Creazione di un gruppo di controlli Winsock in un modulo di classe

http://www.vbsimple.net/cliserv/clser_08.htm

Difficoltà: 4 / 5

L'articolo prende spunto da quello dedicato alla creazione di un singolo controllo Winsock in un modulo di classe ed implementa una soluzione basata sull'uso di due classi collegate: la prima servirà per allocare un singolo controllo e la seconda raggruppa i vari controlli e ne gestisce gli eventi.

All'interno della classe il gruppo di controlli può essere gestito mediante un array oppure mediante una Collection. L'uso della prima soluzione piuttosto che della seconda è determinato da una costante di compilazione condizionale. Vedi le Informazioni aggiuntive sulla compilazione condizionale.

Sebbene le due classi siano inscindibili ed interdipendenti l'una dall'altra cominceremo a vedere il codice della prima classe: ***clsFBIOnClassSocketSingle***:

```
 1. Option Explicit
 2.
 3. Private WithEvents m_Socket As Winsock
 4. Private m_Parent As clsFBIOnClassSocketGroup
 5. Private m_InGroupIndex As Integer
 6.
 7. Private Enum WinsockEvents
 8.     Winsock_Close = 0
 9.     Winsock_Connect = 1
10.     Winsock_ConnectionRequest = 2
11.     Winsock_DataArrival = 3
12.     Winsock_Error = 4
13.     Winsock_SendComplete = 5
14.     Winsock_SendProgress = 6
15. End Enum
16.
```

Alle righe 3-5 sono dichiarate le tre variabili membro utilizzate dalla classe: **m_Socket** è un controllo Windock con eventi creato mediante la terza soluzione del tutorial precedente. Tutti i suoi eventi saranno passati al gruppo ***clsFBIOnClassSocketGroup***, che lo contiene, identificato dalla variabile **m_Parent**. L'ultima variabile membro **m_InGroupIndex** conterrà l'indice di questa istanza all'interno del gruppo identificato da **m_Parent**.

L'enumerazione *WinsockEvents* (righe 7-15) verrà utilizzata per comunicare l'esecuzione di un evento dal controllo Winsock alla classe ***clsFBIOnClassSocketGroup***.

```
17. Private Sub Class_Initialize()
18.     Set m_Socket = New Winsock
19.     Set m_Parent = Nothing
20.     m_InGroupIndex = -1
21. End Sub
22.
```



```
23. Private Sub Class_Terminate()
24.     On Error Resume Next
25.     If m_Socket.State <> sckClosed Then m_Socket.Close
26.     Set m_Socket = Nothing
27.     Set m_Parent = Nothing
28. End Sub
29.
```

All'istanza del controllo sarà allocato il nuovo controllo Winsock (riga 18) ed inizializzato il valore **m_InGroupIndex** a -1. In maniera analoga, alla deallocazione dell'istanza sarà inizialmente chiusa l'eventuale connessione lasciata aperta (riga 25) e verranno deallocati i due oggetti **m_Socket** ed **m_Parent**.

```
30. Private Sub m_Socket_Close()
31.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex, Winsock_Close
32. End Sub
33.
34. Private Sub m_Socket_Connect()
35.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex,
    Winsock_Connect
36. End Sub
37.
38. Private Sub m_Socket_ConnectionRequest(ByVal requestID As Long)
39.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex,
    Winsock_ConnectionRequest, requestID
40. End Sub
41.
42. Private Sub m_Socket_DataArrival(ByVal bytesTotal As Long)
43.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex,
    Winsock_DataArrival, bytesTotal
44. End Sub
45.
46. Private Sub m_Socket_Error(ByVal Number As Integer, Description As String, ByVal
    Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext
    As Long, CancelDisplay As Boolean)
47.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex,
    Winsock_Error, Number, Description, Scode, Source, HelpFile, HelpContext,
    CancelDisplay
48. End Sub
49.
50. Private Sub m_Socket_SendComplete()
51.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex,
    Winsock_SendComplete
52. End Sub
53.
54. Private Sub m_Socket_SendProgress(ByVal bytesSent As Long, ByVal bytesRemaining As
    Long)
55.     If Not m_Parent Is Nothing Then m_Parent.NewEvent m_InGroupIndex,
    Winsock_SendProgress, bytesSent, bytesRemaining
56. End Sub
57.
```

Quelli sopra mostrati sono tutti gli eventi ricevuti dal controllo **m_Socket** e per ognuno di essi sarà richiamato il metodo **NewEvent** della classe ***clsFBIOnClassSocketGroup*** fornendogli come primo argomento l'indice dell'elemento all'interno del gruppo, indicato da **m_InGroupIndex**, come secondo argomento un valore dell'enumerazione *WinsockEvents* e come altri argomenti tutti i dati ricevuti dall'evento del controllo.

Questo farà sì che l'istanza **m_Parent** riceverà l'evento generato dal controllo e l'indice del controllo all'interno del gruppo, in modo da comunicarlo all'esterno della classe e permettere al programma la gestione degli eventi dei singoli controlli all'interno del gruppo.


```
58. Public Property Get Socket() As Winsock
59.     Set Socket = m_Socket
60. End Property
61.
```

La proprietà **Socket** restituisce il controllo Winsock mantenuto all'interno della classe, per permettere l'impostazione e l'uso dall'esterno.

```
62. Friend Sub AddToGroup(ByVal Group As clsFBIOnClassSocketGroup, ByVal Index As
    Integer)
63.     Set m_Parent = Group
64.     m_InGroupIndex = Index
65. End Sub
```

L'ultima funzione della classe è **AddToGroup** e consente di aggiungere l'istanza corrente ad un gruppo ***clsFBIOnClassSocketGroup***, che si occuperà di gestire gli eventi in maniera associata all'indice dell'istanza all'interno del gruppo.

La seconda classe ***clsFBIOnClassSocketGroup*** si occuperà di collegare più istanze della classe ***clsFBIOnClassSocketSingle*** in un unico gruppo. La sua reale funzione è quella di consentire la creazione di un nuovo controllo Winsock, restituire quelli già esistenti e gestire gli eventi provenienti dai singoli controlli Winsock. Vediamone subito il codice:

```
 1. Option Explicit
 2. Option Base 1
 3.
 4. #Const USEARRAY = 1
 5.
 6. #If USEARRAY = 0 Then
 7.     Private SocketsInterni As Collection
 8. #Else
 9.     Private SocketsInterni() As clsFBIOnClassSocketSingle
10. #End If
11.
12. Private intSocketsCount As Integer
13.
```

L'istruzione *Option Base 1* alla riga 2 determina che il limite inferiore di tutte le matrici sia 1, anziché 0; è stato richiesto appositamente per mantenere la compatibilità tra la Collection (sempre a base 1) e l'array di istanze ***clsFBIOnClassSocketSingle*** (solitamente a base 0).

La costante di compilazione condizionale ***USEARRAY*** determina infatti se il codice farà uso della matrice oppure dell'oggetto Collection. La dichiarazione della base dati **SocketsInterni** è effettuata alle righe 6-10. Naturalmente una delle dichiarazioni esclude automaticamente l'altra.

La variabile **intSocketsCount** alla riga 12 tiene conto del numero di controlli Winsock contenuti.

```
14. Private Enum WinsockEvents
15.     Winsock_Close = 0
16.     Winsock_Connect = 1
17.     Winsock_ConnectionRequest = 2
18.     Winsock_DataArrival = 3
19.     Winsock_Error = 4
20.     Winsock_SendComplete = 5
```



```
21.     Winsock_SendProgress = 6
22. End Enum
23.
24. Public Event WSClose(ByVal Index As Integer)
25. Public Event Connect(ByVal Index As Integer)
26. Public Event ConnectionRequest(ByVal Index As Integer, ByVal requestID As Long)
27. Public Event DataArrival(ByVal Index As Integer, ByVal bytesTotal As Long)
28. Public Event Error(ByVal Index As Integer, ByVal Number As Integer, ByVal
    Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile
    As String, ByVal HelpContext As Long, ByVal CancelDisplay As Boolean)
29. Public Event SendComplete(ByVal Index As Integer)
30. Public Event SendProgress(ByVal Index As Integer, ByVal bytesSent As Long, ByVal
    bytesRemaining As Long)
31.
```

L'enumerazione *WinsockEvents*, vista anche nella classe precedente, verrà utilizzata dalla classe per identificare con facilità quale evento è stato ricevuto. Tali eventi a loro volta saranno rilasciati sull'utilizzatore dell'istanza; gli eventi dichiarati alle righe 24-30 sono i medesimi di un normlae controllo Winsock ma vi aggiungono un primo argomento di nome ***Index***, che identifica il singolo controllo nel gruppo, come se si trattasse di una matrice di controlli posta sopra un form.

```
32. Private Sub Class_Initialize()
33.     intSocketsCount = 0
34.     #If USEARRAY = 0 Then
35.         Set SocketsInterni = New Collection
36.     #End If
37. End Sub
38.
39. Private Sub Class_Terminate()
40.     Dim intCount As Integer
41.     #If USEARRAY = 0 Then
42.         For intCount = SocketsInterni.Count To 1 Step -1
43.             SocketsInterni.Remove intCount
44.         Next intCount
45.         Set SocketsInterni = Nothing
46.     #Else
47.         For intCount = UBound(SocketsInterni) To LBound(SocketsInterni) Step -1
48.             Set SocketsInterni(intCount) = Nothing
49.         Next intCount
50.         Erase SocketsInterni
51.     #End If
52. End Sub
53.
```

All'istanza della classe il numero di elementi contenuti nel gruppo è 0 (riga 33). Se è stato scelto di usare un oggetto Collection piuttosto che un array, esso sarà istanziato alla riga 35.

Alla deallocazione dell'istanza sarà necessario fare un po' di pulizia: saranno deallocati uno per uno tutti gli oggetti del gruppo (righe 42-44 e 47-49); inoltre nel caso di un gruppo Collection sarà deallocata anche l'istanza **SocketsInterni** (riga 45); nel caso fosse un array esso sarà semplicemente azzerato (riga 50).

```
54. Public Property Get Count() As Integer
55.     Count = intSocketsCount
56. End Property
57.
58. Public Property Get IsArray() As Boolean
59.     If USEARRAY = 0 Then
60.         IsArray = False
61.     #Else
62.         IsArray = True
```



```
63.     #End If
64. End Property
65.
66. Public Property Get Item(ByVal Index As Integer) As Winsock
67.     Set Item = SocketsInterni(Index).Socket
68. End Property
69.
```

Le tre proprietà sopra dichiarate restituiscono rispettivamente il numero di controlli contenuti nel gruppo (righe 54-56), il tipo di gruppo utilizzato (righe 58-64) ed il controllo Winsock identificato dal'indice Index all'interno del gruppo.

```
70. #If USEARRAY = 0 Then
71. Public Property Get NewEnum() As IUnknown
72.     Set NewEnum = SocketsInterni.[_NewEnum]
73. End Property
74. #End If
75.
```

Se la classe sta facendo uso di un oggetto Collection sarà aggiunta una nuova proprietà (nascosta) di nome **NewEnum**. Essa consentirà l'iterazione con tutti gli elementi del gruppo in un ciclo For..Each.
Nel caso il gruppo utilizzasse un array tale proprietà semplicemente non sarà disponibile.

Importante!
Affinché la proprietà **NewEnum** funzioni nella maniera corretta, è necessario assegnarle ID Routine -4 mediante la finestra di dialogo Strumenti -> Attributi routine.
Inoltre a causa di un bug nell'IDE di Visual Basic, talvolta questo valore può essere rimosso automaticamente. In caso di errore nell'uso di un ciclo For..Each si raccomanda l'impostazione dell'ID Routine della proprietà e subito dopo l'impostazione chiudere la finestra di dialogo con OK, **senza visualizzare le altre proprietà**.

```
76. Public Function Add() As Winsock
77.     Dim newSocketSingle As clsFBIOnClassSocketSingle
78.     Set newSocketSingle = New clsFBIOnClassSocketSingle
79.     #If USEARRAY = 0 Then
80.         SocketsInterni.Add newSocketSingle
81.     #Else
82.         ReDim Preserve SocketsInterni(intSocketsCount + 1) As
    clsFBIOnClassSocketSingle
83.         Set SocketsInterni(intSocketsCount + 1) = newSocketSingle
84.     #End If
85.     intSocketsCount = intSocketsCount + 1
86.     newSocketSingle.AddToGroup Me, intSocketsCount
87.     Set Add = newSocketSingle.Socket
88.     Set newSocketSingle = Nothing
89. End Function
90.
```

Il metodo più complesso della classe è **Add**, che aggiunge un'istanza ***clsFBIOnClassSocketSingle*** al gruppo e restituisce in uscita un riferimento al controllo Winsock generato.

La variabile **newSocketSingle** si è dimostrata obbligatoria per superare uno sciocco limite degli oggetti Collection. Infatti sebbene ogni elemento della Collection sia un'istanza della classe ***clsFBIOnClassSocketSingle***, Visual Basic non consente l'uso implicito di un elemento della Collection per operazioni su oggetti, come ad esempio `SocketsInterni`


```
(i).AddToGroup.
```

Pertanto alla riga 78 è allocata una nuova istanza della prima classe ed alle righe 79-84 essa è aggiunta al gruppo SocketsInterni, nella forma consentita da una fra le due basi di dati. Alla riga 86 la nuova istanza è aggiunta al gruppo corrente mediante **AddToGroup**, per consentirle la comunicazione dei suoi eventi al gruppo di appartenenza.

In uscita dalla funzione saranno in ordine restituiti il nuovo controllo Winsock creato e deallocata la variabile temporanea **newSocketSingle**.

```
 91. Friend Sub NewEvent(ByVal Index As Integer, WSEvent As WinsockEvents, ParamArray
     Arguments() As Variant)
 92.     Select Case WSEvent
 93.         Case Winsock_Close
 94.             RaiseEvent WSClose(Index)
 95.         Case Winsock_Connect
 96.             RaiseEvent Connect(Index)
 97.         Case Winsock_ConnectionRequest
 98.             RaiseEvent ConnectionRequest(Index, Arguments(0))
 99.         Case Winsock_DataArrival
100.             RaiseEvent DataArrival(Index, Arguments(0))
101.         Case Winsock_Error
102.             RaiseEvent Error(Index, Arguments(0), Arguments(1), Arguments(2),
     Arguments(3), Arguments(4), Arguments(5), Arguments(6))
103.         Case Winsock_SendComplete
104.             RaiseEvent SendComplete(Index)
105.         Case Winsock_SendProgress
106.             RaiseEvent SendProgress(Index, Arguments(0), Arguments(1))
107.     End Select
108. End Sub
```

L'ultima funzione della classe è **NewEvent**, dichiarata come Friend perché dovrebbe essere utilizzata esclusivamente dalle classi ***clsFBIOnClassSocketSingle*** per la comunicazione degli eventi generati.

La funzione utilizza un numero indefinito di argomenti: i primi due argomenti identificano l'indice del controllo all'interno del gruppo e l'evento generato, come da enumerazione *WinsockConstants*. Il terzo argomento è un Paramarray, cioè in grado di ricevere un numero indefinito di altri argomenti, che saranno quindi forniti ai singoli eventi richiamati.

Il funzionamento di questo metodo è in realtà molto semplice: in base al valore dell'evento generato sarà richiamato l'evento di questa classe, che verrà fornito all'utilizzatore della stessa. Si tratta quindi di un semplice passaggio di dati dalla prima classe alla seconda e da questa verso il programma utilizzatore.

Per dimostrare il funzionamento delle due classi sarà sviulppato una microscopica chat punto-punto composta da un solo form e 5 caselle di testo.

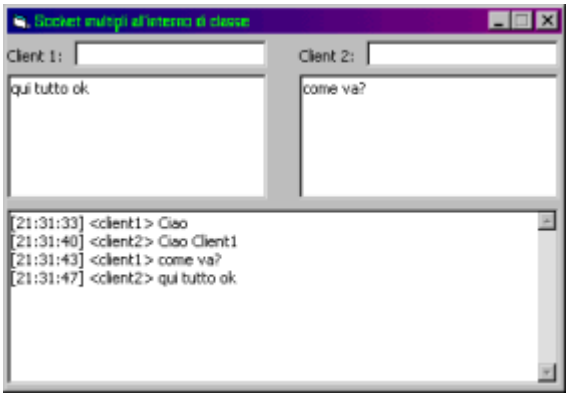

Figura 1

All'interno del form sarà allocata una nuova istanza **clsFBIOnClassSocketGroup** e dichiarata con *WithEvents* in modo da poterne ricevere gli eventi di ritorno:

```
 1. Option Explicit
 2.
 3. Private WithEvents Sockets As clsFBIOnClassSocketGroup
 4.
 5. Private Sub Form_Load()
 6.     Dim varSock As Variant
 7.     Set Sockets = New clsFBIOnClassSocketGroup
 8.     With Sockets
 9.         .Add .Item(1).LocalPort = 1000
10.         .Item(1).Listen
11.         .Add
12.         .Item(2).Connect "127.0.0.1", 1000
13.         If Not .IsArray Then
14.             For Each varSock In Sockets
15.                 Debug.Print varSock.Socket.LocalPort
16.             Next varSock
17.         End If
18.     End With
19. End Sub
```

Così al caricamento del form l'istanza **Sockets** è allocata e con essa (alle righe 9-12) sono creati due controlli Winsock nel gruppo dei quali uno è posto in ascolto sulla porta TCP 1000 (riga 10) e l'altro si connette al primo (riga 12).

Inoltre, se è stato deciso di utilizzare la Collection invece dell'array di controlli, sarà eseguito un ciclo iterativo con *For..Each* (righe 14-16) che mostrerà nella finestra immediata l'indirizzo delle porte locali dei due controlli. Naturalmente la sua funzionalità è puramente simbolica per dimostrare il funzionamento della proprietà **NewEnum** della classe, in un ciclo *For..Each*.

```
20. Private Sub Sockets_ConnectionRequest(ByVal Index As Integer, ByVal requestID As
    Long)
21.     With Sockets.Item(Index)
22.         .Close
23.         .Accept requestID
24.     End With
25. End Sub
26.
```

L'evento **ConnectionRequest** di uno dei due controlli si rispecchia automaticamente (mapping) sul medesimo evento dell'istanza **Sockets**. Tra gli argomenti il primo identifica il controllo Winsock nel gruppo che ha generato l'evento.

Nel caso specifico al tentativo di collegamento si risponderà con la chiusura dalla fase d'ascolto e con l'accoglimento della connessione, in questo caso, del primo controllo.

```
27. Private Sub Sockets_DataArrival(ByVal Index As Integer, ByVal bytesTotal As Long)
28.     Dim strBuffer As String
29.     Sockets.Item(Index).GetData strBuffer
30.     If Index = 1 Then
31.         txtCli1Log.Text = strBuffer
32.         txtFullLog.Text = txtFullLog & "[" & Time$ & "] <client2> " & strBuffer &
    vbNewLine
33.     Else
34.         txtCli2Log.Text = strBuffer
35.         txtFullLog.Text = txtFullLog & "[" & Time$ & "] <client1> " & strBuffer &
    vbNewLine
36.     End If
37. End Sub
```

Tralasciamo il resto del codice, più o meno inutile all'uso della classe e vediamo l'uso di un altro evento dell'istanza Sockets: **DataArrival**. Alla riga 29 sono letti i dati diretti **al controllo** interessato ed alla riga successiva è verificata la destinazione dei dati (argomento *Index*).

Se i dati sono diretti al primo controllo (riga 30) dovranno essere quindi essere inviati alla casella dei messaggi del primo controllo **txtCli1Log**; viceversa i dati saranno inviati alla casella **txtCli2Log**.
In entrambi i casi una copia dei dati è inviata alla grande casella di testo **txtFullLog** con tutti i messaggi ricevuti, differenziandone però l'origine (righe 32 e 35).

Non è stato possibile affrontare l'argomento della microchat completamente perché avrebbe richiesto troppo spazio. La sua funzionalità tuttavia è davvero molto semplice e si è preferito dedicare maggiore spazio alle due classi ***clsFBIOnClassSocketSingle*** e ***clsFBIOnClassSocketGroup*** che formano il gruppo di controlli Winsock.

Il loro funzionamento è davvero molto semplice e per molti versi simile al comportamento di una matrice di controlli Winsock posti sopra la superficie di un form.
Per ragioni di efficienza si è preferito non sviluppare una funzione per l'eliminazione di un controllo dal gruppo.

Fibia FBI
13 Novembre 2002


Torna all'indice Client / Server

