



## **EASYNET 1.2**

**Copyright © 1994 by Patrick Lassalle.**

**ALL RIGHTS RESERVED**

---

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

[Quick Start](#)

[Overview](#)

[Why use EasyNet?](#)

**[Properties](#)**

**[Events](#)**

[Installation](#)

[Registration](#)

[Order Form](#)

[License](#)

(\*) 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 this is the demonstration version, 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...
- ...Well, it is very easy, isn't it?



## Why use EasyNet?

---

EasyNet control is useful in following applications:

- **workflow** diagrams
- **data base** diagrams
- organizational **flowcharts**
- communication **networks**
- **state transitions** diagrams
- ... and all application that needs to manage a network diagram.

## 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 programmatically. 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](#)

[Data Association](#)

[Navigation](#)

[Capabilities](#)

[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.

**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 bitmap or an icon to each node.
- Shape property allows to specify a shape for a node (ellipse or rectangle)
- 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.
- 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.
- Transparent property specifies if a node is transparent 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
```

## Data Association

---

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

- Text property associates a string. The EasyNet control maintains the memory for the strings associated to items.
- 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.
- Picture property associates a picture (bitmap or icon). The EasyNet control does not maintain the memory for that picture.



## 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 cannot 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 links of the current node
        Net1.LoopAction = 3

        ' Get count of selected nodes
        nblink = Net1.LoopCount

        ' For each link of the current node...
        For j = 1 To nblink
            Net1.Item = Net1.LoopItem(j - 1)
            '... if this link is an "out" link changes its color
            If Net1.Org = Node(i) And Net1.Oriented = True Then
                Net1.DrawColor = RGB(255, 0, 0)
            End If
        Next j
    Next i
End Sub
```

```
    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

MultiSel

ReadOnly

ScrollBars

xGrid

yGrid

## Limits

---

For one EasyNet control:

- the maximum number of nodes is **120**.
- the maximum number of links is **120**.
- the maximum length of an item text is **50** characters.
- the maximum number of link points is **8**. (therefore, the maximum number of link segments is **9**).

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

## 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.

<u>CanDrawNode</u>	<u>CanDrawLink</u>	<u>CanMoveNode</u>	<u>CanSizeNode</u>	
<u>CanStretchLink</u>	<u>CanMultiLink</u>	CtlName	BackColor	
BorderStyle	Caption	Data	Dst	
DragIcon	DragMode	<u>DrawColor</u>	<u>DrawStyle</u>	
<u>DrawWidth</u>	<u>EditAction</u>	Enabled	<u>FillColor</u>	
FontBold	FontItalic	FontName	FontSize	
FontStrike	FontUnder	Height	HelpContextId	
Hwnd	Index	<u>IsLink</u>	<u>Item</u>	
Left	<u>LoopAction</u>	<u>LoopCount</u>	<u>LoopItem</u>	
MousePointer	<u>MultiSel</u>	<u>Oriented</u>	<u>Org</u>	
Parent	<u>Picture</u>	<u>PointCount</u>	<u>ScrollBars</u>	<u>PointX</u>
<u>PointY</u>	<u>PointedArea</u>	<u>ReadOnly</u>	TabIndex	
<u>SelectMode</u>	<u>Shape</u>	<u>Sleeping</u>	Top	
TabStop	Tag	<u>Text</u>	Width	
<u>Transparent</u>	<u>Type</u>	Visible	<u>xScroll</u>	
<u>X1</u>	<u>X2</u>	<u>xGrid</u>	<u>yScroll</u>	
<u>Y1</u>	<u>Y2</u>	<u>yGrid</u>		

## 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
DbIClick	DragDrop	DragOver	<u>ErrSpace</u>
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp
<u>SelChange</u>	<u>StartLink</u>	<u>StartMove</u>	<u>StartResize</u>
<u>StartStretch</u>			

## 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	Colors specified with the system color constants from 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	Colors specified with the system color constants
from	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).

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

By default, Shape is set to 0 (ellipse)

### 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](#)

## 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](#)

## **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](#)

## **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](#)

## 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:

<b>Setting</b>	<b>Description</b>
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 programmatically. 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

### Data Type

Integer (enumerated)

### Remarks

1. This property is to be used in conjunction 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 cannot 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](#)

## 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 **8**.

### 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
```

## 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 maximum length of an item text is **50** characters. The EasyNet control maintains the memory for the strings associated to items.

Not available at design time.

### Usage

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

### Data Type

String

### Remarks

The string is truncated if it is longer than the maximum number of bytes allowed: **50**.

### 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) the	Specifies a picture. You can also set this property using the LoadPicture function on a bitmap or an icon.

### Data Type

Integer

### See Also

[Data Association](#)



## 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 that 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.

### 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](#)

## 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)

### 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

## **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

ThePointedArea 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.

## **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.

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 will never occur in the current version of EasyNet control. In fact, this event will be used in next versions.

## StartLink Event

---

### Description

Occurs when the user clicks the little black square at the middle of a selected node.

### Syntax

**Sub** *NET\_StartLink* ()

### Remarks

This event is obsolete and will disappear in next versions. You would rather use PointedArea property combined with mouse events.

## **StartStretch** Event

---

### **Description**

Occurs when the user clicks over a little black square of a selected link.

### **Syntax**

**Sub** *NET\_StartStretch* ()

### **Remarks**

This event is obsolete and will disappear in next versions. You would rather use [PointedArea](#) property combined with mouse events.

## StartResize Event

---

### Description

Occurs when the user clicks over one of the height little black squares around a selected node.

### Syntax

**Sub** *NET\_StartResize* (*direction* As Integer)

### Settings

The *direction* parameter settings are:

Setting	Description
0	Size NW-SE
1	Size N-S
2	Size NE-SW
3	Size W-E

### Remarks

This event is obsolete and will disappear in next versions. You would rather use PointedArea property combined with mouse events.

## **StartMove** Event

---

### **Description**

Occurs when the user clicks over a node but not in one of the 9 black handles.

### **Syntax**

**Sub** *NET\_StartMove* ()

### **Remarks**

This event is obsolete and will disappear in next versions. You would rather use [PointedArea](#) property combined with mouse events.

## 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 fail to load.

If you like EasyNet control then you can receive a full version by registering as follows:

- 1)** Either in the SWREG forum on CompuServe. The fee is **\$99** and the Registration ID number is **2547**. Then you will receive EasyNet by CompuServe E-Mail and the registration fee will be billed to your CompuServe Account.
- 2)** Either by completing and sending the Order Form, along with a check for **FF 520** (French currency) or **\$105** (includes cost of conversion) to:

**Patrick Lassalle  
2 , rue Gutenberg  
92100, Boulogne  
FRANCE**

Then, you will receive the EasyNet control on diskette.

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

- the latest version of the EasyNet control.
- full product **support** (via CompuServe) and free product upgrades for a period of 12 months.
- 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.*

## License

---

The EasyNet control is not public domain or free software.

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

You are licensed to use this software on a restricted number of computers. 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.

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

**Disclaimer:** This software is 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. The authors assume no liability for any alleged or actual damages arising from the use of this software. (Some states do not allow the exclusion of implied warranties, so the exclusion may not apply to you.)

**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 ADDRESS:**     **Patrick Lassalle**  
                                  **2, rue Gutenberg**  
                                  **92100, Boulogne**  
                                  **FRANCE**

**Please send me:**

**EasyNet Custom Control Diskette:.....\$105 (or FF 520) x \_\_\_\_\_**

**TOTAL** \_\_\_\_\_

All payment must be by check in U.S. funds or French funds.  
The US price includes our cost of conversion.  
Sorry, at this time we cannot accept credit cards or "bill-me" orders.  
Please make the check payable to Patrick Lassalle.

## 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.*

