIPTION_LEN = 256
 4. Private Const WSASYS_STATUS_LEN = 128
 5.
 6. Private Type WSADATA
 7.     bVersionL As Byte
 8.     bVersionH As Byte
 9.     bHighVersionL As Byte
10.     bHighVersionH As Byte
11.     szDescription(0 To WSADESCRIPTION_LEN) As Byte
12.     szSystemStatus(0 To WSASYS_STATUS_LEN) As Byte
13.     wMaxSockets As Integer
14.     wMaxUDPDG As Integer
15.     dwVendorInfo As Long
16. End Type
17.
18. Private Enum FBISocketErrors
```



```
19.     SocketSuccess = 0
20.     SocketErrNotResponding = 1
21.     SocketErrUnsupportedVer = 2
22.     SocketErrNotEnoughSockets = 3
23.     SocketErrOnCleanUp = 4
24. End Enum
25.
26. Private Declare Function WSAStartup Lib "WSOCK32.DLL" (ByVal
    wVersionRequired As Long, lpWSADATA As WSADATA) As Long
27. Private Declare Function WSACleanup Lib "WSOCK32.DLL" () As Long
28.
```

La sezione dichiarazioni conterrà la definizione di due costanti utilizzate dal tipo di dati WSADATA definito subito dopo; questo è richiesto dalla funzione di iniziailzzazione WSAStartup e conterrà i dati sulla versione di Windows Socket in uso (***bVersionL*** e ***bVersionH***), della versione massima supportata (***bHighVersionL*** e ***bHighVersionH***) e del numero massimo di socket disponibili (***wMaxSockets***). Segue un'enumerazione utilizzata unicamente all'interno della classe per l'assegnazione dei messaggi di errore ai singoli codici di errore.

Alle righe 26 e 27 sono definite le uniche due funzioni API necessarie: **WSAStartup** rappresenta l'inizializzazione ed utilizza due argomenti: il numero di versione richiesta ed una variabile di tipo WSADATA definita in precedenza. L'altra funzione molto più semplicemente libera le risorse occupate dall'inizializzazione e non richiede nessun argomento. Ogni inizializzazione richiede una deallocazione delle risorse ed è quindi necessario che il numero di inizializzazioni sia uguale al numero di deallocazioni. Questo controllo è effettuato più avanti.

```
29. Private WSAD As WSADATA
30. Private m_MinSockets As Integer
31. Private m_LastError As Long
32. Private m_IgnoreErrors As Boolean
33. Private m_WSAInitialized As Boolean
34.
35. Public Event Error(ByVal ErrorCode As Long, ByVal Description As String,
    ByRef Cancel As Boolean)
36.
```

La variabile **WSAD** è utilizzata unicamente dalla funzione vista in precedenza e sarà impiegata per recuperare altri dati in seguito all'inizializzazione del sistema Windows Socket.

Seguono altre tre proprietà private utilizzate unicamente per contenere i dati che verranno esposti dalle rispettive proprietà pubbliche. In particolare il membro ***m_WSAInitialized*** conterrà un valore che indica il buon esito o meno dell'operazione di iniziailzzazione e quindi della necessaria liberazione delle risorse. La classe espone anche un evento di nome **Error** che consente di intercettare gli errori WSA, con relativa descrizione e con la possibilità di non visualizzare l'avviso di errore.

```
37. Private Sub Class_Initialize()
38.     m_MinSockets = 1
39.     m_IgnoreErrors = False
40.     Call RequestVersion(1, 1)
41. End Sub
42.
43. Private Sub Class_Terminate()
```



```
44.     If m_WSAInitialized Then Call CleanUp
45. End Sub
46.
```

All'istanza della classe saranno inizializzati i membri delle proprietà e sarà richiesta l'inizializzazione del sistema WSA con la versione 1.1; in seguito sarà possibile richiedere una nuova versione, sempre richiamando la funzione **RequestVersion** che vedremo tra poco. Alla deallocazione dell'oggetto sarà invece chiamata la funzione di scaricamento delle risorse se il membro ***m_WSAInitialized*** attivato dalla funzione **RequestVersion** dovesse contenere il valore True/Vero.

```
47. Public Function ErrorDescription(ByVal ErrorCode As Long) As String
48.     Select Case ErrorCode
49.         Case SocketSuccess: ErrorDescription = "Nessun errore"
50.         Case SocketErrNotResponding: ErrorDescription = "L'ambiente Windows
    Socket 32 non risponde correttamente"
51.         Case SocketErrUnsupportedVer: ErrorDescription = "Versione di
    Windows Socket non supportata"
52.         Case SocketErrNotEnoughSockets: ErrorDescription = "Numero di
    socket supportati insufficiente"
53.         Case SocketErrOnCleanUp: ErrorDescription = "Errore durante il
    CleanUp"
54.     End Select
55. End Function
56.
```

Questa semplicissima funzione di nome **ErrorDescription** restituisce una descrizione di errore corrispondente al codice di errore fornito.

```
57. Public Function HandleError(ByVal ErrorCode As Long, Optional Description
    As String) As Boolean
58.     Dim blnCancel As Boolean
59.     Dim strError As String
60.     m_LastError = ErrorCode
61.     If Not m_IgnoreErrors Then
62.         strError = IIf(Len(Description) = 0, ErrorDescription(ErrorCode),
    Description)
63.         RaiseEvent Error(ErrorCode, strError, blnCancel)
64.         If Not blnCancel Then MsgBox "Errore " & CStr(ErrorCode) &
    vbNewLine & _
65.             strError, vbCritical Or vbOKOnly
66.     End If
67. End Function
68.
```

La funzione **HandleError** accoglierà quella noiosa parte di gestione dei messaggi di errore. La prima operazione da farsi è quella di verificare il valore del membro della proprietà **IgnoreErrors**; nel caso che questo valore fosse Vero non sarà manifestato alcun errore. Se il valore non dovesse essere Vero, sarà prepararato un messaggio di errore, lanciato l'evento, verificato il valore di ritorno dall'evento ed eventualmente mostrato un avviso di errore. Questa funzione verrà richiamata ogni volta che si verifica un errore e con essa verrà assegnato il valore al membro ***m_LastError***.

La funzione è stata resa pubblica e come tipo di dato di ErrorCode è stato usato Long anziché *FBISocketErrors* per consentire l'utilizzo della funzione anche dall'esterno, ed utilizzare un'unica funzione di gestione degli errori. Il secondo argomento, Description, conterrà una descrizione dell'errore e se specificato avrà la precedenza sulla descrizione

recuperabile tramite funzione ErrorDescription.

```
69. Public Function RequestVersion(ByVal LowVer As Byte, ByVal HighVer As Byte)
    As Boolean
70.     Dim lngVersionReq As Long
71.     lngVersionReq = HighVer * &H100 + LowVer
72.     If m_WSAInitialized Then CleanUp
73.     If WSAStartup(lngVersionReq, WSAD) <> 0 Then
74.         m_WSAInitialized = False
75.         Call HandleError(SocketErrNotResponding)
76.     Else
77.         m_WSAInitialized = True
78.         With WSAD
79.             If (.bVersionH < HighVer) Or _
80.                 (.bVersionH = HighVer And .bVersionL < LowVer) Then _
81.                 Call HandleError(SocketErrUnsupportedVer)
82.             If (.wMaxSockets < m_MinSockets) Then _
83.                 Call HandleError(SocketErrNotEnoughSockets)
84.         End With
85.     End If
86.     RequestVersion = m_WSAInitialized
87. End Function
88.
```

Il metodo di inizializzazione vero e proprio **RequestVersion**, richiamato anche durante l'istanza della classe, richiede la specifica della versione di Windows Socket da richiedere; sarà inizialmente verificato il valore del membro ***m_WSAInitialized***, in modo da poter effettuare la liberazione delle risorse prima di richiedere una nuova inizializzazione; ciò ci assicura che il numero di iniziailzzazioni in corso sia sempre 0 oppure 1. Il metodo **CleanUp** che vedremo subito dopo si occupa di liberare le risorse.

Richiamata la funzione *WSAStartup* (riga 73) effettueremo un primo controllo sul suo valore di ritorno; se il valore non è zero la funzione ha prodotto un errore e sarà quindi lasciata gestione alla routine HandleError. In tal caso m_WSAInitialized assumerà valore False ad indicare che non sarà necessario liberare le risorse, non essendo infatti riusciti ad inizializzare il supporto WSA.

Viceversa, se il valore di ritorno della funzione dovesse essere 0, assegneremo il valore True al membro m_WSAInitialized e procederemo con qualche controllo extra; un primo controllo riguarda la versione disponibile nel sistema; nel caso richiedessimo una versione non disponibile ed il sistema supporta invece una versione precedente la funzione non produrrà un errore ma riporterà nei valori bVersionH e bVersionL i numeri di versione in uso. Nel caso in cui la versione recuperata sia inferiore a quella richiesta sarà richiesta la generazione dell'errore *SocketErrUnsupportedVer* (righe 79-81).

Un altro controllo riguarda il numero di socket disponibili; mediante la proprietà MinSockets della classe è possibile assicurarsi durante l'inizializzazione che vi sia un numero di socket determinato. Il controllo alle righe 82 e 83 verificherà questo dato; se il numero di socket disponibili sia inferiore al numero contenuto nel membro ***m_MinSockets*** sarà generato l'errore *SocketErrNotEnoughSockets*.

Il valore di ritorno di questa funzione indicherà l'avvenuta inizializzazione o meno del supporto Windows Socket.


```
 89. Private Function CleanUp() As Boolean
 90.     If m_WSAInitialized Then CleanUp = (WSACleanup = 0)
 91.     If Not CleanUp Then Call HandleError(SocketErrOnCleanUp)
 92. End Function
 93.
```

Il metodo privato **CleanUp** si occuperà di liberare le risorse occupate, verificando inizialmente il valore di m_WSAInitialized, in modo da non richiamare la funzione di deallocazione quando non sia necessario farlo. La funzione *WSACleanUp* restituisce 0 nel caso che l'operazione sia andata a buon fine; se dovesse invece verificarsi un errore durante la deallocazione sarà generato l'errore *SocketErrOnCleanUp*.

Seguono le proprietà pubbliche sulla cui utiliità non dovrebbero sorgere dubbi:

```
 94. Public Property Get Version() As String
 95.     Version = CStr(WSAD.bVersionH) & "." & CStr(WSAD.bVersionL)
 96. End Property
 97.
 98. Public Property Get MaxVersion() As String
 99.     MaxVersion = CStr(WSAD.bHighVersionH) & "." & CStr(WSAD.bHighVersionL)
100. End Property
101.
102. Public Property Get MaxSocketSupported() As Integer
103.     MaxSocketSupported = WSAD.wMaxSockets
104. End Property
105.
```

La proprietà **Version** restituirà la versione di Windows Socket in uso che solitamente corrisponde a quella richiesta ma in alcuni casi può essere una versione inferiore se la versione richiesta non dovesse esistere. La proprietà **MaxVersion** restituisce invece il numero massimo di versione supportata dal sistema. La proprietà **MaxSocketSupported** restituisce il numero massimo di socket utilizzabili nella sessione corrente.

```
106. Public Property Get MinSockets() As Integer
107.     MinSockets = m_MinSockets
108. End Property
109.
110. Public Property Let MinSockets(ByVal newValue As Integer)
111.     m_MinSockets = newValue
112. End Property
113.
114. Public Property Get LastError() As Long
115.     LastError = m_LastError
116. End Property
117.
118. Public Property Let LastError(ByVal newValue As Long)
119.     m_LastError = newValue
120. End Property
121.
122. Public Property Get IgnoreErrors() As Boolean
123.     IgnoreErrors = m_IgnoreErrors
124. End Property
125.
126. Public Property Let IgnoreErrors(ByVal newValue As Boolean)
127.     m_IgnoreErrors = newValue
128. End Property
```

La classe si conclude con tre semplici proprietà in lettura e scrittura **MinSockets**, **LastError** e **IgnoreErrors** che puntano ai rispettivi membri.

Il modulo di classe  qui analizzato, davvero molto semplice, si occuperà esclusivamente di inizializzare il supporto WSA ed effettuare quei controlli d'obbligo sull'esecuzione e sulla versione disponibile.

Fibia FBI
26 Marzo 2004

 Torna all'indice Client/Server

