EASYNET 1.61

EasyNet is a Custom Control for Microsoft Visual Basic for Windows (*).
It helps you quickly draw and manage flowchart or network diagrams.

(*) Microsoft is a registered trademark. Windows and Visual Basic are trademarks of Microsoft Corporation.

Quick Start

- **Add the EasyNet VBX** to your project by selecting "Add File..." from Visual Basic's "File" menu. If you have not a license file, an "About" dialog box appears and you have to click Ok.

- **Drag an EasyNet control** from the toolbox to your form.

- **Launch** the program by selecting "Start" from the "Run" menu (or do F5).

- **Draw a node**: bring the mouse cursor into the EasyNet control, press the left button, move the mouse and release the left button. You have created an elliptic node. This node is selected: that's why 9 handles (little squares) are displayed. The handle at the center of the node is used to draw a link. The 8 others allow to **resize the node**. If you want to **move the node** you bring the mouse cursor into the node, press the left button, move the mouse and release the left button.

- **Draw a second node**...(same method)

- **Draw a link**: bring the mouse cursor into the handle at the center of the selected node, press the left button, move the mouse towards the other node. When the mouse cursor is into the other node, release the left button. The link has been created. And it is selected since a handle is displayed at the center of this link.

- **You may stretch this link**: bring the mouse cursor into the link handle, press the left button, move the mouse and release the left button. You have created a new link segment. It has 3 handles allowing you to add or remove segments. (The handle at the intersection of two segments allows you to remove a segment :   you move it with the mouse so that the two segments are aligned and when these two segments are approximately aligned, release the left button).

- **Now, you may return to the Visual Basic design-time mode** in order to change EasyNet control properties. For instance you may change the node filling color with FillColor property, the node shape (Shape property), the drawing color (DrawColor property). You may allow **multiselection** (MultiSel and SelectMode properties), add scrollbars (ScrollBars property), etc...

You may also create items programmaticaly with EditAction property. Or copy the diagram to clipboard as a metafile, save its image to a file as a metafile.

...Well, it is very easy, isnt'it?

Why EasyNet?

If you need flowcharting features
If you want to implement a workflow applications
If you wish to draw organizational charts
If you have to draw communication networks
If you plan to draw state transitions diagrams
If you need to display relationships between entities (database diagrams)

then EasyNet is indispensable. GET IT!!

It allows you to draw diagrams interactively or programmaticaly in minutes.

EasyNet is powerful, opened and customizable:

- *allows to associate your own data to each item (node or link).*
- *allows navigation.*
- *offers many properties allowing you to "customize" your diagramming application.*
- *is a VBX 1.0 level control. Therefore, it can be used in other host environments.*
- *includes Royalty free runtime distribution*
- *only $119 !!*

Overview

This Custom Control allows to draw network diagrams. A network diagram is a set of nodes that can be linked. So an EasyNet control contains items that can be nodes or links. You can associate data to each item and you can navigate in the network diagram.

Drawings can be made interactively with the mouse or programmaticaly. See Quick Start to see how to interactively draw nodes, resize nodes, move nodes, stretch links, select one item or multiselect items.

By exploring following topics, you'll discover all features of the EasyNet control.

- Items
- Drawing
- User Data Association
- Navigation
- Capabilities
- Saving/Loading
- Performance tuning
- Limits

Items

Items are nodes or links. Two nodes can be linked with a link. A link cannot exist without its origin and destination nodes. If one of these two nodes is deleted, the link is also deleted.

You can make an item be the current one either with the mouse or with Item property, allowing you to work with it, get or set its properties. You can also select several items with the mouse if multiselection is allowed (in such a case MultiSel and SelectMode properties are true).

IsLink property allows to know if current item is a link or not.

Sleeping property allows to specify if an item is active or not. If it sleeps, the user cannot interactively make it current or selected.

You can create items, delete items and do other edit actions (like copying the network diagram onto the clipboard in a metafile format) with EditAction property.

Example:*If current item is a link, make its origin node be red.*

```
Dim curLink As Long

If Net1.IsLink = True Then
  ' Save current item
  curLink = Net1.Item

  ' Make origin node be the current item
  ' in order to work with it
  Net1.Item = Net1.Org

  ' Change node filling color
  Net1.FillColor = RGB(255, 0, 0)

  ' Restore current item
  Net1.Item = curLink
End If
```

Drawing

You can change colors, styles and shapes of each item:

- X1, X2, Y1, Y2 properties allows to set or get position
  and size of each item.
- Picture property allows to associate a picture to each node.
- Shape property allows to specify a shape for a node.
- DrawColor, DrawStyle and DrawWidth
  properties allow to specify the color and width of the pen used
  to draw nodes or links.
- FillColor property allows to specify the color used inside a node.
- Text property associates a string that is displayed inside the node at a position
  depending on Alignment property (if item is a node) or near the link
  (if item is a link).
  The EasyNet control maintains the memory for the strings associated to items.
- Alignment sets or returns the alignment of text in a node.
- PointCount, PointX, PointY properties allow
  to have a link composed of several segments.
- Oriented property specifies if a link is oriented or not.
  If the link is oriented, it has an arrowhead.
- LinkHead property the arrowhead shape for a link.
- Transparent property specifies if a node is transparent or not.
- Hiding property specifies if an item (node or link) is visible or not.
- You can create items, delete items and do other edit actions (like copying
  the network diagram onto the clipboard in a metafile format) with
  EditAction property.

Example:

Creates 3 nodes and 2 links. Each node has a text. Two are rectangles and the other is an ellipse. The links are oriented.

```
Sub Exercice ()
 Dim n1, n2, n3 As Long

 ' Cause current item to be null
 ' Therefore, following property settings apply
 ' to next created items.
 Net1.Item = 0
 Net1.Shape = 1  'Default shape = Rectangle.
 Net1.FillColor = RGB(255, 255, 192) 'Default Fill color
 Net1.DrawColor = RGB(0, 0, 128)     'Default Draw color
 Net1.Oriented = True                'Oriented links

 ' Create first node. It has a rectangular shape.
 Net1.EditAction = 0
 Net1.X1 = 100
 Net1.Y1 = 100
 Net1.X2 = 2000
```

```
  Net1.Y2 = 500
  Net1.Text = "A network to implement ?"
  n1 = Net1.Item

  ' Create second node. It has a rectangular shape.
  Net1.EditAction = 0
  Net1.X1 = 2200
  Net1.Y1 = 300
  Net1.X2 = 3600
  Net1.Y2 = 700
  Net1.Text = "FlowChart needs ?"
  n2 = Net1.Item

  ' Create a third node. No shape is indicated.
  ' Therefore its shape is the default one: ellipse.
  Net1.EditAction = 0
  Net1.Shape = 0 ' Ellipse
  Net1.X1 = 1100
  Net1.Y1 = 1500
  Net1.X2 = 3000
  Net1.Y2 = 2000
  Net1.Text = "Use EasyNet.vbx !!"
  n3 = Net1.Item

  ' Create first link
  Net1.Org = n1
  Net1.Dst = n3
  Net1.EditAction = 1

  ' Create second link with an extra point (2 segments)
  Net1.Org = n2
  Net1.Dst = n3
  Net1.EditAction = 1
  Net1.PointCount = 1
  Net1.PointX(0) = 3200
  Net1.PointY(0) = 1000

  ' Unselect last created link
  Net1.Item = 0
End Sub
```

User Data Association

You can associate data to each item (node or link) with following properties:

- ItemTag property associates a string that is NOT displayed.
  The EasyNet control maintains the memory for the tags associated to items.
  This tag can be used to store user data.
- Data property associates a long integer that can be used to store a reference to a user data.
- Type property associates an integer that can be used to store an identifier or a type.

Navigation

You can navigate in the network diagram with the three following properties:

- LoopAction property has to be called first in order to indicate the type of navigation to perform.
- Then, a call to LoopCount gives the count of items involved in this navigation.
- Then, you get each item with LoopItem property.

You can retrieve origin and destination node of a link with Org and Dst properties.
Oriented property specifies if a link is oriented or not.

Example:
Makes color of all "out" links of all selected nodes be red.
Two calls to LoopAction property cannnot be cascaded so you have first to memorize the selected nodes in an array in order to work with them.

```
Sub Exercice ()
  Dim nbnode, nblink, i, j As Integer
  Dim Node() As Long

  ' Do a loop with selected nodes
  Net1.LoopAction = 2

  ' Get count of selected nodes
  nbnode = Net1.LoopCount

  ' If no selected nodes, nothing to do
  If nbnode = 0 Then Exit Sub

  ' Memorize selected nodes in a dynamic array.
  ReDim Node(1 To nbnode)
  For i = 1 To nbnode
    Node(i) = Net1.LoopItem(i - 1)
  Next i

  ' For each node of our array...
  For i = 1 To nbnode
    ' ... makes it be the current item
    Net1.Item = Node(i)

    ' Do a loop with all leaving (out) links of the current node
    Net1.LoopAction = 4

    ' Get count of selected nodes
    nblink = Net1.LoopCount

    ' For each link leaving the current node...
    For j = 1 To nblink
      Net1.Item = Net1.LoopItem(j - 1)
      Net1.DrawColor = RGB(255, 0, 0)
      End If
    Next j
  Next i

 ' Don't forget to delete the array
 Erase Node
End Sub
```

Capabilities

Following properties allow to set capabilities for an EasyNet control:

- CanDrawNode
- CanDrawLink
- CanMoveNode
- CanSizeNode
- CanStretchLink
- CanMultiLink
- DisplayHandles
- DoAddLink
- DoAddNode
- DoChange
- DoSelChange
- MultiSel
- ReadOnly
- ScrollBars
- ShowGrid
- xGrid
- yGrid

Saving/Loading

Saving an EasyNet diagram is under the responsability of the VB application that uses an EasyNet control. The ImageFile property used in conjunction with EditAction property only allows to save an image of the EasyNet diagram. this image file can be used by other drawing applications but it cannot be loaded up again by EasyNet.

You may see demonet1 sample that is supplied with the package in order to see a way to save an EasyNet diagram. It is just an example. You may use another method or/and save more or less properties for each item. You may use a sequential, a binary or a random file format. Let us give another example using a sequential file. You may copy this code into clipboard and paste it in one of your application modules.

Example:

```
' ----------------------------------------------------------------
'  This procedure saves an EasyNet diagram in a sequential file.
'  It saves:
' - the version number
' - the nodes count
' - the links count
' - every properties of each node (except Picture property)
' - every properties of each link.
'
'  Picture property is not saved but you may instead manage
' a correspondance between node types and pictures. For
' instance when you load your file, your VB application knows
' that node of type 1 have one icon, nodes of type 2 have another
' icon, etc...
'
'  This program is just an example to show how an EasyNet file
' may be saved to disk.
'  You may proceed differently: for instance, use a binary or
' a random file and save only the properties you need for your
' application.
'  You may consider this program as a starting point to write
' your EasyNet saving/loading procedures adapted to your needs.
'
' THE CODE PROVIDED HEREUNDER IS PROVIDED AS IS WITHOUT WARRANTY
' OF ANY KIND.
' ----------------------------------------------------------------
'

' Following type is used for loading only.
Type ItemRec
  Type As Integer
  Data As Long
  DrawColor As Long
  DrawWidth As Integer
  DrawStyle As Integer
  Sleeping As Integer
  Hiding As Integer
  ItemTag As String
  Text As String
  Shape As Integer
  FillColor As Long
  Transparent As Integer
  Alignment As Integer
  X1 As Long
  Y1 As Long
  X2 As Long
  Y2 As Long
  Oriented As Integer
  LinkHead As Integer
  SrcNode As Long
  DstNode As Long
  Points As Integer
End Type


Sub SaveEasyNetFile (Net1 As Control, Filename As String)
  Dim i, j As Integer
  Dim NodeCount As Integer
  Dim LinkCount As Integer
  Dim PointCount As Integer
  Dim TextLength As Integer
  Dim TagLength As Integer
  Dim Text As String
  Dim ItemTag As String
  Dim node() As Long
  Dim nodeId As Long
  Dim Org As Long
  Dim Dst As Long
  Dim l As Long
  Dim ptx() As Long
  Dim pty() As Long
  Dim Item As Long
  Dim length As Integer
  Dim s As String
  Dim CR As String

  CR = Chr$(13)

  Open Filename For Output As 1
  Print #1, "EASYNET VERSION = " + Format$(Net1.version)

  ' Node count
  Net1.LoopAction = 0
  NodeCount = Net1.LoopCount
  Print #1, "Nodes = " + Format$(NodeCount)

  ' Link count
  Net1.LoopAction = 1
  LinkCount = Net1.LoopCount
  Print #1, "Links = " + Format$(LinkCount)

  If NodeCount = 0 Then
    Close
    Exit Sub
  End If

  '-----------
  ' Save nodes
```

```
'-----------

Net1.LoopAction = 0  ' Do a nodes loop

' Allocate array to store nodes identifier. This array will be used
' when saving links.
ReDim node(1 To NodeCount)

' For each node:
'   - save its identifier in an array
'   - make it the current one
'   - save its properties in the file

For i = 0 To NodeCount - 1
  node(i + 1) = Net1.LoopItem(i)

  ' Make node the current item
  Net1.Item = node(i + 1)

  ' Get text and its length
  Text = Net1.Text
  TextLength = Len(Text)

  ' Get tag and its length
  ItemTag = Net1.ItemTag
  TagLength = Len(ItemTag)

  ' Save current node properties
  Print #1, "Begin Node " + Format$(i + 1)
  Print #1, "  Type = " + Net1.Type
  Print #1, "  Data = " + Net1.Data
  Print #1, "  DrawColor = " + Net1.DrawColor
  Print #1, "  DrawWidth = " + Net1.DrawWidth
  Print #1, "  DrawStyle = " + Net1.DrawStyle
  If Net1.Sleeping = True Then
    Print #1, "  Sleeping = " + Net1.Sleeping
  End If
  If Net1.Hiding = True Then
    Print #1, "  Hiding = " + Net1.Hiding
  End If
  If Net1.Transparent = True Then
    Print #1, "  Transparent = " + Net1.Transparent
  end If
  Print #1, "  Alignment = " + Net1.Alignment
  Print #1, "  Shape = " + Net1.Shape
  Print #1, "  FillColor = " + Net1.FillColor
  Print #1, "  X1 = " + Net1.X1
  Print #1, "  Y1 = " + Net1.Y1
  Print #1, "  X2 = " + Net1.X2
  Print #1, "  Y2 = " + Net1.Y2
  If TextLength > 0 Then
    s = Text
    length = InStr(s, CR)
    While length > 0
      Print #1, "  Text = " + Left$(s, length - 1)
      s = Mid$(s, length + 2)
      length = InStr(s, CR)
    Wend
    Print #1, "  Text = " + s
  End If
  If TagLength > 0 Then
    s = ItemTag
    length = InStr(s, CR)
    While length > 0
      Print #1, "  ItemTag = " + Left$(s, length - 1)
      s = Mid$(s, length + 2)
      length = InStr(s, CR)
    Wend
    Print #1, "  ItemTag = " + s
  End If
  Print #1, "End"
Next i

'-----------
' Save links
'-----------

Net1.LoopAction = 1  ' Do a links loop

' For each link:
'   - make it the current one
'   - find its origin and destination nodes
'   - save its properties in the file

For i = 0 To LinkCount - 1
  ' Make link the current item
  Net1.Item = Net1.LoopItem(i)

  ' Find origin
  Org = 0
  nodeId = Net1.Org
  For j = 1 To NodeCount
    If node(j) = nodeId Then
      Org = j
      Exit For
    End If
  Next

  ' Find destination
  Dst = 0
  nodeId = Net1.Dst
  For j = 1 To NodeCount
    If node(j) = nodeId Then
      Dst = j
      Exit For
    End If
  Next

  ' Get text and its length
  Text = Net1.Text
```

```
 TextLength = Len(Text)

 ' Get tag and its length
 ItemTag = Net1.ItemTag
 TagLength = Len(ItemTag)

 ' Get Number of points
 PointCount = Net1.PointCount

 ' Get points
 If PointCount > 0 Then
   ReDim ptx(0 To PointCount - 1)
   ReDim pty(0 To PointCount - 1)
   For l = 0 To PointCount - 1
     ptx(l) = Net1.PointX(l)
     pty(l) = Net1.PointY(l)
   Next
 End If

' Save current link properties
 Print #1, "Begin Link " + Format$(i + 1)
 Print #1, "  Type = " + Net1.Type
 Print #1, "  Data = " + Net1.Data
 Print #1, "  DrawColor = " + Net1.DrawColor
 Print #1, "  DrawWidth = " + Net1.DrawWidth
 Print #1, "  DrawStyle = " + Net1.DrawStyle
 If Net1.Sleeping = True Then
   Print #1, "  Sleeping = " + Net1.Sleeping
 End If
 If Net1.Hiding = True Then
   Print #1, "  Hiding = " + Net1.Hiding
 End If
 Print #1, "  Oriented = " + Net1.Oriented
 Print #1, "  LinkHead = " + Net1.LinkHead
 Print #1, "  Src = " + Format$(Org)
 Print #1, "  Dst = " + Format$(Dst)
 Print #1, "  Points = " + Format$(PointCount)
 If PointCount > 0 Then
   For l = 0 To PointCount - 1
     Print #1, "    " + Format$(ptx(l)) + "," + Format$(pty(l))
   Next
 End If
 If TextLength > 0 Then
   s = Text
   length = InStr(s, CR)
   While length > 0
     Print #1, "  Text = " + Left$(s, length - 1)
     s = Mid$(s, length + 2)
     length = InStr(s, CR)
   Wend
   Print #1, "  Text = " + s
 End If
 If TagLength > 0 Then
   s = ItemTag
   length = InStr(s, CR)
```

```
      While length > 0
        Print #1, "  ItemTag = " + Left$(s, length - 1)
        s = Mid$(s, length + 2)
        length = InStr(s, CR)
      Wend
      Print #1, "  ItemTag = " + s
    End If
    Print #1, "End"
  Next i

  Erase node
  Erase ptx
  Erase pty

' Close file
  Close
End Sub

'-----------------------------------------------------
' (See comment of OpenEasyNetFile subroutine.)
'-----------------------------------------------------
'
Sub OpenEasyNetFile (Net1 As Control, Filename As String)
  Dim s As String
  Dim value As String
  Dim keyword As String
  Dim length As Integer
  Dim i As Integer
  Dim NodeCount As Integer
  Dim LinkCount As Integer
  Dim version As Variant
  Dim ir As ItemRec
  Dim l As Long
  Dim ptx() As Long
  Dim pty() As Long
  Dim CRLF As String

  CRLF = Chr$(13) + Chr$(10)

  Open Filename For Input As #1

  Line Input #1, s  ' Version
  version = Val(Mid$(s, InStr(s, "=") + 1))
  If version <> Net1.version Then
    MsgBox "File created by another EasyNet version!"
    Beep
    Exit Sub
  End If

  ' Node count
  Line Input #1, s
  NodeCount = Val(Mid$(s, InStr(s, "=") + 1))

  ' Link count
  Line Input #1, s
```

```
LinkCount = Val(Mid$(s, InStr(s, "=") + 1))

If NodeCount = 0 Then
  Close
  Exit Sub
End If


ReDim node(1 To NodeCount)

' Load all nodes
For i = 1 To NodeCount
  Line Input #1, s  ' Skip Begin keyword
  length = InStr(s, " ")
  keyword = Left$(s, length - 1)

  If keyword = "Begin" Then
    Net1.Item = 0

    ' Default values
    ir.Type = 0
    ir.Data = 0
    ir.ItemTag = ""
    ir.Text = ""
    ir.DrawColor = Net1.DrawColor
    ir.DrawWidth = Net1.DrawWidth
    ir.DrawStyle = Net1.DrawStyle
    ir.Sleeping = Net1.Sleeping
    ir.Hiding = Net1.Hiding
    ir.FillColor = Net1.FillColor
    ir.Shape = Net1.Shape
    ir.Alignment = Net1.Alignment
    ir.Transparent = Net1.Transparent
    ir.X1 = 0
    ir.Y1 = 0
    ir.X2 = 0
    ir.Y2 = 0

    Do
     Line Input #1, s  ' Skip Begin keyword
     s = LTrim$(s)
     length = InStr(s, " ")
     If length > 0 Then
       keyword = Left$(s, length - 1)
     Else
       keyword = s
     End If
     If keyword = "End" Then
       Exit Do
     End If
     value = Mid$(s, length + 2)

     ' Load each node property
     Select Case keyword
     Case "Type"
       ir.Type = Val(value)
     Case "Data"
       ir.Data = Val(value)
     Case "DrawColor"
       ir.DrawColor = Val(value)
     Case "DrawWidth"
       ir.DrawWidth = Val(value)
     Case "DrawStyle"
       ir.DrawStyle = Val(value)
     Case "Sleeping"
       ir.Sleeping = Val(value)
     Case "Hiding"
       ir.Hiding = Val(value)
     Case "Transparent"
       ir.Transparent = Val(value)
     Case "Alignment"
       ir.Alignment = Val(value)
     Case "Shape"
       ir.Shape = Val(value)
     Case "FillColor"
       ir.FillColor = Val(value)
     Case "X1"
       ir.X1 = Val(value)
     Case "X2"
       ir.X2 = Val(value)
     Case "Y1"
       ir.Y1 = Val(value)
     Case "Y2"
       ir.Y2 = Val(value)
     Case "ItemTag"
       If ir.ItemTag = "" Then
         ir.ItemTag = value
       Else
         ir.ItemTag = ir.ItemTag + CRLF + value
       End If
     Case "Text"
       If ir.Text = "" Then
         ir.Text = value
       Else
         ir.Text = ir.Text + CRLF + value
       End If
     End Select
    Loop

    ' Create Node
    Net1.EditAction = 0

    Net1.Type = ir.Type
    Net1.Data = ir.Data
    Net1.DrawColor = ir.DrawColor
    Net1.DrawWidth = ir.DrawWidth
    Net1.DrawStyle = ir.DrawStyle
    Net1.Sleeping = ir.Sleeping
    Net1.Hiding = ir.Hiding
    Net1.FillColor = ir.FillColor
```

```
    Net1.Alignment = ir.Alignment
    Net1.Shape = ir.Shape
    Net1.Transparent = ir.Transparent
    Net1.X1 = ir.X1
    Net1.Y1 = ir.Y1
    Net1.X2 = ir.X2
    Net1.Y2 = ir.Y2
    Net1.ItemTag = ir.ItemTag
    Net1.Text = ir.Text

    ' Store its identifier (will be used for links loading)
    node(i) = Net1.Item
  End If
Next i

'List of link
For i = 1 To LinkCount
  Line Input #1, s  ' Skip Begin keyword
  length = InStr(s, " ")
  keyword = Left$(s, length - 1)

  If keyword = "Begin" Then
    Net1.Item = 0

    ' Default values
    ir.Type = 0
    ir.Data = 0
    ir.ItemTag = ""
    ir.Text = ""
    ir.DrawColor = Net1.DrawColor
    ir.DrawWidth = Net1.DrawWidth
    ir.DrawStyle = Net1.DrawStyle
    ir.Sleeping = Net1.Sleeping
    ir.Hiding = Net1.Hiding
    ir.Oriented = Net1.Oriented
    ir.LinkHead = Net1.LinkHead
    ir.SrcNode = 0
    ir.DstNode = 0
    ir.Points = 0

    Do
     Line Input #1, s  ' Skip Begin keyword
     s = LTrim$(s)
     length = InStr(s, " ")
     If length > 0 Then
       keyword = Left$(s, length - 1)
     Else
       keyword = s
     End If
     If keyword = "End" Then
       Exit Do
     End If
     value = Mid$(s, length + 2)

     ' Load each link property
```

```
 Select Case keyword
 Case "Type"
   ir.Type = Val(value)
 Case "Data"
   ir.Data = Val(value)
 Case "DrawColor"
   ir.DrawColor = Val(value)
 Case "DrawWidth"
   ir.DrawWidth = Val(value)
 Case "DrawStyle"
   ir.DrawStyle = Val(value)
 Case "Sleeping"
   ir.Sleeping = Val(value)
 Case "Hiding"
   ir.Hiding = Val(value)
 Case "Oriented"
   ir.Oriented = Val(value)
 Case "LinkHead"
   ir.LinkHead = Val(value)
 Case "ItemTag"
   If ir.ItemTag = "" Then
     ir.ItemTag = value
   Else
     ir.ItemTag = ir.ItemTag + CRLF + value
   End If
 Case "Text"
   If ir.Text = "" Then
     ir.Text = value
   Else
     ir.Text = ir.Text + CRLF + value
   End If
 Case "Src"
   ir.SrcNode = node(Val(value))
 Case "Dst"
   ir.DstNode = node(Val(value))
 Case "Points"
   ir.Points = Val(value)
   ' Get points
   If ir.Points > 0 Then
     ReDim ptx(0 To ir.Points - 1)
     ReDim pty(0 To ir.Points - 1)
     For l = 0 To ir.Points - 1
       Line Input #1, s  ' Read point
       s = LTrim$(s)
       length = InStr(s, ",")
       ptx(l) = Val(Left$(s, length - 1))
       pty(l) = Val(Mid$(s, length + 1))
     Next l
   End If
 End Select
Loop

' Set origin and destination nodes for next created link
Net1.Org = ir.SrcNode
Net1.Dst = ir.DstNode
```

```
      ' Create Link
      Net1.EditAction = 1

      Net1.Type = ir.Type
      Net1.Data = ir.Data
      Net1.DrawColor = ir.DrawColor
      Net1.DrawWidth = ir.DrawWidth
      Net1.DrawStyle = ir.DrawStyle
      Net1.Sleeping = ir.Sleeping
      Net1.Hiding = ir.Hiding
      Net1.Oriented = ir.Oriented
      Net1.LinkHead = ir.LinkHead
      Net1.ItemTag = ir.ItemTag
      Net1.Text = ir.Text
      Net1.PointCount = ir.Points
      For l = 0 To ir.Points - 1
        Net1.PointX(l) = ptx(l)
        Net1.PointY(l) = pty(l)
      Next l
    End If
  Next i

  ' Erase dynamic arrays
  Erase ptx
  Erase pty
  Erase node

  ' Close file
  Close
End Sub
```

Performance tuning

Setting following properties to False allows to increase speed dramatically:

- DoAddLink
- DoAddNode
- DoChange
- DoSelChange
- Repaint
- CheckItem

Example:

You may insert this portion of code each time you need to do a time consuming task like saving an EasyNet diagram or navigating in the diagram.

```
' Setting those properties to False improve speed
  Net1.Repaint = False
  Net1.DoChange = False
  Net1.DoSelChange = False
  Net1.DoAddNode = False
  Net1.DoAddLink = False
  Net1.CheckItem = False
```

When you have terminated your task, you may reset those properties to True.

```
  Net1.Repaint = True
  Net1.DoChange = True
  Net1.DoSelChange = True
  Net1.DoAddNode = True
  Net1.DoAddLink = True
  Net1.CheckItem = True
```

Limits

For one EasyNet control:

- the maximum number of nodes is **1000**.
- the maximum number of links is **1000**.
- the maximum number of link points is **14**.
  (therefore, the maximum number of link segments is **15**).
- the total amount of memory available for all the text in one EasyNet control is **64 K.**

Of course, one application can manage simultaneously several EasyNet controls
(for instance, in MDI child windows).

Remarks

The node/link limitations are due to the memory scheme used to manage EasyNet items. This memory scheme is based upon subsegment allocation. One segment (64K) is used for nodes, another for link, another for text, etc... Each node or link needs 64 bytes which explains the 1000 limit. In fact, the maximum amount of memory used by each EasyNet control is **7 x 64K**.

Properties

All the properties are listed below. Properties that apply only to the EasyNet Custom Control, or require special consideration when used with it, are underlined. They are documented in this help file. See the Visual Basic *Language Reference* or online Help for documentation of the remaining properties.

(About)	Alignment	BackColor	BackPicture
BorderStyle	Caption	CanDrawNode	CanDrawLink
CanMoveNode	CanSizeNode	CanStretchLink	CanMultiLink
CheckItem	CtlName	DisplayHandles	DoAddLink
DoAddNode	DoChange	DoSelChange	Data
Dst	DragIcon	DragMode	DrawColor
DrawStyle	DrawWidth	EditAction	Enabled
FillColor	FontBold	FontItalic	FontName
FontSize	FontStrike	FontUnder	Height
HelpContextId	Hiding	Hwnd	ImageFile
Index	IsLink	Item	ItemTag
Left	LinkHead	LoopAction	LoopCount
LoopItem	MousePointer	MultiSel	Oriented
Org	Parent	Picture	PointCount
PointedArea	PointedItem	PointX	PointY
ReadOnly	Repaint	ShowGrid	ScrollBars
SelectMode	Shape	Sleeping	TabIndex
TabStop	Tag	Text	Top
Transparent	Type	Version	Visible
Width	X1	X2	xGrid
xScroll	Y1	Y2	yGrid
yScroll			

Events

All the events are listed below. Events that apply only to the EasyNet Custom Control, or require special consideration when used with it, are underlined. They are documented in this help file.   See the Visual Basic *Language Reference* or online Help for documentation of the remaining events.

<u>AddLink</u>	<u>AddNode</u>	<u>Change</u>	Click
DblClick	DragDrop	DragOver	<u>ErrSpace</u>
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp
<u>SelChange</u>			

EditAction Property

Description

Specifies an action that applies to selected items or that allows to select or unselect items.

Not available at design time; write only at run time.

Usage

[*form*.]*NET*.**EditAction**[= *setting*]

Settings

The EditAction property settings are:

Setting	Description
0	create a node
1	create a link
2	delete selected nodes (and their links)
3	select all nodes.
4	unselect.
5	copy selected nodes onto the clipboard in a metafile format.
6	clear network diagram (all items are deleted)
7	copy all the diagram onto the clipboard in a metafile format.
8	the image of the diagram is written to disk as a metafile (.WMF). For this option to work, the ImageFile property must be set to provide a name for the file. Moreover, you need to have a registered version of EasyNet.

Data Type

Integer (enumerated)

Remarks

Link creation: The link that is created when setting EditAction to 1 is a link that links the nodes specified by Org and Dst properties. If one of this node is not valid, the link is not created.

Selection: Only nodes can be selected.by the user.

Delete: When a node is deleted, all its links are also deleted. A link cannot exist without its origin and destination nodes. If one of these two nodes is deleted, the link is also deleted.

See Also

Drawing

FillColor Property

Description

If current item is 0, sets or returns the "current" filling node color (the filling color used for next created nodes).
If current item is a node, sets or returns its color (the color with which the node is filled).
If current item is a link, writing has no effect and reading returns 0.
This property has no effect if Transparent property is set.

Usage

[*form*.]*NET*.**FillColor**[= *color* &]

Settings

The FillColor property settings are:

Setting	Description
Normal RGB Colors	Color set with RGB or QBColor function in code
System Default Colors from	Colors specified with the system color constants CONSTANT.TXT, a Visual Basic file that you can load into a project's global module. Window's substitutes the user's choices, as specified through the user's Control Panel Settings.

By default, FillColor is set to 0 (black)

Data Type

Long

See Also

Drawing

DrawColor Property

Description

If current item is 0, sets or returns the "current" drawing color (the drawing color used for next created items).
If current item is not 0, sets or returns its drawing color.

Usage

[*form*.]*NET*.**DrawColor**[= *color* &]

Settings

The DrawColor property settings are:

Setting	Description
Normal RGB Colors	Color set with RGB or QBColor function in code
System Default Colors from	Colors specified with the system color constants CONSTANT.TXT, a Visual Basic file that you can load into a project's global module. Window's substitutes the user's choices, as specified through the user's Control Panel Settings.

By default, DrawColor is set to 0 (black)

Data Type

Long

See Also

Drawing

DrawStyle Property

Description

If current item is 0, sets or returns the "current" drawing style (the drawing style used for next created items).
If current item is not 0, sets or returns the item drawing style.

Usage

[*form*.]*NET*.**DrawStyle**[= *size*]

Setting

The DrawStyle property settings are:

Setting	Description
0	(Default) Solid
1	Dash
2	Dot
3	Dash-Dot
4	Dash-Dot-Dot
5	Transparent
6	Inside Solid

Data Type

Integer (enumerated)

Remarks

If DrawWidth is set to a value greater than 1, then DrawStyles 1 through 4 produce a solid line (the DrawStyle property value is not changed). If DrawWidth is set to 1, DrawStyle produces the effect described above for each setting.

See Also

Drawing

DrawWidth Property

Description

If current item is 0, sets or returns the "current" drawing pen width (the drawing pen width used for next created items).
If current item is not 0, sets or returns the item drawing pen width.

Usage

[*form*.]*NET*.**DrawWidth**[= *size*]

Setting

You can set DrawWidth to a value of 1 to 8 (pixels).

Data Type

Integer

See Also

Drawing

Shape Property

Description

If current item is 0, sets or returns the "current" node shape (the shape used for next created nodes).
If current item is a node, sets or returns its shape (ellipse, rectangle, round rectangle, diamond).
If current item is a link, writing has no effect and reading returns 0.

Usage

[*form*.]*NET*.**Shape**[= *shape*]

Settings

The Shape property settings are:

Setting	Description
0	Ellipse
1	Rectangle
2	Round rectangle
3	Diamond

By default, Shape is set to 0 (ellipse)

Data Type

Integer

See Also

Drawing

LinkHead Property

If current item is 0, sets or returns the "current" link arrowhead shape (the arrowhead used for next created links).
If current item is a node, writing has no effect and reading returns 0.
If current item is a link, sets or returns its arrowhead

Usage

[*form*.]*NET*.**LinkHead**[= *shape*]

Settings

The LinkHead property settings are:

Setting	Description
0	Filled arrow
1	Filled circle
2	Empty arrow
3	Empty circle

By default, LinkHead is set to 0

Data Type

Integer

See Also

Drawing

Alignment Property

Description

If current item is 0, sets or returns the "current" text alignment style
(the text alignment style used for next created nodes).
If current item is a node, sets or returns its text alignment style.
If current item is a link, writing has no effect and reading returns 0.

Usage

[*form*.]*NET*.**Alignment**[= alignment &]

Settings

The Alignment property settings are:

Setting	Description
0	Left - TOP
1	Left - MIDDLE
2	Left - BOTTOM
3	Right - TOP
4	Right - MIDDLE
5	Right - BOTTOM
6	Center - TOP
7	Center - MIDDLE
8	Center - BOTTOM

Data Type

Integer

See Also

Drawing

Transparent Property

Description

If current item is 0, specify if next created nodes will be transparent or not.
If current item is a node, specify if it is transparent or not.
If current item is a link, writing has no effect and reading returns 0.

Usage

[*form*.]*NET*.**Transparent**[=   {True | False}]

Settings

The Transparent property settings are:

Setting	Description
False	Transparent
True	(default) Opaque

Data Type

Integer (Boolean)

See also

Drawing

X1, Y1, X2, Y2 Property

Description

If current item is 0, sets or returns the coordinates of upper left point (X1, Y1) or lower right point (X2, Y2) of the bounding rectangle of next created node.
If current item is a node, sets or returns the coordinates of upper left point (X1, Y1) or lower right point (X2, Y2) of its bounding rectangle.
If current item is a link, writing those properties has no effect and reading returns the coordinates of upper left point (X1, Y1) or lower right point (X2, Y2) of its bounding rectangle.
Not available at design time.

Usage

[*form*.]*NET*.**X1**[= *numeric expression*]

[*form*.]*NET*.**Y1**[= *numeric expression*]

[*form*.]*NET*.**X2**[= *numeric expression*]

[*form*.]*NET*.**Y2**[= *numeric expression*]

Data Type

Long

See Also

Drawing

PointCount Property

Description

If current item is 0 or is a node, writing this property has no effect
and reading it returns 0.
If current item is a link, sets or returns the number of its points.
Not available at design time.

Usage

[*form*.]*NET*.**PointCount**[= *numeric expression*]

Data Type

Integer

Remarks

A link point is a point that joins two segments of a link. If a link has **n** points,
it is composed of **n+1** segments.
The maximum value for the number of link points is **14**.

See Also

Drawing

PointX Property

Description

If current item is 0 or is a node, writing this property has no effect and reading it returns 0.
If current item is a link, sets or returns a long integer value that identifies an x position of a specified link point.
Not available at design time.

Usage

[*form*.]*NET*.**PointX**(index)[= *numeric expression*]

Data Type

Long

Remarks

If current item is a link reading this property has special meanings if index has a negative value between -1 and -4:

- **-1**: returns x position of intersection point between origin node border and link.
- **-2**: returns x position of intersection point between destination node border and link
- **-3:** if link is oriented, returns x position of one arrowhead point. If link is not oriented, it has the same effect as the case -2.
- **-4**: if link is oriented, returns x position of the other arrowhead point.

  If link is not oriented, it has the same effect as the case -2.

See Also

Drawing

Example *Print an arrow*

```
Dim i, nbpoint As Integer
Dim l, ptx1, pty1, ptx2, pty2, ptx3, pty3 As Long
Dim ptx(), pty() As Long

'Number of extra points
nbpoint = Net1.PointCount

'Allocate an array of nbpoint + 2
ReDim ptx(0 To nbpoint + 1)
ReDim pty(0 To nbpoint + 1)

'First point (intersection between origin node border and link)
ptx(0) = Net1.PointX(-1)
pty(0) = Net1.PointY(-1)

' Normal extra points
For l = 1 To nbpoint
  ptx(l) = Net1.PointX(l - 1)
  pty(l) = Net1.PointY(l - 1)
Next l
```

```
'Last point (intersection between destination node border and link)
ptx(nbpoint + 1) = Net1.PointX(-2)
pty(nbpoint + 1) = Net1.PointY(-2)

' Draw all link segments
For l = 0 To nbpoint
  printer.Line (ptx(l), pty(l))-(ptx(l+1), pty(l+1)),   Net1.DrawColor
Next l

'Get point arrow head
ptx1 = Net1.PointX(-3)
pty1 = Net1.PointY(-3)
ptx2 = Net1.PointX(-4)
pty2 = Net1.PointY(-4)
ptx3 = ptx(nbpoint + 1)
pty3 = pty(nbpoint + 1)

'Draw arrow head
printer.Line (ptx1, pty1)-(ptx2, pty2), Net1.DrawColor
printer.Line (ptx1, pty1)-(ptx3, pty3), Net1.DrawColor
printer.Line (ptx3, pty3)-(ptx2, pty2), Net1.DrawColor
```

PointY Property

Description

If current item is 0 or is a node, writing this property has no effect and reading it returns 0.
If current item is a link, sets or returns a long integer value that identifies an y position of a specified link point.
Not available at design time.

Usage

[*form*.]*NET*.**PointY**(index)[= *numeric expression*]

Data Type

Long

Remarks

If current item is a link, reading this property has special meanings if index has a negative value between -1 and -4:

- **-1**: returns y position of intersection point between origin node border and link.
- **-2**: returns y position of intersection point between destination node border and link
- **-3:** if link is oriented, returns y position of one arrowhead point.
  If link is not oriented, it has the same effect as the case -2.
- **-4**: if link is oriented, returns y position of the other arrowhead point.

  If link is not oriented, it has the same effect as the case -2.

See Also

Drawing

Example *Print an arrow*

```
Dim i, nbpoint As Integer
Dim l, ptx1, pty1, ptx2, pty2, ptx3, pty3 As Long
Dim ptx(), pty() As Long

'Number of extra points
nbpoint = Net1.PointCount

'Allocate an array of nbpoint + 2
ReDim ptx(0 To nbpoint + 1)
ReDim pty(0 To nbpoint + 1)

'First point (intersection between origin node border and link)
ptx(0) = Net1.PointX(-1)
pty(0) = Net1.PointY(-1)

' Normal extra points
For l = 1 To nbpoint
  ptx(l) = Net1.PointX(l - 1)
  pty(l) = Net1.PointY(l - 1)
Next l
```

```
'Last point (intersection between destination node border and link)
ptx(nbpoint + 1) = Net1.PointX(-2)
pty(nbpoint + 1) = Net1.PointY(-2)

' Draw all link segments
For l = 0 To nbpoint
  printer.Line (ptx(l), pty(l))-(ptx(l+1), pty(l+1)), Net1.DrawColor
Next l

'Get point arrow head
ptx1 = Net1.PointX(-3)
pty1 = Net1.PointY(-3)
ptx2 = Net1.PointX(-4)
pty2 = Net1.PointY(-4)
ptx3 = ptx(nbpoint + 1)
pty3 = pty(nbpoint + 1)

'Draw arrow head
printer.Line (ptx1, pty1)-(ptx2, pty2), Net1.DrawColor
printer.Line (ptx1, pty1)-(ptx3, pty3), Net1.DrawColor
printer.Line (ptx3, pty3)-(ptx2, pty2), Net1.DrawColor
```

Oriented Property

Description

If current item is 0, specify if next created links will be oriented or not.
If current item is a link, specify if it is oriented or not.
If current item is a node, writing has no effect and reading returns 0.
When a link is oriented, it is displayed with an arrowhead at its destination node.

Usage

[*form*.]*NET*.**Oriented**[= {True | False}]

Settings

The Oriented property settings are:

Setting	Description
False	no arrowhead
True	(default) one arrowhead

Data Type

Integer (Boolean)

See also

Navigation

Org Property

Description

Sets the origin node of next created links (The value of the current item has no effect when writing this property).
If current item is 0, or if it is not a link, returns the origin node of next created links.
If current item is a link, returns its origin node.
Not available at design time.

Usage

[*form*.]*NET*.**Org**[= *idNode*]

Data Type

Long

Remarks

It is not possible to change directly the origin node of a link. If you want to do that, you have to memorize the link properties, destroy it, create a new one with the new origin node and sets previous saved properties.

See Also

Navigation

Dst Property

Description

Sets the destination node of next created links (The value of the current item has no effect when writing this property).
If current item is 0, or if it is not a link, returns the destination node of next created links.
If current item is a link, returns its destination node.
Not available at design time.

Usage

[*form*.]*NET*.**Dst**[= *idNode*]

Data Type

Long

Remarks

It is not possible to change directly the destination node of a link. If you want to do that, you have to memorize the link properties, destroy it, create a new one with the new destination node and sets previous saved properties.

See Also

Navigation

Item Property

Description

Sets or returns the current item (node or link). The current item is the selected one. Making an item be the current one allows to work with it (setting or getting its properties: position ,size, text, colors, etc).
Setting this property causes previous selection to disappear.
Not available at design time.

Usage

[*form*.]*NET*.**Item**[= *item*]

Data Type

Long

See Also

Items

IsLink Property

Description

Indicates if the current item is a link.
Not available at design time; read only at run time.

Usage

[*form*.]*NET*.**IsLink**

Settings

The IsLink property settings are:

Setting	Description
False	current item is 0 or it is a node
True	current item is not 0 and it is a link

Data Type

Integer (Boolean)

See Also

Items

Sleeping Property

Description

If current item is 0, specify if next created items will be in "sleeping mode" or not.
If current item is not 0, specify if it is in "sleeping mode" or not.
Not available at design time
When an item is in "sleeping mode", it is inactive and the user cannot interactively make it current or selected. He can do this only programmaticaly by saving its identifier in a global variable. Such an item can be used to display a bitmap or a text but the user cannot move, stretch or resize it with the mouse.

Usage

[*form*.]*NET***.Sleeping** [=   {True | False}]

Settings

The Sleeping property settings are:

Setting	Description
False	(default) The item is active.
True	The item is sleeping.

Data Type

Integer (Boolean)

See also

Items

LoopAction Property

Description

Specifies the type of item navigation to perform.
Not available at design time; write only at run time.

Usage

[*form*.]*NET*.**LoopAction** = *setting*

Settings

The LoopAction property settings are:

Setting	**Description**
0	all nodes
1	all links
2	all selected nodes
3	all links of a node
4	all links leaving current node (out links)
5	all links coming to current node (in links)
6	all nodes connected to a node (in and out nodes)
7	all destination nodes of current node
8	all origin nodes of current node

Data Type

Integer (enumerated)

Remarks

1. This property is to be used in conjonction with LoopCount and LoopItem properties:

- LoopAction specifies the type of loop to do: for instance a loop among all current node links (LoopAction = 3).
- After a call to LoopAction, LoopCount indicates the number of items involved in this loop.
- Finally, LoopItem allows to read each item and to perform any work with it.

2. Two calls to LoopAction property cannnot be cascaded.

See Also

Navigation

LoopCount Property

Description

Specifies the count of items involved in a navigation action performed by a call to LoopAction property.
Not available at design time; read only at run time.

Usage

[*form*.]*NET*.**LoopCount**

Data Type

Integer

Remarks

This property has to be called just after a call to LoopAction property.

See Also

Navigation

LoopItem Property

Description

Returns an item selected in a navigation action performed by a call to LoopAction property.
Not available at design time; read only at run time.

Usage

[*form*.]*NET*.**LoopItem**(index)

Data Type

Long

See Also

Navigation

Type Property

Description

If current item is 0, writing this property has no effect and reading it returns 0.
If current item is not 0, sets or returns its associated integer data.
Not available at design time.

Usage

[*form*.]*NET*.**Type**[= *setting*]

Data Type

Integer

Remarks

Typically, this property allows the user to define node or link types. Like Data property, the value of Type property is not used by the EasyNet control but only stored. The meaning of this property depends on the application that uses it.

See Also

Data Association

Data Property

Description

If current item is 0, writing this property has no effect and reading it returns 0.
If current item is not 0, sets or returns its associated long data.
Not available at design time.

Usage

[*form*.]*NET*.**Data**[= *setting*]

Data Type

Long

Remarks

Like Type property, the value of Data property is not used by the EasyNet control but only stored. The meaning of this property depends on the application that uses it.

See Also

Data Association

Text Property

Description

If current item is 0, writing this property has no effect and reading it returns an empty string.
If current item is not 0 (node or link), sets or returns the text associated with this item. The EasyNet control maintains the memory for the strings associated to items.
Not available at design time.
The text associated to a node is displayed inside the node. It is a multiline display. The text is wrapped automatically inside the node. Linefeed and carriage return characters are supported.
The text associated to a link is displayed at the middle of its segment number n/2 + 1 (n is the number of segments). This text is displayed in a single line.

Usage

[*form*.]*NET*.**Text**[= *string expression*]

Data Type

String

Remarks

The total amount of memory available for all the text in one EasyNet control is **64 K.**

See Also

Drawing

ItemTag Property

Description

If current item is 0, writing this property has no effect and reading it returns an empty string.
If current item is not 0 (node or link), sets or returns a tag associated with this item. The EasyNet control maintains the memory for the tags associated to items.
Not available at design time.

Usage

[*form*.]*NET*.**ItemTag**[= *string expression*]

Data Type

String

Remarks

The total amount of memory available for all the tags in one EasyNet control is **64 K.**

See Also

Data Association

Picture Property

Description

If current item is 0, sets or returns the picture to be displayed in next created nodes.
If current item is a node, sets or returns the picture to be displayed in this node. This picture can be a bitmap or an icon.
If current item is a link, writing this property has no effect and reading it returns 0.
Not available at design time.

Usage

[*form*.]*NET*.**Picture**[= *picture*]

Settings

The Picture Property settings are:

Setting	Description
(none)	(Default)
(bitmap, icon)	Specifies a picture. You can also set this property using the LoadPicture function on a bitmap or an icon.

Data Type

Integer

See Also

Drawing

SelectMode Property

Description

Allow to enter in selection mode instead of drawing mode. This property has no effect if MultiSel property is not set.
Not available at design time.
The **selection mode** allows to select several items. You bring the mouse cursor into the EasyNet control, press the left button, move the mouse and release the left button. All nodes inside the selection rectangle are selected. Then you can unselect some items by clicking them with the mouse and simultaneously pressing the shift or control key. You can select them again by using the same method.

Usage

[*form*.]*NET*.**SelectMode**[= {True | False}]

Settings

The SelectMode Property settings are:

Setting	Description
False	(Default) Drawing mode.
True	Select mode is set.

Data Type

Integer (Boolean)

CanDrawNode Property

Description

Specify if you can create nodes interactively.

Usage

[*form*.]*NET*.**CanDrawNode**[= {True | False}]

Settings

The CanDrawNode Property settings are:

Setting	Description
False	Drawing nodes is not allowed.
True	(Default) Drawing nodes is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

CanDrawLink Property

Description

Specify if you can create links interactively.

Usage

[*form*.]*NET*.**CanDrawLink**[=   {True | False}]

Settings

The CanDrawLink Property settings are:

Setting	**Description**
False	Drawing links is not allowed.
True	(Default) Drawing links is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

CanMoveNode Property

Description

Specify if you can move (drag) nodes interactively.

Usage

[*form*.]*NET*.**CanMoveNode**[=   {True | False}]

Settings

The CanMoveNode Property settings are:

Setting	Description
False	Moving nodes is not allowed.
True	(Default) Moving nodes is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

CanSizeNode Property

Description

Specify if you can resize nodes interactively.

Usage

[*form*.]*NET*.**CanSizeNode**[=   {True | False}]

Settings

The CanSizeNode Property settings are:

Setting	**Description**
False	Sizing nodes is not allowed.
True	(Default) Sizing nodes is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

CanStretchLink Property

Description

Specify if you can "stretch" links (i.e add or remove segments)interactively

Usage

[*form*.]*NET*.**CanStretchLink**[=   {True | False}]

Settings

The CanStretchLink Property settings are:

Setting	Description
False	Stretching links is not allowed.
True	(Default) Stretching links is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

CanMultiLink Property

Description

Specify if you can have several links between two nodes.

Usage

[*form*.]*NET*.**CanMultiLink**[= {True | False}]

Settings

The CanMultiLink Property settings are:

Setting	Description
False	(Default) Multi links is not allowed.
True	Multi links is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

MultiSel Property

Description

Specify if multiselection mode is possible or not.

Usage

[*form*.]*NET*.**MultiSel**[=   {True | False}]

Settings

The MultiSel Property settings are:

Setting	**Description**
False	Multi selection is not allowed.
True	(Default) Multi selection is allowed.

Data Type

Integer (Boolean)

See Also

Capabilities

ReadOnly Property

Description

Set "read only" mode. In such a mode user interaction is not allowed.

Usage

[*form*.]*NET*.**ReadOnly**[=   {True | False}]

Settings

The ReadOnly Property settings are:

Setting	Description
False	(Default) "Read only" mode is set.
True	"Read only" mode is not set.

Data Type

Integer (Boolean)

See Also

Capabilities

ScrollBars Property

Description

Allows to add scrollbars for the EasyNet control. Read-only at run time.

Usage

[*form*.]*NET*.**ScrollBars**[= *setting*]

Settings

The ScrollBars Property settings are:

Setting	Description
0	(Default) No scrollbar.
1	Horizontal scrollbar.
2	Vertical scrollbar.
3	Both Horizontal and Vertical scrollbars.

Data Type

Integer (Enumerated)

See Also

Capabilities

xGrid, yGrid Property

Description

Sets or returns the grid values in twips.

Usage

[*form*.]*NET*.**xGrid**[= *numeric expression*]

[*form*.]*NET*.**yGrid**[= *numeric expression*]

Data Type

Long

See Also

Capabilities

ShowGrid Property

Description

Specify if the grid is displayed or not.

Usage

[*form*.]*NET*.**ShowGrid**[=   {True | False}]

Settings

The ShowGrid Property settings are:

Setting	Description
False	(Default) The grid is not displayed.
True	The grid is displayed.

Data Type

Integer (Boolean)

See Also

Capabilities

xScroll, yScroll Property

Description

Sets or returns the scroll values in twips.

Not available at design time.

Usage

[*form*.]*NET*.**xScroll**[= *numeric expression*]

[*form*.]*NET*.**yScroll**[= *numeric expression*]

Data Type

Long

PointedArea Property

Description

Returns the type of the area pointed by the mouse (sizing square, stretching square, linking square, node, over no special area).

Not available at design time;   read only at run time

Usage

[*form*.]*NET*.**PointedArea**

Settings

The PointedArea property settings are:

Setting	**Description**
0	Size NW-SE square area
1	Size N-S square area
2	Size NE-SW square area
3	Size W-E square area
4	Stretching square area
5	Linking square area
6	Node area
7	No special area.

Data Type

Integer

Remarks

This property allows to change dynamically the mouse pointer BEFORE the user clicks anywhere, to indicate what actions are possible.

For example, when the pointer is over one of the corner points of a node, it should change to the standard NE/SW or NW/SE diagonal arrow. When it is over a side node, it would be the N/S or E/W arrow.

PointedItem Property

Description

Returns the item identifier pointed by the mouse.

Not available at design time;   read only at run time

Usage

[*form*.]*NET*.**PointedItem**

Data Type

Long

BackPicture Property

Description

This property is the same as the standard Visual Basic Picture property except that it only supports bitmap (.BMP) files.

DoAddLink Property

Description

Specify if AddLink event can be fired. Setting this property to False increases speed performance.

Usage

[*form*.]*NET*.**DoAddLink**[= {True | False}]

Settings

The DoAddLink Property settings are:

Setting	Description
False	AddLink event cannot be fired
True	(Default) AddLink event can be fired

Data Type

Integer (Boolean)

See Also

Capabilities
Performance tuning

DoAddNode Property

Description

Specify if AddNode event can be fired. Setting this property to False increases speed performance.

Usage

[*form*.]*NET*.**DoAddNode**[= {True | False}]

Settings

The DoAddNode Property settings are:

Setting	Description
False	AddNode event cannot be fired
True	(Default) AddNode event can be fired

Data Type

Integer (Boolean)

See Also

Capabilities
Performance tuning

DoChange Property

Description

Specify if Change event can be fired. Setting this property to False increases speed performance.

Usage

[*form*.]*NET*.**DoChange**[= {True | False}]

Settings

The DoChange Property settings are:

Setting	Description
False	Change event cannot be fired
True	(Default) Change event can be fired

Data Type

Integer (Boolean)

See Also

Capabilities
Performance tuning

DoSelChange Property

Description

Specify if SelChange event can be fired. Setting this property to False increases speed performance.

Usage

[*form*.]*NET*.**DoSelChange**[= {True | False}]

Settings

The DoSelChange Property settings are:

Setting	Description
False	SelChange event cannot be fired
True	(Default) SelChange event can be fired

Data Type

Integer (Boolean)

See Also

Capabilities
Performance tuning

Repaint Property

Description

Specify if repainting the EasyNet control is allowed or not.Setting this property to False increases speed performance. Setting this property to True causes a refresh.

Not available at design time

Usage

[*form*.]*NET*.**Repaint**[= {True | False}]

Settings

The Repaint Property settings are:

Setting	Description
False	Repainting not allowed.
True	(Default) Repainting allowed

Data Type

Integer (Boolean)

See Also

Performance tuning

CheckItem Property

Description

Specify if item checking is performed or not. Setting this property to False increases speed performance.

IMPORTANT: Setting this property to False requires to be very cautious when using Item, Org and Dst properties. Setting wrong values to those properties when CheckItem is False may result in a General Protection Fault .

Not available at design time

Usage

[*form*.]*NET*.**CheckItem**[= {True | False}]

Settings

The CheckItem Property settings are:

Setting	Description
False	Item checking is not performed.
True	(Default) Item checking is performed

Data Type

Integer (Boolean)

See Also

Performance tuning

Version Property

Description

Returns the version of the EasyNet control currently loaded in memory.

Read only.

Usage

[*form*.]*NET*.**Version**

Data Type

Integer

Remarks

The version number is a three digit integer where the first digit is the major version number and the last two represent the minor version number. For example, if current version is 1.60, then this property returns 160.

Hiding Property

Description

If current item is 0, specify if next created items will be visible or not
If current item is not 0, specify if it is visible or not.
Not available at design time

Usage

[*form*.]*NET.***Hiding** [=   {True | False}]

Settings

The Hiding property settings are:

Setting	Description
False	(default) The item is visible.
True	The item is not visible.

Data Type

Integer (Boolean)

See also

Drawing

ImageFile Property

Description

Sets a file name to which the metafile is written when EditAction is set to 8.
If a path is not specified, the current directory is used.

Usage

[*form*.]*NET.***ImageFile** [= filename$]

Data Type

String

Remarks

The appropriate extension (.WMF) is appended automatically.

See also

EditAction

DisplayHandles Property

Description

Specify if handles are displayed. The handles are the little black squares on the selected item.

Usage

[*form*.]*NET*.**DisplayHandles**[= {True | False}]

Settings

The DisplayHandles Property settings are:

Setting	**Description**
False	Handles are not displayed.
True	(Default) Handles are displayed.

Data Type

Integer (Boolean)

Change Event

Description

Occurs when a change is made. (For instance, an item is added, moved, deleted or one of its properties is changed).

Syntax

Sub *NET*_**Change** ()

SelChange Event

Description

Occurs when selection is changed.

Syntax

Sub *NET*_**SelChange** ()

AddNode Event

Description

Occurs when a node is added.

Syntax

Sub *NET*_**AddNode** ()

Remarks

- Typically, this event allows the user to change a property of the node just after its creation and just before its display. For instance, if you need fixed size nodes, you have just to give values to X1, X2, Y1, Y2 properties:

```
Sub Net1_AddNode ()
  Net1.X2 = Net1.X1 + 500
  Net1.Y2 = Net1.Y1 + 500
End Sub
```

- In fact when a node is created, three events are generated in the following order:

SelChange

AddNode

Change

AddLink Event

Description

Occurs when a link is added.

Syntax

Sub *NET*_**AddLink** ()

Remarks

Typically, this event allows the user to change a property of the link just after its creation and just before its display.
In fact when a link is created, three events are generated in the following order:

SelChange

AddLink

Change

ErrSpace Event

Description

Occurs when no more memory is available.

Syntax

Sub *NET*_**ErrSpace** ()

Remarks

This event occurs only in the case where the total amount of memory available for all the text (64 K) is reached.

Registration

The demonstration version of the EasyNet control is FULLY FUNCTIONAL but may only be used in the development environment. If you generate an EXE file with this version of the EasyNet control but without an EasyNet license file, then any attempt to use this EXE file will display a dialog box explaining that it has been generated without license file and the control will not work correctly.

If you like EasyNet control then you can receive EasyNet license file by registering as follows:

1) EITHER in the SWREG forum on Compuserve:

License type	SWREG id	Price
Single User	2547	$ 119
3-5 Users	5487	$ 350
Unlimited User License	5488	$ 650

Then you will receive the EasyNet license file by Compuserve E-Mail and the registration fee will be billed to your Compuserve Account. This is a quick and easy way to register EasyNet.

2) EITHER by completing and sending the Order Form, along with a check for:

License type	Price (US)	Price (French)
Single User	$ 125	FF 625
3-5 Users	$ 356	FF 1780
Unlimited User License	$ 656	FF 3280

(Those prices include s&h)

to:

Patrick Lassalle
247, Avenue du Marechal Juin
92100, Boulogne
FRANCE

Then, you will receive the EasyNet license file on diskette.

Registration benefits. In return for your registration you receive these benefits:

- a **license** file giving a royalty-free right to reproduce and distribute the control file EasyNet.vbx with any application that you develop and distribute.*This license file is not for distribution.*
- full product **support** (via Compuserve) for a period of 12 months.
- the right to use EasyNet in your design environment.

License Agreement

The EasyNet custom control is not public domain or free software.

The EasyNet custom control is copyrighted, and all rights are reserved by its author: Patrick Lassalle.

Licensing:

1. shareware version

You may use the shareware version of the EasyNet custom control for up to **30 days** in your design environment for evaluation purposes only. You may copy and distribute it freely as long as all the files in the package, including the demo programs are distributed with it and no changes or additions of any kind are made to the original package.

2. registered version

Note this: Registered version = Shareware version + license file.

As a registered user, you can use the EasyNet custom control in your design environment and you have a royalty-free right to distribute executables that use EasyNet as a runtime component. Only registered users can distribute executables using the EasyNet custom control.

You may copy the software to facilitate your use of it on as many computers as there are licensed users specified in the **EasyNet.lic** file. Making copies for any other purpose violates international copyright laws. In particular, you are prohibited from distributing a registered version of the EasyNet custom control, except as a runtime component of one of your applications.

The **EasyNet.lic** file allows you to compile your applications with the EasyNet custom control. YOU ARE NOT ALLOWED TO DISTRIBUTE EASYNET.LIC FILE.

Disclaimer of Warranty:

THIS SOFTWARE AND THE ACCOMPANYING FILES ARE SOLD "AS IS" WITHOUT WARRANTY OF ANY KIND EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
Good data processing procedure dictates that any program be thoroughly tested with non-critical data before relying on it.
The user must assume the entire risk of using the program.

Your use of this product indicates that you have read and agreed to these terms.

EasyNet Order Form (Select "Print Topic" from the File menu to print this order).

Date of order: ____________

SHIPPING ADDRESS

Name	___________________________________
Company	___________________________________
Address	___________________________________
	___________________________________
	___________________________________
	___________________________________

Phone ______________ **FAX** ______________

PAYMENT ADRESS: **Patrick Lassalle**
247 , Avenue du Marechal Juin
92100, Boulogne
FRANCE

Please send me an EasyNet Custom Control diskette:

Single User License..............US $ 125	(or FF 625)	**x**	_____
3-5 Users License..................US $ 356	(or FF 1780)	**x**	_____
Unlimited User License.........US $ 656	(or FF 3280)	**x**	_____

TOTAL _____

The diskette contains the EasyNet license file and the EasyNet package in a zip file.
All payment must be by check in U.S. funds or French funds. .
Please make the check payable to Patrick Lassalle.
Prices and terms subject to change without notice.

Installation

Demonstration version:The files **easynet.vbx** and **easynet.hlp** should be copied in your WINDOWS\SYSTEM directory.

Registered version:The files **easynet.vbx, easynet.hlp** and **easynet.lic** should be copied in your WINDOWS\SYSTEM directory.

Distribution note: When you create and distribute applications that use the EasyNet control you should install the file **easynet.vbx** in the customer's Microsoft Windows \SYSTEM subdirectory. The Visual Basic Setup Kit included with the Professional VB product provides tools to help you write setup programs that install you applications correctly.

You are not allowed to distribute ***easynet.lic*** *file with any application that you distribute.*

Support

EasyNet support can be obtained

- via Compuserve. Our address is **100325,725**
- at the address indicated in Registration

Thanks in advance for your feedbacks or questions!

Acknowledgments

Many people have helped make EasyNet what it is, but in particular I'd like to thank the following individuals:

- Gils Gayraud for making good suggestions and his amazing ability to find bugs.
- Michel Lassalle for extensive help testing EasyNet.
- Jeff Simms (author of VBCTL3D.VBX) for its help about license file management.