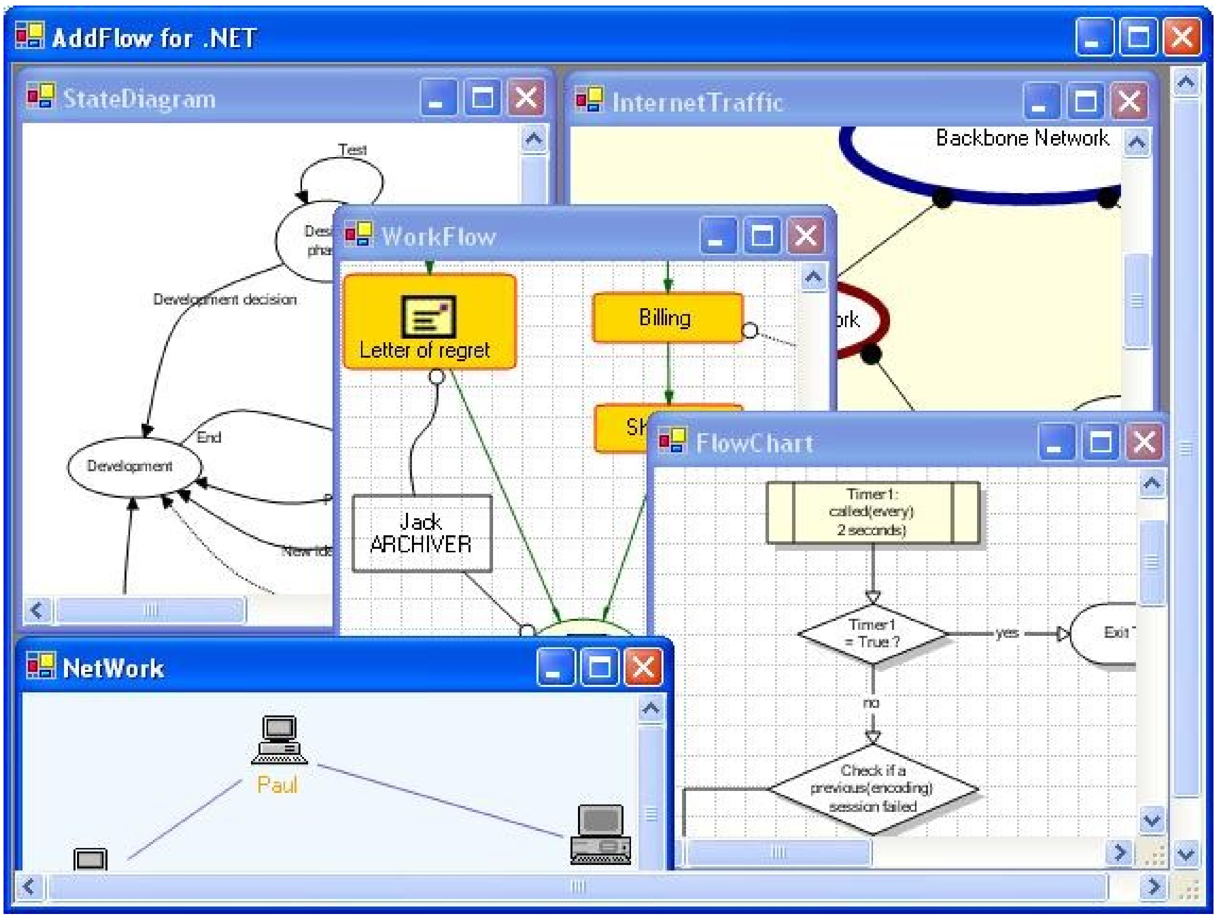
# AddFlow for WinForms V2.3.2 Tutorial



**April 2011**

**Lassalle Technologies**

**http://www.lassalle.com**

**CONTENTS**

# 1 Introduction

AddFlow for WinForms is a general purpose Flowcharting/Diagramming .NET Windows form control, which lets you quickly build flowchart-enabled .NET applications.

AddFlow for WinForms allows the creation and the manipulation of two-dimensional diagrams (a.k.a graphs). An AddFlow diagram is a set of objects called nodes (also called vertices or entities) that can be linked each other with links (also called edges, arcs or relations). These diagrams can be created programmatically or interactively.

Each time you need to graphically display interactive diagrams, you should consider using AddFlow, a royalty-free control that offers unique support to create diagrams interactively or programmatically: workflow diagrams, database diagrams, communication networks, organizational charts, process flows, state transitions diagrams, CTI applications, CRM (Customer Relationship Management), expert systems, graph theory, quality control diagrams, ...

**Purpose of this tutorial**

This tutorial provides informationon:

- creating diagrams programmatically, using the AddFlow control and classes
- creating diagrams interactively
- installing AddFlow for WinForms
- licensing

**Who should use this tutorial?**

This guide is intended for application programmers using the .NET platform to build Windows Forms applications.

**Samples**

AddFlow for WinForms is installed with many samples (see the Samples paragraph for a description of each sample).

- Each sample is provided in a C# version and in a VB version.
- The more important sample is **afEdit** since it allows manipulating the properties of AddFlow and of each node or link object.
- The **DemoLayout** sample is also interesting since it allows working with the Graph Layout components HFlow, OFlow, SFlow, SPFlow and TFlow.
- And finally the new **Tables2** sample demonstrates many of the version 2 enhancements.

# 2 Version enhancements

## 2.1 Version 2.3.2 enhancements

AddFlow for .NET is renamed AddFlow for WinForms.

## 2.2 Version 2.3.1 enhancements

All the dlls and samples have been created using **Visual Studio 2010** and the **.NET Framework 4.0 Client profile**.

However, we provide also a version compiled with **Visual Studio 2008** and **.NET Framework 3.5**.

## 2.3 Version 2.3 enhancements

- The full product has been regenerated with **Visual Studio 2010** on a **64 bits** machine.
- Implementation of the **Bentley-Ottmann algorithm** to quickly find the link intersections.
- **Pan** feature (See the MouseAction property. Demonstrated in the afEdit sample)
- **DisplayDragFrame** property (allowing hiding the drag frame of a node)
- **ScrollPositionChange** event.
- The DefNode and DefLink types disappear!The DefNodeProp and DefLinkProp properties are now of type Node and Link instead of DefNode and DefLink.

Important information: The license file(s) must be recreated using your license key(s) and the LicenseManager.exe tool provided with AddFlow.

## 2.4 Version 2 enhancements

AddFlow for WinForms version 2 has been created with Microsoft **Visual Studio 2005**. This is in fact the major enhancement!

AddFlow for WinForms version 2 is compatible with the previous versions of AddFlow for WinForms on a source-code level (with a few exceptions explained in paragraph 2.7)

Most of the changes consist of new features that we will describe briefly in this paragraph.

### 2.4.1 Serialization

AddFlow for WinForms version 2 is now supporting the **IXmlSerializable** interface. To save a diagram (in a file or in a stream), just call the **WriteXml** method. To load it, call the **ReadXml** method.

This XML serialization provides the same kind of files as those produced by for the old XMLFlow.dll component. And it is compatible with XMLFlow. However, it is quicker and more flexible (serialization events).

### 2.4.2 New events

AddFlow for WinForms version 2 offers new events for serialization or selection:

- **SelectionChange** Event
- **BeforeReadXMLNode** Event (Version 2.1)
- **BeforeReadXMLLink** Event (Version 2.1)
- **BeforeWriteXMLNode** Event (Version 2.1)
- **BeforeWriteXMLLink** Event (Version 2.1)
- **ReadXMLLinkExtraData** Event
- **ReadXMLNodeExtraData** Event
- **WriteXMLLinkExtraData** Event
- **WriteXMLNodeExtraData** Event

Moreover, some old events can be used programmatically if the property **InteractiveEventsOnly** is false. It is the case for the following events:

- AfterAddNode
- AfterAddLink
- AfterRemoveNode
- AfterRemoveLink
- BeforeAddNode
- BeforeAddLink
- BeforeRemoveNode
- BeforeRemoveLink

### 2.4.3 Property bag

The property bag allows extending the functionality of nodes and link by adding new properties. This feature is very easy to use. For more information, see the paragraph Property Bag.

### 2.4.4 Parent-Child relationship

It is now possible to set a node as the child of another node or a link. This allows placing nodes inside another node and the placement can be very easy with the **Dock** property. This allows also defining labels for a node or for a link. If you move the node or stretch the link, its labels follow it.

Using this feature, you don't need to use a rigid and hidden link as with the previous versions. For more information, see the paragraph Parent-Child relationship.

### 2.4.5 Undo/Redo enhancements

The undo/redo can be customized. For that, you have to create a custom Task class by deriving the Task class and then you can insert it in the undo list with the **SubmitTask** method.

Another interesting method is the **AddToLastAction** method. For instance, it allows grouping some actions with the last recorded action.

### 2.4.6 Minor enhancements

There are also other minor enhancements like:

- the **ConnectionStyleDst** and **ConnectionStyleOrg** properties
- the way a Bezier link is highlighted when selected.
- the **PageGrid** property which returns/sets a Grid object allowing to set the properties of the grid used to display printing pages.
- the possibility for a link to have a Shadow too.
- links drawn with double lines (**DoubleLine** property of the Line class)
- a new link style: Database. Such a link has 3 segments and the first and the third segments are horizontal.

### 2.4.7 Compatibility

AddFlow for WinForms version 2 is compatible with the previous versions of AddFlow for WinForms on a source-code level except for the following minor points:

- The Scroll event provided in the previous version is now replaced by the new Scroll event of the ScrollableControl class of the .NET 2.0 framework.
- The default PageUnit property value is now GraphicsUnit.Pixel (instead of GraphicsUnit.Point).

# 3 Getting Started

## 3.1 Installation

The AddFlow for WinForms installation package is a Windows Installer file. It is the same file for the evaluation version and the full version. However, when you install it, you install the evaluation version. As explained in the Licensing section if you purchase the product, you will receive a license key allowing turning the evaluation version into the full version.

In the AddFlow for WinForms installation folder, it creates 3 subdirectories: Bin, Doc and Src:

- The **Bin** subdirectory contains the assemblies: DLLs, sample executables, LicenseManager program. The list of the DLLs is given in the following AddFlow extensions parapagraph.
- The **Doc** subdirectory contains the help file, the tutorial, the readme file and the license agreement.
- The **Src** subdirectory contains the source code (C# **and** VB) of the samples that demonstrate AddFlow for WinForms (You can find the list of the samples and a short description of each in the Samples paragraph). It contains also the C# source code of some AddFlow extensions (PrnFlow, XMLFlow, GraphAlgo and DlgFlow).

**Remarks**

- Note that the installation package does not contain the .NET Framework which must be already installed on the target machine.
- The installation procedure does **not** install the AddFlow for WinForms assemblies into the Global Assembly Cache (GAC).
- AddFlow is installed with several extensions, as described in the following section.

## 3.2 AddFlow extensions

Following is the list of the AddFlow for WinForms extensions, including AddFlow itself. All these assemblies are installed with AddFlow for WinForms. All these assemblies are written in C#.

| **Class** | **Assembly** | **Type** | **Description** | **To be purchased?** | **Source code provided?** | **Support provided?** |
|---|---|---|---|---|---|---|
| AddFlow | Lassalle.Flow.dll | Control | AddFlow for WinForms | Yes | No | Yes |
| PrnFlow | Lassalle.PrnFlow.dll | Component | print and preview of AddFlow diagrams | No (1) | Yes (3) | No |
| XMLFlow | Lassalle.XMLFlow.dll | Class library | load/save a diagram or a portion of it in a XML stream | No (1) | Yes (3) | No |
| DlgFlow | Lassalle.DlgFlow.dll | Component | provides dialog boxes to change the properties of items | No (1) | Yes (3) | No |
| HFlow | Lassalle.Flow.Layout.Hierarchic.dll | Component | hierarchic graph layout algorithm | Yes (2) | No | Yes |
| OFlow | Lassalle.Flow.Layout.Orthogonal.dll | Component | orthogonal graph layout algorithm | Yes (2) | No | Yes |
| SFlow | Lassalle.Flow.Layout.Symmetric.dll | Component | force directed (symmetric) graph layout algorithm | Yes (2) | No | Yes |
| SPFlow | Lassalle.Flow.Layout.SP.dll | Component | Series Parallel graph layout algorithm | Yes (2) | No | Yes |
| TFlow | Lassalle.Flow.Layout.Tree.dll | Component | tree graph layout algorithm | Yes (2) | No | Yes |
| SVGFlow | Lassalle.Flow.SVG.dll | Class library | export a diagram in SVG format | No | No | Yes |
| RouteFlow | Lassalle.Flow.Router.dll | Component | link auto routing | No | No | Yes |

(1) XMLFlow, PrnFlow and DlgFlow can be used freely.
(2) Included in the **LayoutFlow** license (HFlow, SFlow, SPFlow, TFlow and OFlow cannot be purchased separately).
(3) The source code of XMLFlow, PrnFlow and DlgFlow is installed with AddFlow.

## 3.3 Samples

Following is the list of the samples provided with AddFlow. The source code (C# and VB version) of each sample is also installed.

| | |
|---|---|
| afEdit | A small diagram editor. It allows manipulating each property of each node and link. |
| CustomShape | Shows how to create nodes with custom shapes. |
| Demo | Shows different kind of diagrams created with AddFlow. |
| DemoLayout | Demonstrates the possibilities of the Graph Layout controls. |
| DeriveNode | Shows how to add custom data to a node by deriving the Node class. It shows also how to use the serialization events to save and load such a diagram. |
| DragDrop | Shows how to drag a TreeView item and drop it on an AddFlow control |
| GraphTest | Shows how use the GraphAlgo dll.extension. |
| Lists | Shows how to create nodes with a ListView control inside. |
| Navig | Shows how to use collections to navigate in a diagram. It shows for instance how to change the color of all the links of a node or all the connected nodes of a node. |
| OwnerDraw | Shows how to use the OwnerDraw properties. |
| Pins | Shows how to create complex nodes, using the Parent property. |
| Propertybag | Shows how to add custom data to a node by using the property bag. It shows also how to use the serialization events to save and load such a diagram. |
| RouteLink | Shows how to use the RouteFlow extension for link autorouting. |
| Shapes | Shows the predefined shapes that can be used for nodes. |
| Stress | Allows creating many nodes and links randomly. |
| Tables2 | Shows how to use AddFlow to create database diagrams. It demonstrates many of the version 2 enhancements. |
| TreeEdit | Shows how to use AddFlow and TFlow to create and draw trees. |

## 3.4 Licensing

### 3.4.1 3.4.1 Type of licenses

Notice that you can purchase either:

• an AddFlow for WinForms **Standard** license. This license does not include LayoutFlow. If you try to execute a graph layout algorithm, you will face sometimes a nag screen.

• an AddFlow for WinForms **Professional** license. This license includes LayoutFlow. You can execute the graph layout algorithms provided by LayoutFlow without any restriction. This licence includes two license keys: one for for AddFlow and one for LayoutFlow

### 3.4.2 How it works?

**Remarks**

- You can use the evaluation version only during a period of 30 days.
- The only difference between the evaluation and the full (purchased) version of AddFlow for WinForms is that the registered version will stamp every application you compile so a nag screen (banner or dialog box) will not appear when your users run the applications.
- You do not have the right in any case to distribute the license files or divulge the license keys (as indicated in the license agreement).
- Now the graph layout components (HFlow, OFlow, SFlow, SPFlow and TFlow) are grouped in only one product: LayoutFlow. Therefore only one license key is needed for the five components.

**The evaluation version**

When you install AddFlow for WinForms, you install in fact an evaluation version of AddFlow. You install also an evaluation version of HFlow, OFlow, SFlow, SPFlow and TFlow.

If you generate ("compile") an application that uses this evaluation version of AddFlow for WinForms, then any attempt to use this application will display an evaluation label explaining that it has been generated only with an evaluation version of AddFlow.

And if you execute one of the graph layout methods provided by the LayoutFlow extensions (HFlow, OFlow, SFlow, SPFlow and TFlow), you will face sometimes a nag screen.

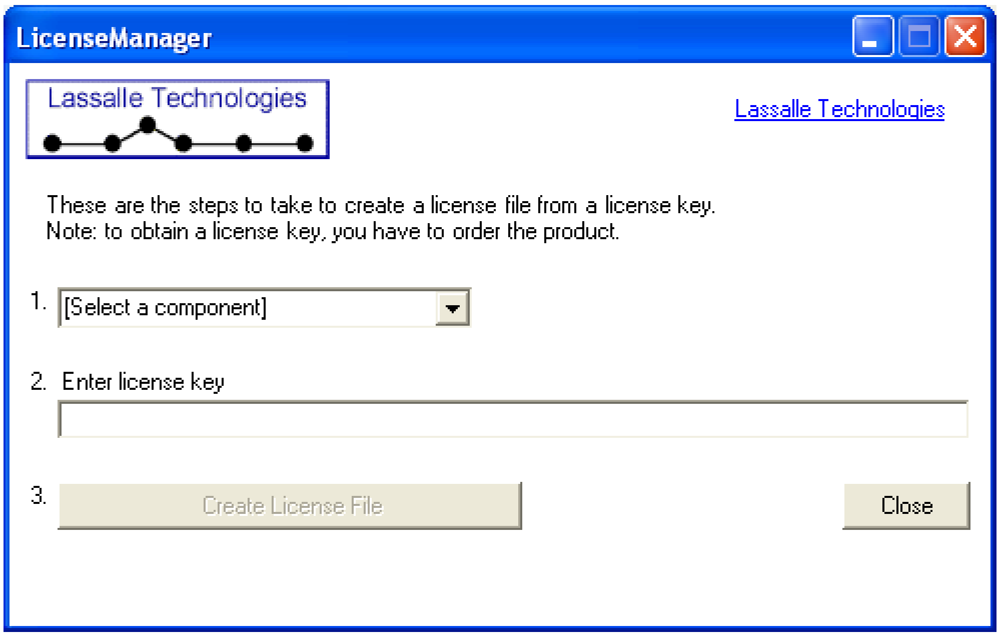
**The full version**

To get the full version of AddFlow, you have to purchase an AddFlow license. In such a case, you will receive an AddFlow license key (also called serial number or license number).

To get the full versions of HFlow, OFlow, SFlow, SPFlow and TFlow, you have to purchase a LayoutFlow license. In such a case, you will receive a LayoutFlow license key. In such a case, the four components share the same license key.

### The LicenseManager program

The LicenseManager.exe program, provided with AddFlow for WinForms, allows generating a license file from a license key.



These are the steps to take to create a license file from a license key.
Note: to obtain a license key, you have to order the product.

1. [Select a component]

2. Enter license key

3. Create License File

The license file is created in the same directory as the LicenseManager application, therefore in the bin subdirectory of the AddFlow for WinForms installation folder.

When developing your application, this license file must be in the same directory as the licensed assembly: for instance, the file Lassalle.Flow.AddFlow.LIC must be placed in the same directory as the file Lassalle.Flow.dll.

### The license files

The following table gives the names of the license files. Remember that you have to use the LicenseManager program to create these license files. The AddFlow license key will allow you creating the AddFlow license file whereas the LayoutFlow license key will allow you creating the HFlow, OFlow, SFlow, SPFlow and TFlow license files.

| **Assembly** | **License** | **License file name** |
|---|---|---|
| Lassalle.Flow.dll | AddFlow for WinForms | Lassalle.Flow.AddFlow.LIC |
| Lassalle.Flow.Layout.Hierarchic.dll | LayoutFlow for WinForms | Lassalle.Flow.Layout.Hierarchic.HFlow.LIC |
| Lassalle.Flow.Layout.Orthogonal.dll | LayoutFlow for WinForms | Lassalle.Flow.Layout.Orthogonal.OFlow.LIC |
| Lassalle.Flow.Layout.Symmetric.dll | LayoutFlow for WinForms | Lassalle.Flow.Layout.Symmetric.SFlow.LIC |
| Lassalle.Flow.Layout.SP.dll | LayoutFlow for WinForms | Lassalle.Flow.Layout.SP.SPFlow.LIC |

|  |  |  |
|---|---|---|
| Lassalle.Flow.Layout.Tree.dll | LayoutFlow for WinForms | Lassalle.Flow.Layout.Tree.TFlow.LIC |

**Example**

After having installed AddFlow for WinForms, load for instance the VB project Navig provided with AddFlow, then compile it and run it: the nag screen will appear.

Now place the AddFlow license file in the bin subdirectory of the AddFlow for WinForms installation folder (This is possible only if you have already created a license file with the LicenseManager program, therefore if you have already a license key, therefore if you have purchased the product!). Then re-compile the application and run it again: there is not any nag screen this time.

Notice the file licenses.licx in the **Navig** folder. This file is a text file that identifies which licensed classes are used in a project. There should be one licenses.licx file (as an embedded resource) for each project in a VS.NET solution. Without this file, the licensing would not work. It must contain the assembly qualified name of AddFlow.

Following is the list of assembly qualified names for AddFlow and its licensed extensions:

**The assembly qualified name of AddFlow**

```
Lassalle.Flow.AddFlow, Lassalle.Flow, Version=2.3.2.0, Culture=neutral,
PublicKeyToken=bfc5c756e54a9d2a
```

**The assembly qualified name of HFlow**

```
Lassalle.Flow.Layout.Hierarchic.HFlow, Lassalle.Flow.Layout.Hierarchic,
Version=2.3.2.0, Culture=neutral, PublicKeyToken=a5098d67a8d68ddf
```

**The assembly qualified name of OFlow**

```
Lassalle.Flow.Layout.Orthogonal.OFlow, Lassalle.Flow.Layout.Orthogonal,
Version=2.3.2.0, Culture=neutral, PublicKeyToken=b1252ad24c259c82
```

**The assembly qualified name of SFlow**

```
Lassalle.Flow.Layout.Symmetric.SFlow, Lassalle.Flow.Layout.Symmetric,
Version=2.3.2.0, Culture=neutral, PublicKeyToken=f43759eeb25c46d5
```

**The assembly qualified name of SPFlow**

```
Lassalle.Flow.Layout.SP.SPFlow, Lassalle.Flow.Layout.SP, Version=2.3.2.0,
Culture=neutral, PublicKeyToken=c744fe88085eaad0
```

**The assembly qualified name of TFlow**

```
Lassalle.Flow.Layout.Tree.TFlow, Lassalle.Flow.Layout.Tree,
Version=2.3.2.0, Culture=neutral, PublicKeyToken=577d6084e440e212
```

For controls as AddFlow for WinForms and components as HFlow for WinForms, if you use the toolbox to place them on your form, the file licenses.licx is automatically created. Otherwise, you can create it manually.

### 3.4.3 Licensing problems

If, after having obtained a license for AddFlow and having followed the instructions, you have still a nag screen, don't worry: you are not the first! Despite the fact that the licensing technology used in AddFlow is 100% standard and that it follows 100% the .NET guidelines, many customers are facing such a problem. It was also my case at the beginning. The following remarks may help you solving this licensing issue.

1) First of all, be sure that your project contains a licenses.licx file (as an embedded resource) and that this file contains the following line:

```
Lassalle.Flow.AddFlow, Lassalle.Flow, Version=2.3.2.0, Culture=neutral,
PublicKeyToken=bfc5c756e54a9d2a
```

2) License keys are only recognized when they are linked into an executable assembly (.EXE), not a library (.DLL). Therefore, if you use AddFlow in another DLL (for instance a control library), then you will have to:

- include a reference to AddFlow in the application that uses that DLL (even though this executable did not directly use the AddFlow control)
- be sure that the licenses.licx file of the application itself contains a line referencing AddFlow.

### Licensing when using AddFlow in a web page

To specify the location of the license file, you have to use a special attribute on the "**LINK**" HTML tag. The license file is a ".licenses" file. It contains the license keys for the licensed .NET classes. An example for AddFlow is below:

```
<html>
<head>
<title>AddFlow</title>
</head>
<body>
<LINK REL="licenses" HREF="lassalle.flow.html.licenses">
<OBJECT id="addflow1" height="200" width="400"
  classid=http:Lassalle.Flow.dll#Lassalle.Flow.AddFlow>
</OBJECT>
</body>
</html>
```

1) To create the .licenses file, you have to use the **lc.exe** tool (on my computer, I can find it at: C:\Program Files\Microsoft.NET\FrameworkSDK\Bin).

For example, you can create a .bat file (named for instance build.bat) containing the following command:

```
lc /target:Lassalle.Flow.html /complist:licenses.licx /i:Lassalle.Flow.dll
```

To create the binary .licenses file, open a command window and run “build.bat”. The licenses.licx file is the same as previously described.

2) An IIS virtual directory needs to be setup to run the HTML page from.

## 3.5 Customize Visual Studio for WinForms

You will probably want to customize the toolbox of Visual Studio for WinForms with AddFlow for WinForms:

- Start up Visual Studio.NET
- View the toolbox if it is not already visible (**View\Toolbox** menu item)
- Then open the tab where you wish to contain the AddFlow for WinForms control (or create a new tab if you wish).
- Right-click this tab to show the context menu and choose the **Add/Remove Items…** menu item. The **Customize Toolbox** dialog will appear.
- Select the **.NET Framework Components** tab.
- If you do not see AddFlow for WinForms in the list, click the **Browse…** button to open the **Lassalle.Flow.dll** assembly in the Bin subdirectory of the AddFlow for WinForms installation folder. Select it.
- Click **OK**. The AddFlow for WinForms should appear in the toolbox, allowing you to drag it onto a form. Then, you can customize the behavior of AddFlow with the **Properties** window.

Then, if you wish, you may proceed the same way to add the AddFlow extension components: HFlow, SFlow, SPFlow, TFlow, PrnFlow, XMLFlow, DlgFlow, SVGFlow and RouteFlow.

# 4 Interactive creation of a diagram

## 4.1 Overview

The interactive creation of diagrams is mouse-based. It includes:

- the creation of nodes and links (including reflexive links)
- the selection of nodes and links (including multi-selection)
- the resizing of nodes
- the moving of nodes
- the stretching of links (the possibility to add or remove segments in a link)
- the possibility to change the origin or the destination of a link

It supports also the scrolling of diagrams, the node in-place editing and the use of grids.

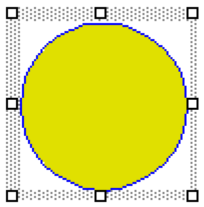
Moreover, many properties allow customizing the interactive behavior of an AddFlow control. For instance, you can prevent the user to create reflexive links with the **CanReflexLink** property or to move nodes with the **CanMoveNode** properties.

And a set of methods and properties allow implementing a powerful Undo/Redo feature.

## 4.2 Create a diagram interactively

### 4.2.1 Draw a node

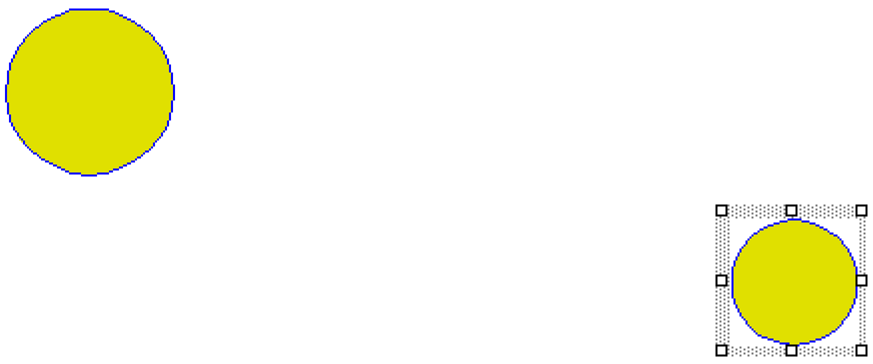
Bring the mouse cursor into the control, press the left button, move the mouse and release the left button. You have created an elliptic node. This node is selected: that's why 8 handles (little squares) and a drag frame are displayed.



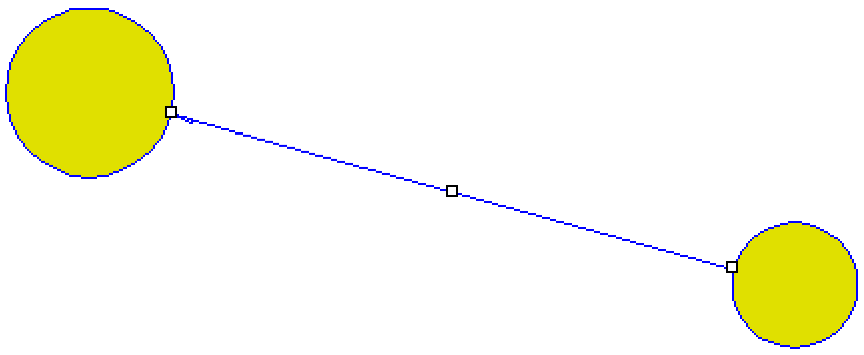
The 8 handles allow **resizing the node**. If you want to **move the node,** you bring the mouse cursor into the drag frame, press the left button, move the mouse and release the left button.

### 4.2.2 Draw a link

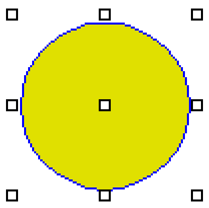
Draw a second node.



Then bring the mouse cursor inside 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: 3 handles are displayed in the link.

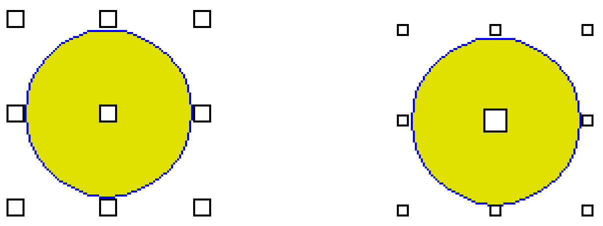


**Note**. In the ActiveX version, the selected node had 9 handles and no drag frame. The handle at the center of the node is used to draw a link. Such a behavior is also possible in the .NET version, if the **LinkCreationMode** property of AddFlow is set to **LinkCreationMode.MiddleHandle.** In such a case, the selected node would be as in the following diagram:



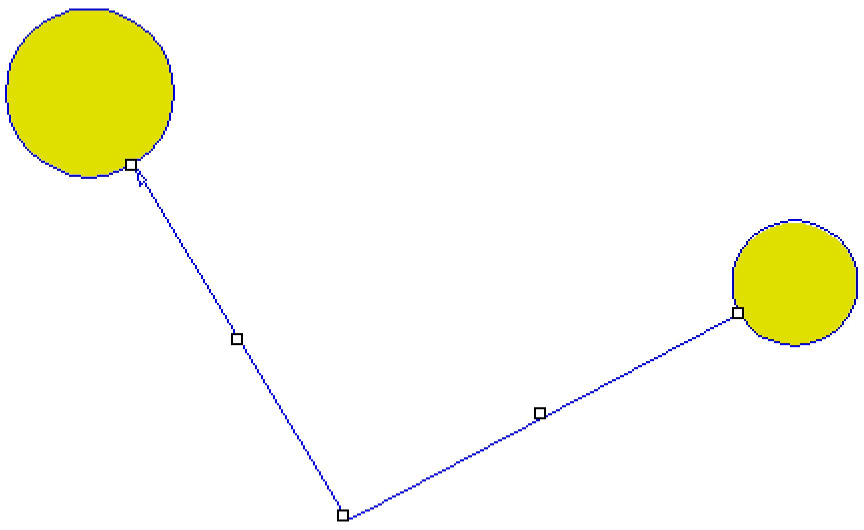
To draw a link, bring the mouse cursor into the handle at the center of the selected node, press the left button, move the mouse towards the other node. When the mouse cursor is into the other node, release the left button.

You can change the size of the handles, using the **SelectionHandleSize** and the **LinkHandleSize** properties. For instance, you can have the following representations of the selected node:

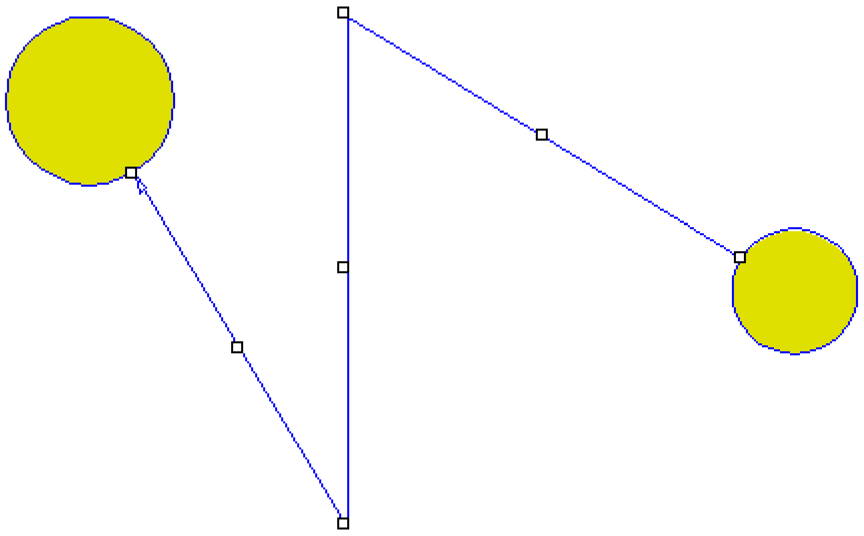


### 4.2.3 Stretch a link

Bring the mouse cursor into the link handle in the middle of the link, press the left button, move the mouse and release the left button. You have created a new link segment. It has now 5 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).

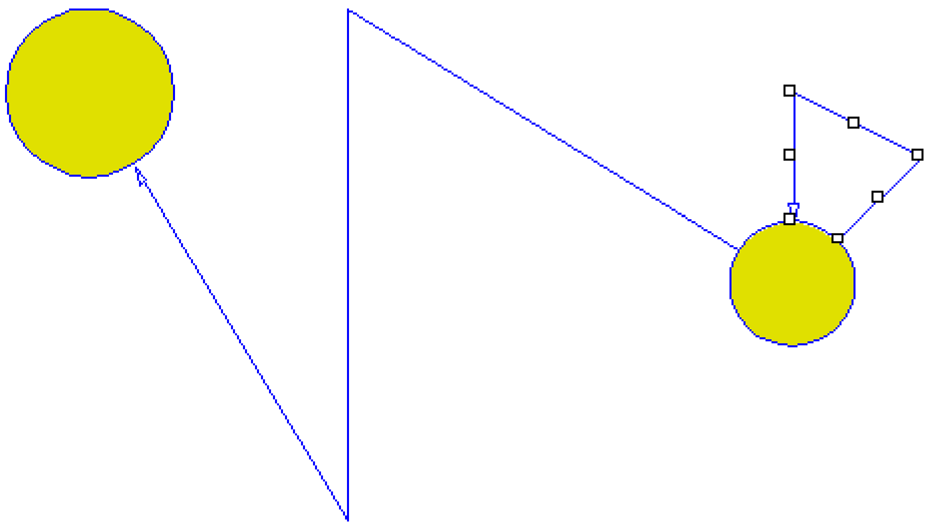


Create another segment



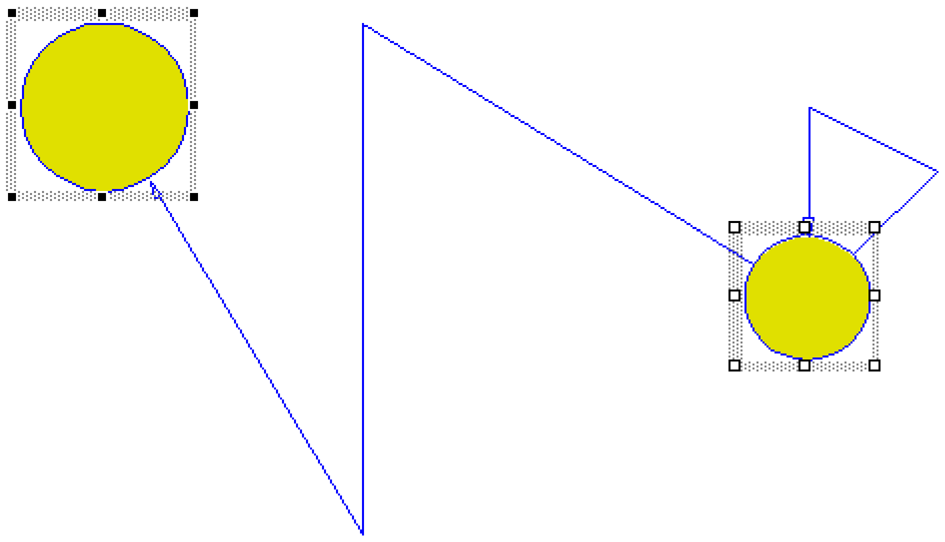
### 4.2.4 Draw a reflexive link

Select a node by clicking on it. Then bring the mouse cursor inside the selected node. Press the left button, move the mouse outside the selected node, then move it inside the selected node again, then release the left button. You have created a reflexive link, i.e. a link whose origin and destination are the same.

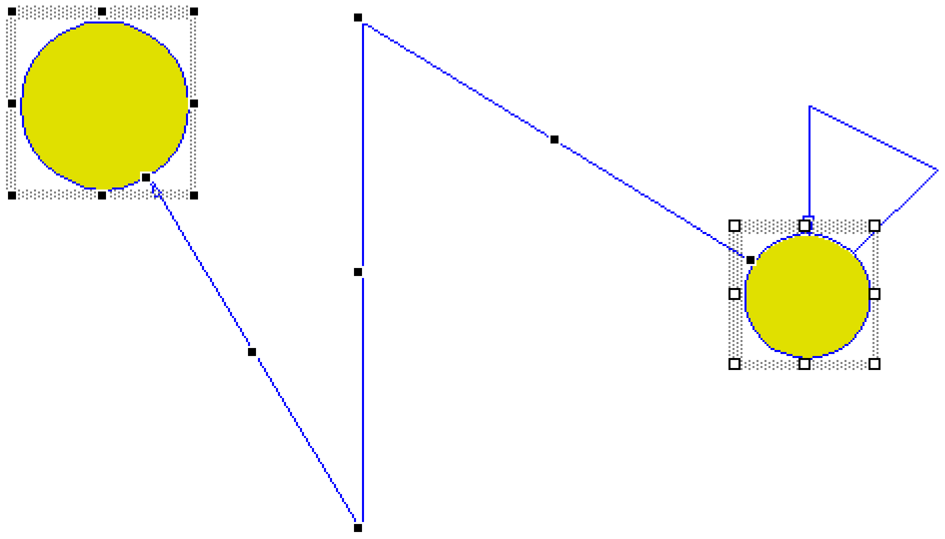


### 4.2.5 Multiselection

You can select several nodes by clicking them with the mouse and simultaneously pressing the shift or control key.



You can also select links or nodes and links.

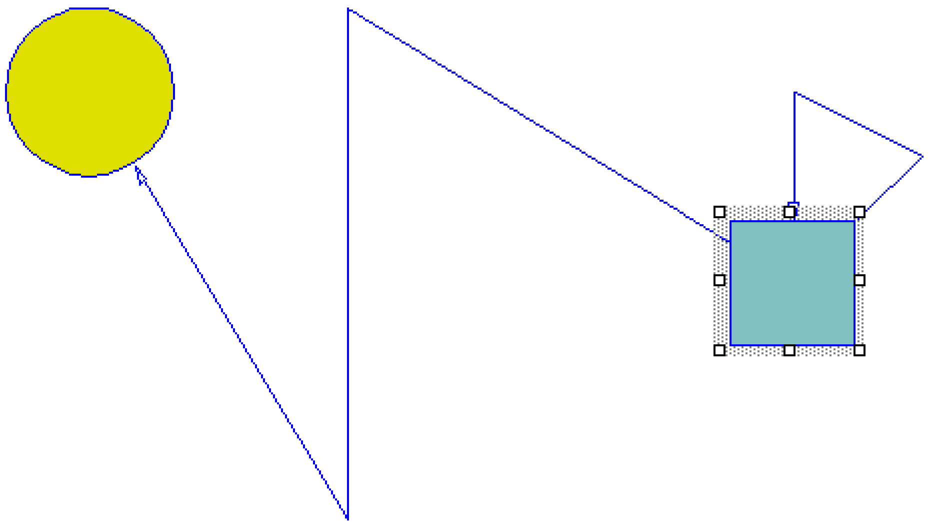


There is another way to perform multiselection, using the **MouseAction** property and assigning it the **MouseAction.Selection** value. Then you can select several nodes and links: you bring the mouse cursor into the AddFlow control, press the left button, move the mouse and release the left button. All nodes or links partly inside the selection rectangle are selected. Then you can unselect some nodes by clicking them with the mouse and simultaneously pressing the shift or control key. You can select them again by using the same method.
You can use the Shift Key in conjunction with a selection rectangle. For instance, if the MouseAction property is set to MouseAction.Selection, you can draw a selection rectangle with a mouse, which cause items inside this rectangle to be selected. Then, if you press the Shift key, you can select additional items.

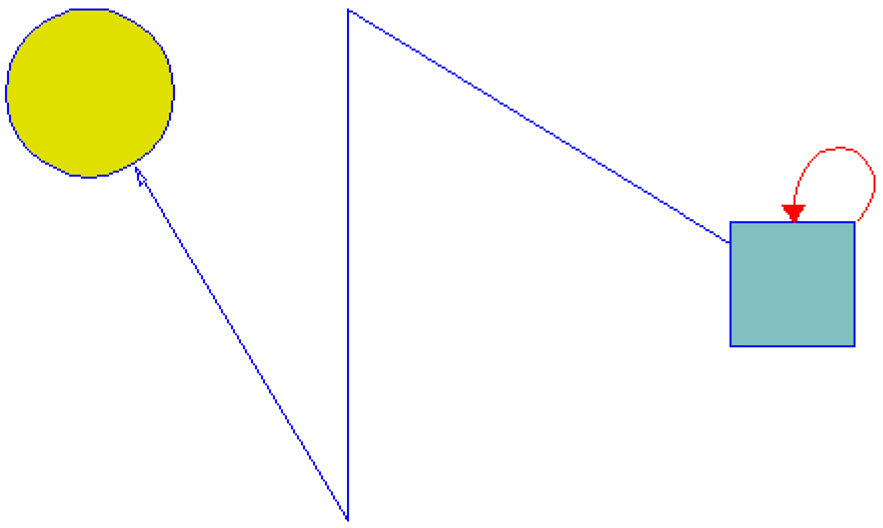
### 4.2.6 Change properties of a node or a link

Interactively, without adding any code, you can change the position and the size of a node (and also its text as described later). You can add segments to a link or remove them. To change the other properties (shape, styles, colors, behaviors, etc) of a node or a link, you have to write some code. For instance, you can add code that displays a property page when the user right clicks on a node or a link. This is what is done in the afEdit sample provided with AddFlow .NET. The code that manages the property pages is encapsulated in a small dll: **DlgFlow.DLL**.

If, using the afEdit sample, you right click on the second node, its property page will appear and you can change some of its properties, for instance its shape and its filling color:

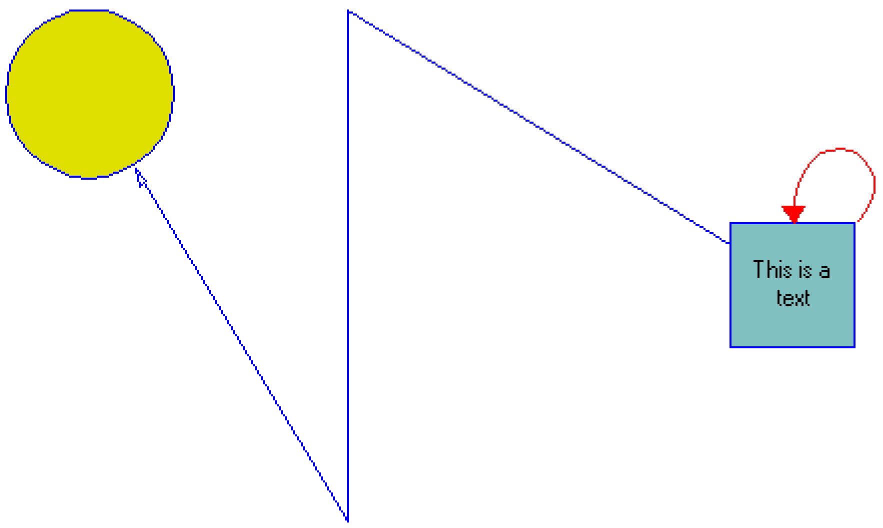


You can proceed the same way with, for instance, the reflexive link and change its style, its color and its arrow head:



### 4.2.7 Add a text to a node

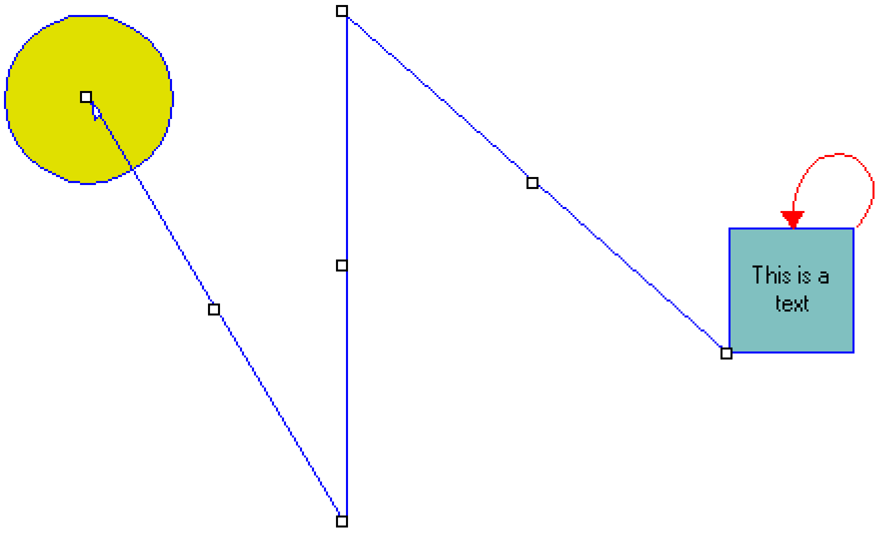
Click for instance on the blue node. This will select it. Click another time. An edit box is displayed inside the node, allowing you to enter a text.



### 4.2.8 Adjust the link origin and destination points

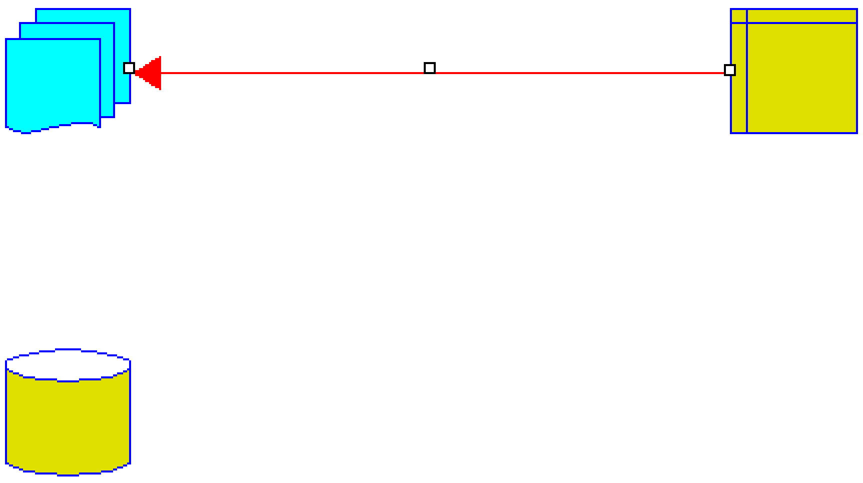
In the previous example, you cannot adjust the extremities of the links. For instance, if you select the blue link and bring the mouse cursor into the last (or the first) handle of the blue link, press the left button, move the mouse in another place then relinquish the mouse button, the link springs back again, retrieving its initial position.

However, you can change this behavior by setting the **AdjustDst** and **AdjustOrg** properties to true. In such a case, if you bring the mouse cursor, for instance, into the last handle of the link, press the left button, move the mouse and release it, you'll see that you have defined a new destination position for the link. If you move the destination node, the new link destination position keeps on following the node.



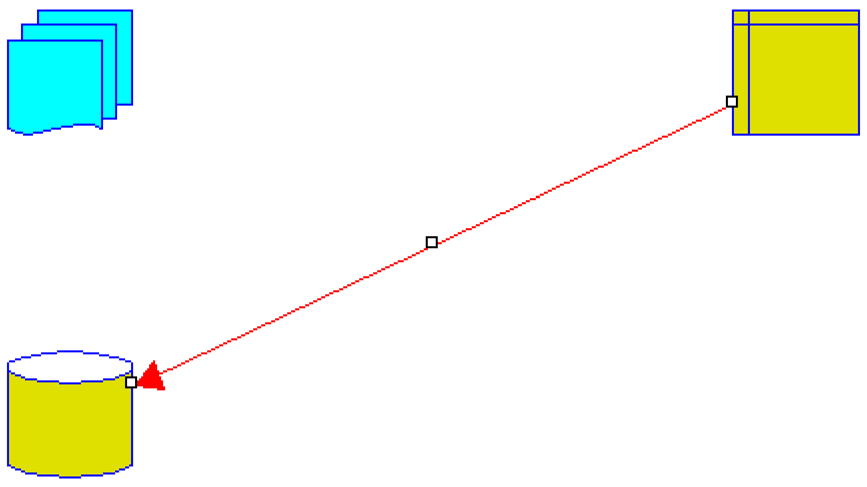
### 4.2.9 Change the destination or the origin node of a link

You can change interactively the destination or the origin of a link. For instance, consider the following diagram.



You can adjust the last point of the red link. Move it from node 1 to node 2: bring the mouse cursor into the third link handle (near the arrow head), press the left button, move the mouse

until the node 2 and release the left button.
Then, the new destination of the link will be the node 2 (If you move the node 2, the link will follow it).



# 5 Programmatic creation of a diagram

## 5.1 Overview

In this chapter we will focus on how to create a diagram programmatically.

The AddFlow library is a .NET class library containing a set of classes for creating interactive diagrams very easily.

The main class is the **AddFlow** class that derives from the .NET UserControl class. An AddFlow diagram contains two kinds of objects, **Node** objects and **Link** objects, which derive from the **Item** class. A Link object allows to link two nodes. It is a line that leaves the origin node and comes to the destination node. A link cannot exist without its origin and destination nodes. If one of these two nodes is removed, the link is also removed.

Other classes are:

- either collection classes:
    - **Items**: the collection of all items of the diagram
    - **SelectedItems**: collection of all selected items of the diagram
    - **Nodes**: collection of nodes of the diagram
    - **OutLinks**: collection of all links leaving a node
    - **InLinks**: collection of all links coming to a node
    - **Links**: collection of all links (in and out) of a node
    - **Images**: collection of all images used in the diagram

- either classes dedicated to manage some complex properties:
  1.
    - **Grid**: manages the grid of the AddFlow control
    - **Zoom**: manages the AddFlow zooming
    - **Shape**: manages the shape of the node
    - **Shadow**: manages the shadow of the node
    - **Arrow**: manages the arrows of the link
    - **Line**: manages the segments of the link
    - **FlowImage**: manages the images associated to nodes.

For detailed information about the types, classes and interfaces used in AddFlow, see the **Lassalle.Flow** namespace in the AddFlow help file.

## 5.2 Diagram creation

### 5.2.1 Our first program

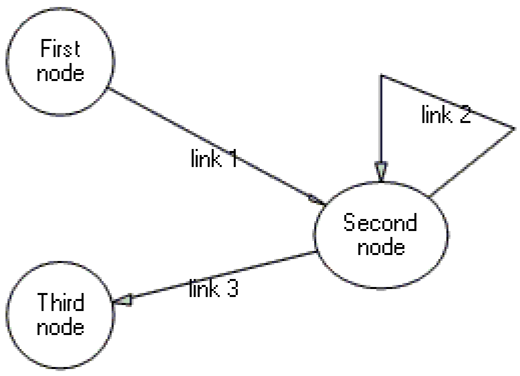
Using Visual Studio .NET, select the menu item **File | Add Project | New Project**. In the Add New Project dialog box, select a project type of **Visual C# Projects**, then choose **Empty Project**. Then, right-click the project name in Solution Explorer and select **Add New Item** from the context menu. In the Add New Item dialog box, in the **Categories** list, choose **Local Project Items**. In the **Template** section, choose **Code File** (Using this template, Visual Studio will not generate code for you).

Right-click in the References item underneath the project name and select Add Reference from the context menu. Select these three items from the list in the dialog box that you're presented with: Sytem.dll, System.Drawing.dll, System.Windows.Forms.dll. Then, use the **Browse...** button to search for the Lassalle.Flow.dll and select it.

Finally, copy and paste the following C# code:

```
// C#
using System;
using System.Drawing;
using System.Windows.Forms;
using Lassalle.Flow;

public class Minimal: Form
{
  [STAThread]
  public static void Main()
  {
    Application.Run(new Minimal());
  }

  public Minimal()
  {
    Text = "Minimal AddFlow";
    Size = new Size(500, 300);

    // Create the AddFlow control
    AddFlow addflow = new AddFlow();
    addflow.Parent = this;
    addflow.Dock = DockStyle.Fill;
    addflow.AutoScroll = true;
    addflow.BackColor = SystemColors.Window;

    // Create a diagram
    CreateDiagram0(addflow);
  }

  void CreateDiagram0(AddFlow addflow)
  {
    // Create 3 nodes
    Node node1 = new Node(5, 5, 40, 40, "First node");
    Node node2 = new Node(120, 70, 50, 40, "Second node");
    Node node3 = new Node(5, 100, 40, 40, “Third node”);

    // Create 3 links
    Link link1 = new Link(“link 1”);
    Link link2 = new Link(“link 2”);
    Link link3 = new Link(“link 3”);

    // Add the nodes and the links to the diagram
    addflow.Nodes.Add(node1);
    addflow.Nodes.Add(node2);
    addflow.Nodes.Add(node3);
    node1.OutLinks.Add(link1, node2);
    node2.OutLinks.Add(link2, node2);
    node2.OutLinks.Add(link3, node3);
  }
}

' VB
Public Class Minimal
    Private Sub Minimal_Load(ByVal sender As System.Object, ByVal e As -
                                System.EventArgs) Handles MyBase.Load
        Text = "Minimal AddFlow"
        Size = New Size(500, 300)

        ' Create the AddFlow control
        Dim addflow As AddFlow = New AddFlow()
        addflow.Parent = Me
        addflow.Dock = DockStyle.Fill
        addflow.AutoScroll = True
        addflow.BackColor = SystemColors.Window

        ' Create a diagram
        CreateDiagram8(addflow)
    End Sub

    Private Sub CreateDiagram0(ByVal addflow As AddFlow)
        ' Create 3 nodes
        Dim node1 As Node = New Node(5, 5, 40, 40, "First node")
        Dim node2 As Node = New Node(120, 70, 50, 40, "Second node")
        Dim node3 As Node = New Node(5, 100, 40, 40, "Third node")

        ' Create 3 links
        Dim link1 As Link = New Link("link 1")
        Dim link2 As Link = New Link("link 2")
        Dim link3 As Link = New Link("link 3")

        ' Add the nodes and the links to the diagram
        addflow.Nodes.Add(node1)
        addflow.Nodes.Add(node2)
        addflow.Nodes.Add(node3)
        node1.OutLinks.Add(link1, node2)
        node2.OutLinks.Add(link2, node2)
        node2.OutLinks.Add(link3, node3)
    End Sub
End Class
```

If we compile and execute this program, it will create the following diagram:



In this diagram, the nodes and links receive default property values. For instance, the nodes have an elliptical shape. The links are composed of one line terminated by an arrow. The link 2 is reflexive and by default, it is created with 3 segments. The drawing colour is black. The text colour is black.

We are going to enhance this diagram. However, let us focus on the way nodes and links are created. First we create objects (nodes and links), then we add them to the AddFlow control. If we just write:

```
// C#
Node node1 = new Node(5, 5, 40, 40, "First node");

' VB
Dim node1 As Node = New Node(5, 5, 40, 40, "First node")
```

the node object is created. However it is not still part of the diagram. To make this node belong to the diagram, we have to add the following line:

```
// C#
addflow.Nodes.Add(node1);

' VB
addflow.Nodes.Add(node1)
```

As indicated in the reference guide, the Node class has 8 constructors. We have chosen one which creates a new instance of the Node class with an initial position and size and an initial text.

We could have also written:

```
// C#
Node node1 = new Node(5, 5, 40, 40);
node1.Text = "First node";

' VB
Dim node1 As Node = New Node(5, 5, 40, 40)
node1.Text = "First node"
```

Or:

```
// C#
RectangleF rect = new RectangleF(5, 5, 40, 40);
```

```
Node node1 = new Node(rect);
Node1.Text = "First node";

' VB
Dim rect As RectangleF = New RectangleF(5, 5, 40, 40)
Dim node1 As Node = New Node(rect)
node1.Text = "First node"
```

## Node Constructors

```
Node()
Node(Node model)
Node(RectangleF rect)
Node(RectangleF rect, Node model)
Node(float left, float top, float width, float height)
Node(float left, float top, float width, float height, Node model)
Node(float left, float top, float width, float height, string text)
Node(float left, float top, float width, float height, string text, Node
model)
```

Notice that you can also create a node by a cloning action:

```
// C#
Node node2 = (Node)node1.Clone();

' VB
Dim node2 As Node = node1.Clone
```

The cloned node has the same property values as its model. However, it is not still inserted in any AddFlow control.

Same thing for links: if we just write:

```
// C#
Link link1 = new Link("link 1");

' VB
Dim link1 As Link = New Link("link 1")
```

the link object is created. However, to give a true existence to this link, we have to add the following line:

```
// C#
node1.OutLinks.Add(link1, node2);

' VB
node1.OutLinks.Add(link1, node2)
```

This indicates that the destination node of the link is node 2 and the origin node is node 1.

The Link class has 4 constructors. We have chosen one that creates a new instance of the Link class with an initial text.

We could have also written:

```
// C#
Link link1 = new Link();
link1.Text = "link 1";

' VB
Dim link1 As Link = New Link()
link1.Text = "link 1"
```

**Link Constructors**

```
Link()
Link(Link model)
Link(string text)
Link(string text, Link model)
```

Notice that you can also create a link by a cloning action:

```
// C#
Link link2 = (Link)link1.Clone();

' VB
Dim link2 As Link = link1.Clone
```

The cloned link has the same property values as its model and it has the same collection of points. However, it is not still inserted in any AddFlow control. Its origin and destination nodes are null.

And finally you may also use the AddLink and CreateLink methods to create links:

```
// C#
Link link1 = new Link("link 1");
addflow.AddLink(link1, node1, node2);

' VB
Dim link1 As Link = New Link("link 1")
addflow.AddLink(link1, node1, node2)
```

or (anticipating the following section):

```
// C#
Link link1 = addflow.CreateLink(node1, node2);

' VB
Dim link1 As Link = addflow.CreateLink(node1, node2)
```

### 5.2.2 Another way to create the diagram

Now let us replace the CreateDiagram0 method by the following CreateDiagram1 method:

```
// C#
void CreateDiagram1(AddFlow addflow)
{
  // Create 3 nodes and add them to the diagram
  Node node1 = addflow.Nodes.Add(5, 5, 40, 40, "First node");
  Node node2 = addflow.Nodes.Add(120, 70, 50, 40, “Second node”);
  Node node3 = addflow.Nodes.Add(5, 100, 40, 40, “Third node”);

  // Create 3 links and add them to the diagram
  Link link1 = node1.OutLinks.Add(node2, “link 1”);
  Link link2 = node2.OutLinks.Add(node2, “link 2”);
  Link link3 = node2.OutLinks.Add(node3, “link 3”);
}

' VB
Private Sub CreateDiagram1(ByVal addflow As AddFlow)
  ' Create 3 nodes and add them to the diagram
  Dim node1 As Node = addflow.Nodes.Add(5, 5, 40, 40, "First node")
  Dim node2 As Node = addflow.Nodes.Add(120, 70, 50, 40, "Second node")
  Dim node3 As Node = addflow.Nodes.Add(5, 100, 40, 40, "Third node")

  ' Create 3 links and add them to the diagram
  Dim link1 As Link = node1.OutLinks.Add(node2, "link 1")
  Dim link2 As Link = node2.OutLinks.Add(node2, "link 2")
  Dim link3 As Link = node2.OutLinks.Add(node3, "link 3")
End Sub
```

If we compile and execute this new program, it will create the same diagram as before. However, the method is different. We do not use explicitly any constructor. The objects (nodes and links) are created and placed in the diagram with only one line of code. We will call this method the “old ActiveX” method since it is the method used in the ActiveX version of AddFlow.

Although this method seems to give a more compact code, we will prefer the previous method because it allows setting property values before inserting the node or the link in the diagram and this is faster to execute. For instance, the following code

```
Node node1 = new Node(5, 5, 40, 40, “First node”);
node1.FillColor = Color.LightYellow;
addflow.Nodes.Add(node1);
```

is faster than:

```
Node node1 = addflow.Nodes.Add(5, 5, 40, 40, “First node”);
node1.FillColor = Color.LightYellow;
```

Of course, you’ll see a difference only with very big diagrams.

The second method produces a more compact code? This is not actually true. We could have written the first method in the following manner:

```
// C#
void CreateDiagram2(AddFlow addflow)
{
  // Create and add the nodes to the diagram
  addflow.Nodes.Add(new Node(5, 5, 40, 40, “First node”));
  addflow.Nodes.Add(new Node(120, 70, 50, 40, “Second node”));
  addflow.Nodes.Add(new Node(5, 100, 40, 40, “Third node”));

  // Create and add the links to the diagram
  addflow.Nodes[0].OutLinks.Add(new Link(“link 1”), addflow.Nodes[1]);
  addflow.Nodes[1].OutLinks.Add(new Link(“link 2”), addflow.Nodes[1]);
  addflow.Nodes[1].OutLinks.Add(new Link("link 3"), addflow.Nodes[2]);
}

' VB
Private Sub CreateDiagram2(ByVal addflow As AddFlow)
  ' Create and add the nodes to the diagram
  addflow.Nodes.Add(New Node(5, 5, 40, 40, "First node"))
  addflow.Nodes.Add(New Node(120, 70, 50, 40, "Second node"))
  addflow.Nodes.Add(New Node(5, 100, 40, 40, "Third node"))

  ' Create and add the links to the diagram
  addflow.Nodes(0).OutLinks.Add(New Link("link 1"), addflow.Nodes(1))
  addflow.Nodes(1).OutLinks.Add(New Link("link 2"), addflow.Nodes(1))
  addflow.Nodes(1).OutLinks.Add(New Link("link 3"), addflow.Nodes(2))
End Sub
```

In this case, we use the **Nodes** collection instead of using a reference for each Node object.

To terminate this section, notice another way to create links, using the CreateLink method of AddFlow:

```
Link link1 = addflow.CreateLink(node1, node2);
Link link2 = addflow.CreateLink(node2, node2);
Link link3 = addflow.CreateLink(node2, node3);
```

Now we are going to enhance our diagram. However we need to have a better view of the node and link properties.

### 5.2.3 Node Properties

The node properties are described in the **Lassalle.Flow** namespace in the AddFlow help file. Some are inherited from the Item class. We can group them in 5 categories:

***Position and size***

**Rect**, **Location**, **Size** properties

***Style properties***

For each node, we can associate:

- A shape via its **Shape** object property (43 predefined shapes are available and you can create custom shapes with the **GraphicsPath** property of the Shape object: See the paragraph Custom Shapes).
- A shadow via its **Shadow** object property
- Some colors (**FillColor**, **DrawColor**, **TextColor**, **GradientColor**).
- Some properties that defines how to display the text in the node: **Font**, **Trimming**
- Some properties (**Alignment**, **TextMargin** and **ImagePosition**) that determines how the text and the image are positioned in the node
- The **AutoSize** property which allows to adjust the node size to its picture size or its text size or to adjust the picture size to the node size
- Some properties about how is displayed the node (**Transparent**, **Hidden**, **DashStyle**, **DrawWidth, OwnerDraw, Gradient** properties)

***Data properties***

- A text (**Text** property) can be displayed inside a node.
- An image (**ImageIndex** property) can also be displayed inside a node.
- A tooltip (**Tooltip** property) can be associated to a node.
- A url (**Url** property) can be associated to a node. This url can be used to assign a HTML hyperlink to the node in a web page when the diagram is displayed in a web page using SVG.
- It is also possible to associate an object to a node via its **Tag** property.

***Graph properties***

These are the **OutLinks**, **InLinks** and **Links** collection properties that allow getting the links (in, out, in and out) of the node and therefore allowing navigating in the graph.

There is also the **ConnectedItems** collection property that returns the collection of items (nodes and links) that belong to the same connected part of the graph as the node.

The **Index** property returns the position of the node in the Nodes collection.

***Behaviour properties***

**Selected**, **Selectable**, **XMoveable**, **YMoveable**, **XSizeable**, **YSizeable**, **ZOrder**, **LabelEdit**, **IsInView**, **IsEditing**, **Logical**, etc...

### 5.2.4 Link Properties

The link properties are described in the **Lassalle.Flow** namespace in the AddFlow help file. Some are inherited from the Item class. We can group them in 5 categories:

***Segments***

A link is composed of general segments defined by a collection of points (**Points** properties).

***Style properties***

For each link, we can associate:

- A **Line** object that defines the style of link (polyline, Bezier, Spline, orthogonal)
- Three properties defining the arrow styles (**ArrowOrg**, **ArrowDst**, **ArrowMid**). However you have another method to create arrows, using the **StartCap**, **EndCap**, **CustomStartCap** and **CustomEndCap** properties)
- The **Jump** property (A jump is displayed at the intersection of 2 links)
- Some colors (**DrawColor**, **TextColor**).
- Some properties that defines how to display the text in the link: **Font**, **OrientedText**
- Some properties about how is displayed the link (**Hidden**, **DashStyle**, **DrawWidth, OwnerDraw** properties)

***Data properties***

- A text (**Text** property) can be displayed near a link.
- A tooltip (**Tooltip** property) can be associated to a link.
- A url (**Url** property) can be associated to a node. This url can be used to assign a HTML hyperlink to the link in a web page when the diagram is displayed in a web page using SVG.
- It is also possible to associate an object to a link via its **Tag** property.

***Graph properties***

The **Org** property returns/sets the reference of the origin node of the link whereas the **Dst** property returns/sets the reference of the destination node of the link.

***Behaviour properties***

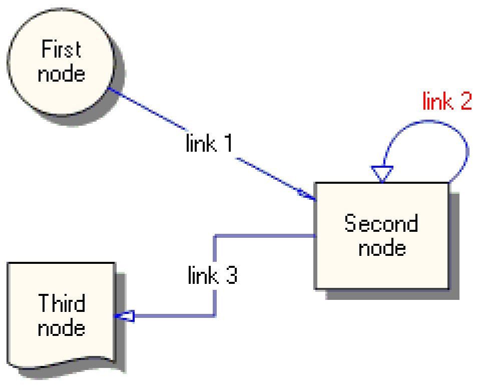
**Selected**, **Selectable**, **ZOrder**, **IsInView**, **Logical**, **Stretchable**, **Rigid**, **AdjustDst**, **AdjustOrg**, etc...

### 5.2.5 Changing property values

Let us use the following diagram creation method:

```
// C#
void CreateDiagram3(AddFlow addflow)
{
  // Create 3 yellow nodes with a shadow.
  // The second node is rectangular
  // and the third one has a Document shape style.
  Node node1 = new Node(5, 5, 40, 40, "First node");
  node1.FillColor = Color.LightYellow;
  node1.Shadow.Style = ShadowStyle.RightBottom;

  Node node2 = new Node(120, 70, 50, 40, "Second node");
  node2.FillColor = Color.LightYellow;
  node2.Shadow.Style = ShadowStyle.RightBottom;
  node2.Shape.Style = ShapeStyle.Rectangle;

  Node node3 = new Node(5, 100, 40, 40, "Third node");
  node3.FillColor = Color.LightYellow;
  node3.Shadow.Style = ShadowStyle.RightBottom;
  node3.Shape.Style = ShapeStyle.Document;

  // Create 3 links.
  // Each link is blue and its BackMode property set to Opaque.
  // The second link has a Bezier style, color of its text is red, and
  // its destination arrow head angle is 30°.
  // The third link has a "HVH" style.
  Link link1 = new Link("link 1");
  link1.DrawColor = Color.Blue;
  link1.BackMode = BackMode.Opaque;

  Link link2 = new Link("link 2");
  link2.DrawColor = Color.Blue;
  link2.BackMode = BackMode.Opaque;
  link2.Line.Style = LineStyle.Bezier;
  link2.TextColor = Color.Red;
  link2.ArrowDst.Angle = ArrowAngle.deg30;
  Link link3 = new Link(“link 3”);
  link3.DrawColor = Color.Blue;
  link3.BackMode = BackMode.Opaque;
  link3.Line.Style = LineStyle.HVH;

  // Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1);
  addflow.Nodes.Add(node2);
  addflow.Nodes.Add(node3);
  node1.OutLinks.Add(link1, node2);
  node2.OutLinks.Add(link2, node2);
  node2.OutLinks.Add(link3, node3);
}

' VB
Private Sub CreateDiagram3(ByVal addflow As AddFlow)
  ' Create 3 yellow nodes with a shadow.
  ' The second node is rectangular and the third one has the “Document”
  ' shape style.
  Dim node1 As Node = New Node(5, 5, 40, 40, "First node")
  node1.FillColor = Color.LightYellow
  node1.Shadow.Style = ShadowStyle.RightBottom

  Dim node2 As Node = New Node(120, 70, 50, 40, "Second node")
  node2.FillColor = Color.LightYellow
  node2.Shadow.Style = ShadowStyle.RightBottom
  node2.Shape.Style = ShapeStyle.Rectangle

  Dim node3 As Node = New Node(5, 100, 40, 40, "Third node")
  node3.FillColor = Color.LightYellow
  node3.Shadow.Style = ShadowStyle.RightBottom
  node3.Shape.Style = ShapeStyle.Document

  ' Create 3 links.
  ' Each link is blue and its BackMode property set to Opaque.
  ' The second link has a Bezier style, color of its text is red, and its
  ' destination arrow head angle is 30°.
  ' The third link has a "HVH" style.
  Dim link1 As Link = New Link("link 1")
  link1.DrawColor = Color.Blue
  link1.BackMode = BackMode.Opaque

  Dim link2 As Link = New Link("link 2")
  link2.DrawColor = Color.Blue
  link2.BackMode = BackMode.Opaque
  link2.Line.Style = LineStyle.Bezier
  link2.TextColor = Color.Red
  link2.ArrowDst.Angle = ArrowAngle.deg30

  Dim link3 As Link = New Link("link 3")
  link3.DrawColor = Color.Blue
  link3.BackMode = BackMode.Opaque
  link3.Line.Style = LineStyle.HVH

  ' Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1)
  addflow.Nodes.Add(node2)
  addflow.Nodes.Add(node3)
  node1.OutLinks.Add(link1, node2)
  node2.OutLinks.Add(link2, node2)
  node2.OutLinks.Add(link3, node3)
End Sub
```

If we compile and execute this program, it will create the following diagram:



Now, our nodes have distinct shapes. They have a shadow. Their filling colour is LightYellow. And our links are blue. The reflexive link is a curved line, its text is red and the angle of its arrow head is larger.

Therefore we know how to give distinct property values for each object.

Notice however that to specify the colour of each node, we had to do it for each node, even if the colour is the same. It is the same thing for the links. For a big diagram, this may be annoying to repeat always the same code for each object.

Fortunately, AddFlow allows using default property values that apply to all the next created nodes or links.

### 5.2.6 Default property values

Now let us replace the CreateDiagram3 method by the following CreateDiagram4 method:

```
// C#
void CreateDiagram4(AddFlow addflow)
{
  // Default property values for nodes
  addflow.DefNodeProp.FillColor = Color.LightYellow;
  addflow.DefNodeProp.Shadow.Style = ShadowStyle.RightBottom;

  // Default property values for links
  addflow.DefLinkProp.DrawColor = Color.Blue;
  addflow.DefLinkProp.BackMode = BackMode.Opaque;

  // Create 3 nodes and assign them some property values
  Node node1 = new Node(5, 5, 40, 40, "First node", addflow.DefNodeProp);
  Node node2 = new Node(120,70,50,40,"Second node", addflow.DefNodeProp);
  node2.Shape.Style = ShapeStyle.Rectangle;

  Node node3 = new Node(5,100,40,40, "Third node", addflow.DefNodeProp);
  node3.Shape.Style = ShapeStyle.Document;

  // Create 3 links
```

```
    Link link1 = new Link("link 1", addflow.DefLinkProp);

    Link link2 = new Link("link 2", addflow.DefLinkProp);
    link2.ArrowDst.Angle = ArrowAngle.deg30;
    link2.Line.Style = LineStyle.Bezier;
    link2.TextColor = Color.Red;

    Link link3 = new Link("link 3", addflow.DefLinkProp);
    link3.Line.Style = LineStyle.HVH;

    // Add the nodes and the links to the diagram
    addflow.Nodes.Add(node1);
    addflow.Nodes.Add(node2);
    addflow.Nodes.Add(node3);
    node1.OutLinks.Add(link1, node2);
    node2.OutLinks.Add(link2, node2);
    node2.OutLinks.Add(link3, node3);
  }

  ' VB
  Private Sub CreateDiagram4(ByVal addflow As AddFlow)
    ' Default property values for nodes
    addflow.DefNodeProp.FillColor = Color.LightYellow
    addflow.DefNodeProp.Shadow.Style = ShadowStyle.RightBottom

    ' Default property values for links
    addflow.DefLinkProp.DrawColor = Color.Blue
    addflow.DefLinkProp.BackMode = BackMode.Opaque

    ' Create 3 nodes and assign them some property values
    Dim node1 As Node = New Node(5, 5, 40, 40, "First node", _
    addflow.DefNodeProp)

    Dim node2 As Node = New Node(120, 70, 50, 40, "Second node", _
    addflow.DefNodeProp)
    node2.Shape.Style = ShapeStyle.Rectangle

    Dim node3 As Node = New Node(5, 100, 40, 40, "Third node", _
    addflow.DefNodeProp)
    node3.Shape.Style = ShapeStyle.Document

    ' Create 3 links
    Dim link1 As Link = New Link("link 1", addflow.DefLinkProp)

    Dim link2 As Link = New Link("link 2", addflow.DefLinkProp)
    link2.ArrowDst.Angle = ArrowAngle.deg30
    link2.Line.Style = LineStyle.Bezier
    link2.TextColor = Color.Red

    Dim link3 As Link = New Link("link 3", addflow.DefLinkProp)
    link3.Line.Style = LineStyle.HVH

    ' Add the nodes and the links to the diagram
    addflow.Nodes.Add(node1)
    addflow.Nodes.Add(node2)
    addflow.Nodes.Add(node3)
    node1.OutLinks.Add(link1, node2)
    node2.OutLinks.Add(link2, node2)
    node2.OutLinks.Add(link3, node3)
  End Sub
```

If we compile and execute this new program, it will create the same diagram. However, our program is smaller because we have used the **DefNodeProp** and the **DefLinkProp** properties of AddFlow which allow specifying default property values for nodes and links. For instance, writing:

```
addflow.DefNodeProp.FillColor = Color.LightYellow;
```

indicates that all the nodes that will be created after will be filled with a LightYellow color.

Then you just need to specify the property values that differ from the defaults.

How to write the same program using the “old activex” style? This is demonstrated in the following CreateDiagram5 method:

```
// C#
void CreateDiagram5(AddFlow addflow)
{
  // Default property values for nodes
  addflow.DefNodeProp.FillColor = Color.LightYellow;
  addflow.DefNodeProp.Shadow.Style = ShadowStyle.RightBottom;

  // Default property values for links
  addflow.DefLinkProp.DrawColor = Color.Blue;
  addflow.DefLinkProp.BackMode = BackMode.Opaque;

  // Create 3 nodes, add them to the diagram and
  // assign them some property values
  Node node1 = addflow.Nodes.Add(5, 5, 40, 40, "First node");

  Node node2 = addflow.Nodes.Add(120, 70, 50, 40, "Second node");
  node2.Shape.Style = ShapeStyle.Rectangle;

  Node node3 = addflow.Nodes.Add(5, 100, 40, 40, "Third node");
  node3.Shape.Style = ShapeStyle.Document;

  // Create 3 links, add them to the diagram and assign
  // them some property values
  Link link1 = node1.OutLinks.Add(node2, "link 1");
  Link link2 = node2.OutLinks.Add(node2, "link 2");

  link2.Line.Style = LineStyle.Bezier;
  link2.TextColor = Color.Red;
  link2.ArrowDst.Angle = ArrowAngle.deg30;

  Link link3 = node2.OutLinks.Add(node3, "link 3");
  link3.Line.Style = LineStyle.HVH;
}

' VB
Private Sub CreateDiagram5(ByVal addflow As AddFlow)
  ' Default property values for nodes
  addflow.DefNodeProp.FillColor = Color.LightYellow
  addflow.DefNodeProp.Shadow.Style = ShadowStyle.RightBottom

  ' Default property values for links
  addflow.DefLinkProp.DrawColor = Color.Blue
  addflow.DefLinkProp.BackMode = BackMode.Opaque

  ' Create 3 nodes, add them to the diagram
  ' and assign them some property values
  Dim node1 As Node = addflow.Nodes.Add(5, 5, 40, 40, "First node")

  Dim node2 As Node = addflow.Nodes.Add(120, 70, 50, 40, "Second node")
  node2.Shape.Style = ShapeStyle.Rectangle

  Dim node3 As Node = addflow.Nodes.Add(5, 100, 40, 40, "Third node")
  node3.Shape.Style = ShapeStyle.Document

  ' Create 3 links, add them to the diagram
  ' and assign them some property values
  Dim link1 As Link = node1.OutLinks.Add(node2, "link 1")

  Dim link2 As Link = node2.OutLinks.Add(node2, "link 2")
  link2.Line.Style = LineStyle.Bezier
  link2.TextColor = Color.Red
  link2.ArrowDst.Angle = ArrowAngle.deg30

  Dim link3 As Link = node2.OutLinks.Add(node3, "link 3")
  link3.Line.Style = LineStyle.HVH
 End Sub
```

Notice that the **DefNodeProp** and the **DefLinkProp** properties have also an interactive effect. Not only the nodes created programmatically will be filled with a LightYellow colour but also the nodes created interactively with the mouse. This may be interesting or not, depending on what you intend to do.

Anyway, it is also possible to specify default values when creating a diagram programmatically and have other default values for the interactive creation.

### 5.2.7 The DefNodeProp and DefLinkProp properties

Now let us replace the CreateDiagram5 method by the following CreateDiagram6 method:

```
// C#
void CreateDiagram6(AddFlow addflow)
{
  Node dn = (Node)addflow.DefNodeProp.Clone();
  Link dl = (Link)addflow.DefLinkProp.Clone();

  // Default property values for nodes created programmatically
  dn.FillColor = Color.LightYellow;
  dn.Shadow.Style = ShadowStyle.RightBottom;

  // Default property values for links created programmatically
  dl.DrawColor = Color.Blue;
  dl.BackMode = BackMode.Opaque;

  // Create 3 nodes and assign them some property values
  Node node1 = new Node(5, 5, 40, 40, “First node”, dn);

  Node node2 = new Node(120, 70, 50, 40, “Second node”, dn);
  node2.Shape.Style = ShapeStyle.Rectangle;

  Node node3 = new Node(5, 100, 40, 40, “Third node”, dn);
  node3.Shape.Style = ShapeStyle.Document;
  // Create 3 links
  Link link1 = new Link(“link 1”, dl);

  Link link2 = new Link(“link 2”, dl);
  link2.ArrowDst.Angle = ArrowAngle.deg30;
  link2.Line.Style = LineStyle.Bezier;
  link2.TextColor = Color.Red;

  Link link3 = new Link(“link 3”, dl);
  link3.Line.Style = LineStyle.HVH;

  // Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1);
  addflow.Nodes.Add(node2);
  addflow.Nodes.Add(node3);
  node1.OutLinks.Add(link1, node2);
  node2.OutLinks.Add(link2, node2);
  node2.OutLinks.Add(link3, node3);
}

' VB
Private Sub CreateDiagram6(ByVal addflow As AddFlow)
  Dim dn As Node = addflow.DefNodeProp.Clone()
  Dim dl As Link = addflow.DefLinkProp.Clone()

  ' Default property values for nodes created programmatically
  dn.FillColor = Color.LightYellow
  dn.Shadow.Style = ShadowStyle.RightBottom

  ' Default property values for links created programmatically
  dl.DrawColor = Color.Blue
  dl.BackMode = BackMode.Opaque

  ' Create 3 nodes and assign them some property values
  Dim node1 As Node = New Node(5, 5, 40, 40, "First node", dn)

  Dim node2 As Node = New Node(120, 70, 50, 40, "Second node", dn)
  node2.Shape.Style = ShapeStyle.Rectangle

  Dim node3 As Node = New Node(5, 100, 40, 40, "Third node", dn)
  node3.Shape.Style = ShapeStyle.Document

  ' Create 3 links
  Dim link1 As Link = New Link("link 1", dl)

  Dim link2 As Link = New Link("link 2", dl)
  link2.ArrowDst.Angle = ArrowAngle.deg30
  link2.Line.Style = LineStyle.Bezier
  link2.TextColor = Color.Red

  Dim link3 As Link = New Link("link 3", dl)
  link3.Line.Style = LineStyle.HVH

  ' Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1)
  addflow.Nodes.Add(node2)
  addflow.Nodes.Add(node3)
  node1.OutLinks.Add(link1, node2)
  node2.OutLinks.Add(link2, node2)
  node2.OutLinks.Add(link3, node3)
End Sub
```

In this method, we clone the DefNodeProp and the DefLinkProp properties respectively in a **Node** and a **Link** object. Then we change some property values of these objects and we use them when creating nodes and links.

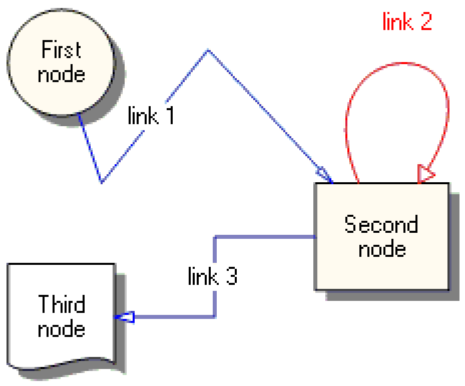
### 5.2.8 Stretching the links

We would like to add segments to our links. The following CreateDiagram7 method demonstrates how to do that.

```
// C#
void CreateDiagram7(AddFlow addflow)
{
  Node dn = (Node)addflow.DefNodeProp.Clone();
  Link dl = (Link)addflow.DefLinkProp.Clone();

  // Default property values for nodes created programmatically
  dn.FillColor = Color.LightYellow;
  dn.Shadow.Style = ShadowStyle.RightBottom;

  // Default property values for links created programmatically
  dl.DrawColor = Color.Blue;
  dl.BackMode = BackMode.Opaque;

  // Create 3 nodes and assign them some property values
  Node node1 = new Node(5, 5, 40, 40, "First node", dn);

  Node node2 = new Node(120, 70, 50, 40, "Second node", dn);
  node2.Shape.Style = ShapeStyle.Rectangle;

  Node node3 = new Node(5, 100, 40, 40, "Third node", dn);
  node3.Shape.Style = ShapeStyle.Document;

  // Create 3 links
  Link link1 = new Link("link 1", dl);

  Link link2 = new Link("link 2", dl);
  link2.ArrowDst.Angle = ArrowAngle.deg30;
  link2.Line.Style = LineStyle.Bezier;
  link2.TextColor = Color.Red;

  Link link3 = new Link("link 3", dl);
  link3.Line.Style = LineStyle.HVH;

  // Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1);
  addflow.Nodes.Add(node2);
  addflow.Nodes.Add(node3);
  node1.OutLinks.Add(link1, node2);
  node2.OutLinks.Add(link2, node2);
  node2.OutLinks.Add(link3, node3);

  // Add 2 points (therefore 2 segments) to the first link
  link1.Points.Add(new PointF(40, 70));
  link1.Points.Add(new PointF(80, 20));

  // Stretch the reflexive link
  link2.Points[1] = new PointF(110, 10);
  link2.Points[2] = new PointF(200, 10);
}
```

```vb
' VB
Private Sub CreateDiagram7(ByVal addflow As AddFlow)
  Dim dn As Node = addflow.DefNodeProp.Clone()
  Dim dl As Link = addflow.DefLinkProp.Clone()

  ' Default property values for nodes created programmatically
  dn.FillColor = Color.LightYellow
  dn.Shadow.Style = ShadowStyle.RightBottom

  ' Default property values for links created programmatically
  dl.DrawColor = Color.Blue
  dl.BackMode = BackMode.Opaque

  ' Create 3 nodes and assign them some property values
  Dim node1 As Node = New Node(5, 5, 40, 40, "First node", dn)

  Dim node2 As Node = New Node(120, 70, 50, 40, "Second node", dn)
  node2.Shape.Style = ShapeStyle.Rectangle

  Dim node3 As Node = New Node(5, 100, 40, 40, "Third node", dn)
  node3.Shape.Style = ShapeStyle.Document

  ' Create 3 links
  Dim link1 As Link = New Link("link 1", dl)

  Dim link2 As Link = New Link("link 2", dl)
  link2.ArrowDst.Angle = ArrowAngle.deg30
  link2.Line.Style = LineStyle.Bezier
  link2.TextColor = Color.Red

  Dim link3 As Link = New Link("link 3", dl)
  link3.Line.Style = LineStyle.HVH

  ' Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1)
  addflow.Nodes.Add(node2)
  addflow.Nodes.Add(node3)
  node1.OutLinks.Add(link1, node2)
  node2.OutLinks.Add(link2, node2)
  node2.OutLinks.Add(link3, node3)

  ' Add 2 points (therefore 2 segments) to the first link
  link1.Points.Add(New PointF(40, 70))
  link1.Points.Add(New PointF(80, 20))

  ' Stretch the reflexive link
  link2.Points(1) = New PointF(110, 10)
  link2.Points(2) = New PointF(200, 10)
End Sub
```

If we compile and execute this program, it will create the following diagram:



To add segments to a link or to alter its shape, you have to use the **Points** property collection of the link. It is important to notice that **this has to be done after the link insertion in the diagram**.

You can add points (and therefore segments) to the link 1 because its link line style is Polyline. You could also do that it its link line style was Spline. However, for the other cases (for instance Bezier as for the link 2), you cannot add points. You can however still modify the position of the points.

The rules for managing the link collection of points are the following:

- You can manipulate the link Points collection only after its insertion in the diagram.
- After its insertion in the diagram, a link has at least 2 points.
- You cannot remove these 2 points, even if you execute the Clear Method of the Points collection. The Count property of the Points collection is always superior or equal to 2. If you execute the Clear Method, then Count = 2.
- You can add or delete points only if the link line style is Polyline or Spline. However, if the link line style is Bezier, then the Points collection has 4 points in any case.
- You cannot change the first point of the Points collection except if the AdjustOrg property is true.
- You cannot change the last point of the Points collection except if the AdjustDst property is true.
- You can change each other point of the Points collection in any case.

## 5.3 Displaying an image in a node

You can associate an image to a node with the **ImageIndex** property. This property returns/sets the index of an image stored in the array of images defined by the **Images** property of the AddFlow class.
The Images collection is a collection of **FlowImage** objects. The FlowImage class is used to define the images used in nodes. It offers 2 ways to define such images:

- directly via a GDI+ Image object (property **Image**)
- via the url of an image object (property **Url**)

Example (using urls)

```
// C#
// Create the collection of images that will be used for nodes
```

```
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\heart.bmp");
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\diamond.bmp");
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\spade.bmp");
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\club.bmp");

// Create 4 nodes
Node node1 = new Node(20, 20, 60, 60);
Node node2 = new Node(20, 100, 60, 60);
Node node3 = new Node(100, 20, 60, 60);
Node node4 = new Node(100, 100, 60, 60);

// Associate an image to each node
node1.ImageIndex = 0;
node2.ImageIndex = 1;
node3.ImageIndex = 2;
node4.ImageIndex = 3;

// Add the nodes to the diagram
AddFlow1.Nodes.Add(node1);
AddFlow1.Nodes.Add(node2);
AddFlow1.Nodes.Add(node3);
AddFlow1.Nodes.Add(node4);

' VB
' Create the collection of images that will be used for nodes
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\heart.bmp")
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\diamond.bmp")
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\spade.bmp")
AddFlow1.Images.Add("C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\Graphics\bitmaps\assorted\club.bmp")

' Create 4 nodes
Dim node1 As Node = New Node(20, 20, 60, 60)
Dim node2 As Node = New Node(20, 100, 60, 60)
Dim node3 As Node = New Node(100, 20, 60, 60)
Dim node4 As Node = New Node(100, 100, 60, 60)

' Associate an image to each node
node1.ImageIndex = 0
node2.ImageIndex = 1
node3.ImageIndex = 2
node4.ImageIndex = 3

' Add the nodes to the diagram
AddFlow1.Nodes.Add(node1)
AddFlow1.Nodes.Add(node2)
AddFlow1.Nodes.Add(node3)
AddFlow1.Nodes.Add(node4)
```

## 5.4 Selection of items

### 5.4.1 Interactive selection

You can select a node or a link interactively by clicking it with the mouse.

You can also select several nodes or links interactively with the mouse if multi-selection is allowed (in such a case, the **MultiSel** property must be true).

**Note on selecting interactively a link with the mouse**

If the link is made of one or several segments, then if you want to select it with the mouse, you have just to click near one of its segments. If the link is a Bezier curve, then you have just to click near the curve.

**Note on interactive multi-selection**

You can select several items by clicking them with the mouse and simultaneously pressing the shift or control key. Or you can select items with a selection rectangle, if the **MouseAction** property is set to MouseAction.Selection or MouseAction.Selection2. In this last case, you bring the mouse cursor into the AddFlow control, press the left button, move the mouse and release the left button. All nodes or links partly inside the selection rectangle are selected. Then you can unselect some nodes by clicking them with the mouse and simultaneously pressing the shift or control key. You can select them again by using the same method.

You can use the Shift Key in conjunction with a selection rectangle. For instance, you can draw a selection rectangle with the mouse, which cause items inside this rectangle to be selected. Then, if you press the Shift key, you can select additional items or unselect some selected items.

### 5.4.2 Programmatic selection

You can select a node or a link programmatically with the AddFlow **SelectedItem** property or with the **Selected** property of the Item object:

```
addflow.SelectedItem = node;
```

```
or
```

```
node.Selected = true;
```

You can also select several nodes or links programmatically with the **Selected** property:

```
// C#
node1.Selected = true;
node2.Selected = true;
link1.Selected = true;

' VB
node1.Selected = True
```

```
node2.Selected = True
link1.Selected = True
```

The **SelectedItems** collection property of AddFlow allows getting each selected item. For instance:

```
// C#
// Make each selected item (node or link) red
foreach (Item item in addflow.SelectedItems)
  item.DrawColor = Color.Red;

' VB
' Make each selected item (node or link) red
For Each item As Item In AddFlow1.SelectedItems
  item.DrawColor = Color.Red
Next
```

### 5.4.3 Selection events

Two events are dealing with selection:

- The **AfterSelect** event is fired after a user has finished drawing a selection rectangle with the mouse.
- The **SelectionChange** event is fired each time the selection status of an item is changed.

### 5.4.4 Hit Testing

You can also know what object is under the mouse with the **PointedItem** property that returns the reference of the item under the mouse. If several objects are under the mouse, the returned object is the one that is at the top of the Z-order list. You may change this order with the **ZOrder** property of the Item object.

Instead of using the **PointedItem** property, you may use the **GetItemAt** method.

## 5.5 Diagram navigation

AddFlow provides a set of properties and methods to navigate in a diagram (“Network traversals”). Notice that the majority of the properties and methods described here are demonstrated in the **navig** sample provided with AddFlow.

The **Items** property of the AddFlow control allows accessing every item (nodes and links) of a diagram. For instance:

```
// C#
foreach (Item item in AddFlow1.Items)
  item.DrawColor = Color.Red;

' VB
For Each item As Item In AddFlow1.Items
  item.DrawColor = Color.Red
Next
```

The **SelectedItems** property of the AddFlow control allows accessing every selected item (nodes and links) of a diagram. For instance:

```
// C#
foreach (Item item in AddFlow1.SelectedItems)
  item.DrawColor = Color.Red;

' VB
For Each item As Item In AddFlow1.SelectedItems
  item.DrawColor = Color.Red
Next
```

The **Nodes** property of the AddFlow control allows accessing every nodes of a diagram. For instance:

```
// C#
foreach (Node node in AddFlow1.Nodes)
  node.DrawColor = Color.Red;

' VB
For Each node As Node In AddFlow1.Nodes
  node.DrawColor = Color.Red
Next
```

The **OutLinks** property of the Node object allows accessing every link that leave a node. For instance:

```
// C#
foreach (Link link in node.OutLinks)
  link.DrawColor = Color.Red;

' VB
For Each link As Link In node.OutLinks
  link.DrawColor = Color.Red
Next
```

The **InLinks** property of the Node object allows accessing every link that come to a node. For instance:

```
// C#
foreach (Link link in node.InLinks)
  link.DrawColor = Color.Red;

' VB
For Each link As Link In node.InLinks
  link.DrawColor = Color.Red
Next
```

The **Links** property of the Node object allows accessing every link that come to a node or leave it. For instance:

```
// C#
```

```
foreach (Link link in node.Links)
  link.DrawColor = Color.Red;

' VB
For Each link As Link In node.Links
  link.DrawColor = Color.Red
Next
```

The **GetLinkedNode** method of the Node object returns the node connected via a given link.

```
node2 = node1.GetLinkedNode(link);
```

The **Org** property of the Link object returns/sets the reference of the origin node of the link.

The **Dst** property of the Link object returns/sets the reference of the destination node of the link.

The **Reverse** method of the Link object reverses the link origin and destination nodes.

## 5.6 Parent-Child relationship

An item (node or link) can be the parent of a collection of nodes. This Parent-Child relationship is established using the **Parent** property. The items must be added in the diagram before using this property.

For a given item, you can get its collection of children using the **Children** property. For instance:

```
// C#
foreach (Node label in node.Children)
  label.Transparent = false;

' VB
For Each label As Node In node.Children
  node.Transparent = False
Next
```

This hierarchy feature can be a way to associate many labels to a node or a link. If you move the node or stretch the link, its labels follow it. The **AttachmentStyle** property defined how the child node follows its parent item.

It is also a way to place several nodes inside a node. To facilitate that, you can use the **Dock** property which works exactly as the Dock property of controls.

And finally, it is also an efficient way to group nodes to create another node. (See the Group and Ungroup menu items in the afEdit sample).

What happens for the children nodes if you select the parent or if you remove it? This is determined by the value of the **HighlightChildren** and **RemoveChildren** properties.

Finally, the following table gives us the list of properties needed to implement the Parent-Child hierarchy.

| | |
|---|---|
| **AttachmentStyle** | Determines how a node is attached to its parent item. |
| **Children** | Returns the collection of children nodes of an item (node or link). |
| **Dock** | Determines how a node is placed inside its parent node. |
| **HighlightChildren** | Determines if the children are highlighted when the item is selected. |
| **Parent** | Returns/sets the parent item (node or link) of a node. |
| **RemoveChildren** | Determines if the children are removed when the item is removed. |

In this paragraph, we will see:

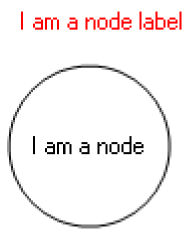
- how to attach a label to a node
- how to attach a label to a link
- how to place nodes inside a node.

## 5.6.1 Attach a label to a node

The following piece of code shows how to make a node be the child of another node.

```
' VB

' Create a node
Dim node As Node = New Node(50, 50, 80, 80, "I am a node")

' Create another node that will be used as a label for the first
' node.
' This node is defined as unselectable, transparent. It will follow
' its parent node if any. Its Logical property is set to False to
' emphasize its labeling nature.
Dim label As Node = New Node(50, 20, 60, 20, "I am a node label")
label.Shape.Style = ShapeStyle.Rectangle
label.DrawColor = Color.Transparent
label.TextColor = Color.Red
label.Transparent = True
label.Logical = False
label.Selectable = False
label.AutoSize = Lassalle.Flow.AutoSize.NodeToText
label.AttachmentStyle = AttachmentStyle.Item

' Add  nodes in the diagram.
AddFlow1.Nodes.Add(node)
AddFlow1.Nodes.Add(label)

' Create the Parent-Child relationship.
label.Parent = node
```

If we execute this code, it will create the following diagram:



If you move the circle node, its label node will follow it because its Attachment style is `AttachmentStyle.Item`.

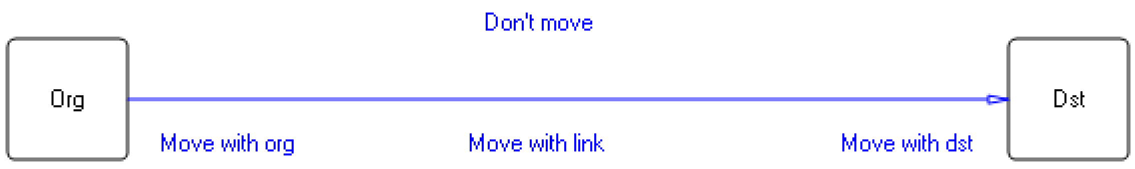
### 5.6.2 Attach a label to a link

The following piece of code shows how to several labels to a link.

```
' VB
' Create 2 nodes and a link
Dim node1 As Node = New Node(50, 50, 60, 60, "Org")
node1.Shape.Style = ShapeStyle.RoundRect

Dim node2 As Node = New Node(550, 50, 60, 60, "Dst")
node2.Shape.Style = ShapeStyle.RoundRect

Dim link As Link = New Link()
link.DrawColor = Color.Blue

' Create 4 labels
Dim label1 As Node = New Node(120, 90, 80, 25, "Move with org")
label1.Shape.Style = ShapeStyle.Rectangle
label1.DrawColor = Color.Transparent
label1.TextColor = Color.Blue
label1.Transparent = True
label1.AttachmentStyle = AttachmentStyle.OriginNode

Dim label2 As Node = New Node(275, 90, 80, 25, "Move with link")
label2.Shape.Style = ShapeStyle.Rectangle
label2.DrawColor = Color.Transparent
label2.TextColor = Color.Blue
label2.Transparent = True
label2.AttachmentStyle = AttachmentStyle.Item

Dim label3 As Node = New Node(460, 90, 80, 25, "Move with dst")
label3.Shape.Style = ShapeStyle.Rectangle
label3.DrawColor = Color.Transparent
label3.TextColor = Color.Blue
label3.Transparent = True
label3.AttachmentStyle = AttachmentStyle.DestinationNode

Dim label4 As Node = New Node(275, 30, 80, 25, "Don't move")
label4.Shape.Style = ShapeStyle.Rectangle
label4.DrawColor = Color.Transparent
label4.TextColor = Color.Blue
label4.Transparent = True
label4.AttachmentStyle = AttachmentStyle.None

' Add the items to the diagram.
AddFlow1.Nodes.Add(node1)
AddFlow1.Nodes.Add(node2)
node1.OutLinks.Add(link, node2)
AddFlow1.Nodes.Add(label1)
AddFlow1.Nodes.Add(label2)
AddFlow1.Nodes.Add(label3)
AddFlow1.Nodes.Add(label4)

' Create the Parent-Child relationships.
label1.Parent = link
label2.Parent = link
label3.Parent = link
```

```
label4.Parent = link
```

If we execute this code, it will create the following diagram:



If you move the origin node, only the node “Move with org” and the node “Move with link” will move.

If you move the destination node, only the node “Move with dst” and the node “Move with link” will move.

If you stretch the link, only the node “Move with link” will move. Then if you create several segments, the label node will be attached to the nearer segment and it will follow this segment.

The node “Don’t move” does not move because its AttachmentStyle is set to AttachmentStyle.None.

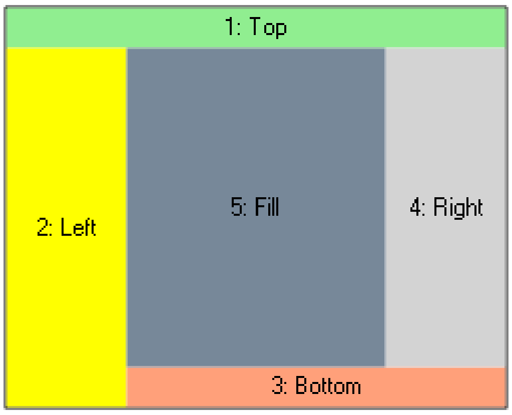
### 5.6.3 Place nodes inside a node.

You can place a node inside another one using the Dock property which works the same way as the Dock property for controls. For instance the first child node is placed at the top of the parent node because its Dock property is set to DockStyle.Top.

```
' VB

' Create the "parent" node
Dim parent As Node = New Node(150, 50, 250, 200)
parent.Shape.Style = ShapeStyle.Rectangle

' Create 5 child nodes
Dim child1 As Node = New Node(50, 130, 90, 20, "1: Top")
child1.Shape.Style = ShapeStyle.Rectangle
child1.DrawColor = Color.Transparent
child1.FillColor = Color.LightGreen
child1.AttachmentStyle = AttachmentStyle.Item
child1.Dock = DockStyle.Top
child1.Selectable = False

Dim child2 As Node = New Node(50, 160, 60, 20, "2: Left")
child2.Shape.Style = ShapeStyle.Rectangle
child2.DrawColor = Color.Transparent
child2.FillColor = Color.Yellow
child2.AttachmentStyle = AttachmentStyle.Item
child2.Dock = DockStyle.Left
child2.Selectable = False

Dim child3 As Node = New Node(50, 190, 60, 20, "3: Bottom")
```

```
        child3.Shape.Style = ShapeStyle.Rectangle
        child3.DrawColor = Color.Transparent
        child3.FillColor = Color.LightSalmon
        child3.AttachmentStyle = AttachmentStyle.Item
        child3.Dock = DockStyle.Bottom
        child3.Selectable = False

        Dim child4 As Node = New Node(50, 220, 60, 20, "4: Right")
        child4.Shape.Style = ShapeStyle.Rectangle
        child4.DrawColor = Color.Transparent
        child4.FillColor = Color.LightGray
        child4.AttachmentStyle = AttachmentStyle.Item
        child4.Dock = DockStyle.Right
        child4.Selectable = False

        Dim child5 As Node = New Node(50, 250, 90, 20, "5: Fill")
        child5.Shape.Style = ShapeStyle.Rectangle
        child5.DrawColor = Color.Transparent
        child5.FillColor = Color.LightSlateGray
        child5.AttachmentStyle = AttachmentStyle.Item
        child5.Dock = DockStyle.Fill
        child5.Selectable = False

        ' Add the items to the diagram.
        AddFlow1.Nodes.Add(parent)
        AddFlow1.Nodes.Add(child1)
        AddFlow1.Nodes.Add(child2)
        AddFlow1.Nodes.Add(child3)
        AddFlow1.Nodes.Add(child4)
        AddFlow1.Nodes.Add(child5)

        ' Create the Parent-Child relationships.
        child1.Parent = parent
        child2.Parent = parent
        child3.Parent = parent
        child4.Parent = parent
        child5.Parent = parent
```

If we execute this code, it will create the following diagram:



Au you can see, you children nodes have been placed automatically in the parent node.

## 5.7 Some other information about drawing

AddFlow for WinForms provides also some properties that control the general aspect of a diagram:

| | |
|---|---|
| **AntiAliasing** | Determines whether anti-aliasing graphics rendering technique is used to display the graph. |
| **Images** | Returns the collection of images that can be used to display an image in a node. |
| **OwnerDraw** | Determines whether you want to provide custom drawing for the diagram. |
| **PageScale** | Returns/sets a value that determines the scaling used to display the graph. |
| **PageUnit** | Returns/sets a value that determines the unit of measure used to display the graph. |
| **Zoom** | Returns/sets the horizontal and vertical zooming factors. |

## 5.8 Serialization

There are several methods:

### 5.8.1 The IXmlSerializable method

AddFlow for WinForms version 2 is supporting the **IXmlSerializable** interface. Therefore you can save a diagram in a file using the **WriteXml** method and de-serialize it using the **ReadXml** method. This method is demonstrated in the **afEdit** and **DemoLayout** samples which use also this interface to implement the Cut/Copy/Paste features.

An interesting feature of this method is that it can be customized. Each time a node or a link is serialized or de-serialized, some events are fired, giving the opportunity to save or load custom data.

The serialization events are the following:

- **BeforeReadXMLNode** event. Occurs before a node is de-serialized.
- **BeforeWriteXMLNode** event. Occurs before a node is serialized.
- **BeforeReadXMLLink** event. Occurs before a link is de-serialized.
- **BeforeWriteXMLLink** event. Occurs before a link is serialized.
- **ReadXMLLinkExtraData** event. Occurs when a link is de-serialized, allowing loading custom data.
- **ReadXMLNodeExtraData** event. Occurs when a node is de-serialized, allowing loading custom data.
- **WriteXMLLinkExtraData** event. Occurs when a link is serialized, allowing saving custom data
- **WriteXMLNodeExtraData** event. Occurs when a node is serialized, allowing saving custom data

The use of these events is explained in the paragraph Derivation of Node and Link classes and demonstrated in the **DeriveNode** sample (and also in the **PropertyBag** and **Tables2** samples).

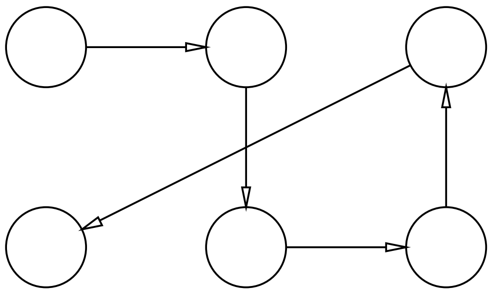
Following are some examples of diagram xml files.

**Example 1**

Here, all property values are the default AddFlow values, therefore the xml file is very small.

```
<?xml version="1.0"?>
<!--AddFlow.net diagram-->
<AddFlow Nodes="6" Links="5">
  <Version>2.1.0.0</Version>
  <Node Index="0" Left="40" Top="40" Width="40" Height="40" />
  <Node Index="1" Left="140" Top="140" Width="40" Height="40" />
  <Node Index="2" Left="140" Top="40" Width="40" Height="40" />
  <Node Index="3" Left="40" Top="140" Width="40" Height="40" />
  <Node Index="4" Left="240" Top="40" Width="40" Height="40" />
  <Node Index="5" Left="240" Top="140" Width="40" Height="40" />
  <Link Index="6" Org="0" Dst="2" />
  <Link Index="7" Org="2" Dst="1" />
  <Link Index="8" Org="1" Dst="5" />
  <Link Index="9" Org="5" Dst="4" />
  <Link Index="10" Org="4" Dst="3" />
</AddFlow>
```

If you load this diagram with the **ReadXml** method you obtain the following diagram.

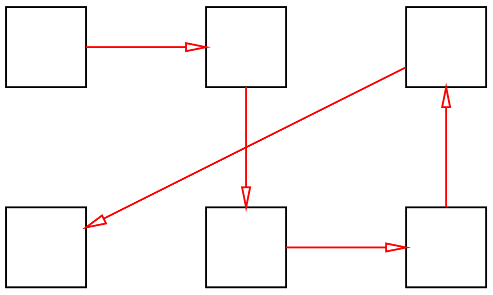


**Example 2**

Here, some default property values are defined for the nodes (Shape) and the links (DrawColor).

```
<?xml version="1.0"?>
<!--AddFlow.net diagram-->
<AddFlow Nodes="6" Links="5">
  <Version>2.1.0.0</Version>
  <DefaultNode>
    <Shape Style="Rectangle" Orientation="so_0" />
  </DefaultNode>
  <DefaultLink>
    <DrawColor>-65536</DrawColor>
  </DefaultLink>
  <Node Index="0" Left="40" Top="40" Width="40" Height="40" />
  <Node Index="1" Left="140" Top="140" Width="40" Height="40" />
  <Node Index="2" Left="140" Top="40" Width="40" Height="40" />
  <Node Index="3" Left="40" Top="140" Width="40" Height="40" />
  <Node Index="4" Left="240" Top="40" Width="40" Height="40" />
  <Node Index="5" Left="240" Top="140" Width="40" Height="40" />
  <Link Index="6" Org="0" Dst="2" />
  <Link Index="7" Org="2" Dst="1" />
  <Link Index="8" Org="1" Dst="5" />
  <Link Index="9" Org="5" Dst="4" />
  <Link Index="10" Org="4" Dst="3" />
</AddFlow>
```

If you load this diagram with the **ReadXml** method you obtain the following diagram.

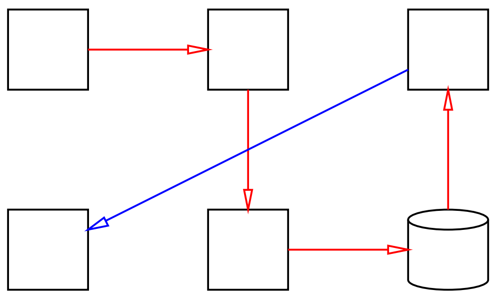


**Example 3**

Here, a node has a property value distinct from the default value. It is also the case for a link.

```
<?xml version="1.0"?>
<!--AddFlow.net diagram-->
<AddFlow Nodes="6" Links="5">
  <Version>2.1.0.0</Version>
  <DefaultNode>
    <Shape Style="Rectangle" Orientation="so_0" />
  </DefaultNode>
  <DefaultLink>
    <DrawColor>-65536</DrawColor>
  </DefaultLink>
  <Node Index="0" Left="40" Top="40" Width="40" Height="40" />
  <Node Index="1" Left="140" Top="140" Width="40" Height="40" />
  <Node Index="2" Left="140" Top="40" Width="40" Height="40" />
  <Node Index="3" Left="40" Top="140" Width="40" Height="40" />
  <Node Index="4" Left="240" Top="40" Width="40" Height="40" />
  <Node Index="5" Left="240" Top="140" Width="40" Height="40">
    <Shape Style="MagneticDisk" Orientation="so_0" />
  </Node>
  <Link Index="6" Org="0" Dst="2" />
  <Link Index="7" Org="2" Dst="1" />
  <Link Index="8" Org="1" Dst="5" />
  <Link Index="9" Org="5" Dst="4" />
  <Link Index="10" Org="4" Dst="3">
    <DrawColor>-16776961</DrawColor>
  </Link>
</AddFlow>
```

If you load this diagram with the **ReadXml** method you obtain the following diagram:

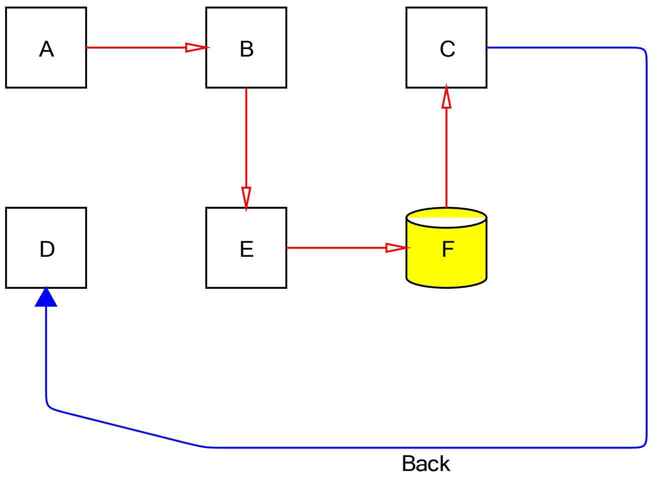


## Example 4

Here, the diagram is a little more complicated.

```
<?xml version="1.0"?>
<!--AddFlow.net diagram-->
<AddFlow Nodes="6" Links="5">
  <Version>2.1.0.0</Version>
  <DefaultNode>
    <Shape Style="Rectangle" Orientation="so_0" />
  </DefaultNode>
  <DefaultLink>
    <DrawColor>-65536</DrawColor>
  </DefaultLink>
  <Node Index="0" Left="40" Top="40" Width="40" Height="40">
    <Text>A</Text>
  </Node>
  <Node Index="1" Left="140" Top="140" Width="40" Height="40">
    <Text>E</Text>
  </Node>
  <Node Index="2" Left="140" Top="40" Width="40" Height="40">
    <Text>B</Text>
  </Node>
  <Node Index="3" Left="40" Top="140" Width="40" Height="40">
    <Text>D</Text>
  </Node>
  <Node Index="4" Left="240" Top="40" Width="40" Height="40">
    <Text>C</Text>
  </Node>
  <Node Index="5" Left="240" Top="140" Width="40" Height="40">
    <Shape Style="MagneticDisk" Orientation="so_0" />
    <FillColor>-256</FillColor>
    <Text>F</Text>
  </Node>
  <Link Index="6" Org="0" Dst="2" />
  <Link Index="7" Org="2" Dst="1" />
  <Link Index="8" Org="1" Dst="5" />
  <Link Index="9" Org="5" Dst="4" />
  <Link Index="10" Org="4" Dst="3">
    <Line Style="Polyline" OrthogonalDynamic="False" RoundedCorner="True"
DoubleLine="False" />
    <ArrowDst Head="Arrow" Size="Small" Angle="deg30" Filled="True" />
    <DrawColor>-16776961</DrawColor>
    <Text>Back</Text>
    <Point X="360" Y="60" />
    <Point X="360" Y="260" />
    <Point X="140" Y="260" />
    <Point X="60" Y="240" />
  </Link>
</AddFlow>
```

If you load this diagram with the **ReadXml** method you obtain the following diagram.



**Remarks about the XML format**

Currently, there is no DTD or schema provided for the AddFlow XML files. However, we can give the following information about the structure of our XML files.

1) In the previous examples, the Node elements are placed before the Link elements. This is not required. This is just because when creating this diagram, the nodes have been created before the links (and their Z-Order position has not been changed after). And because AddFlow serializes the items following the Z-Order position, the nodes have been placed before the links.

2) The number of Node elements must be the same than the **Nodes** attribute of the AddFlow element. The number of Link elements must be the same than the **Links** attribute of the AddFlow element.

3) Inside a Node or a Link element, the subnodes elements ("Text", "FillColor", etc) don't need to appear in a specific order. For instance:

```
  <Node Index="5" Left="240" Top="140" Width="40" Height="40">
    <Shape Style="MagneticDisk" Orientation="so_0" />
    <FillColor>-256</FillColor>
    <Text>F</Text>
  </Node>
```

will give the same output as:

```
  <Node Index="5" Left="240" Top="140" Width="40" Height="40">
    <Text>F</Text>
    <Shape Style="MagneticDisk" Orientation="so_0" />
    <FillColor>-256</FillColor>
  </Node>
```

4) **Attributes**.

- The **Node** element has 5 or 6 attributes. The **Index, Left**, **Top**, **Width** and **Height** attributes are required. The **Custom** attribute is optional.
- The **Link** element has 3 or 4 attributes. The **Index**, **Org** and **Dst** attributes are required. The **Custom** attribute is optional.
-

The **Custom** attribute is described in the paragraph Derivation of Node and Link classes.

5) **Subnodes**.

All the subnodes of the Node and Link elements are optional.

### 5.8.2 The XMLFlow method

You may prefer using the old method based on the **XMLFlow** extension. It is a component, provided with AddFlow, and which allows saving and loading a diagram or a portion of a diagram in a XML stream. You can use XMLFlow to save/load a diagram in a file or to copy/paste a portion of the diagram onto the clipboard or to implement a Drag&Drop feature.

**Remarks**

- This component is free and its C# source code is provided with AddFlow.
- Its use is demonstrated in the **TreeEdit** sample provided with AddFlow.
- We do not provide any support for XMLFlow.
- The method using the IXmlSerializable interface is quicker and it avoids using an additional dll in your project.

### 5.8.3 Your own method

You may also develop your own method. In such a case, the source code of XMLFlow may be a starting point for you.

## 5.9 Printing a diagram

AddFlow itself does not offer any printing feature. However, the printing and the print previewing of AddFlow diagrams can be performed with the **PrnFlow** DLL provided with AddFlow.

This component is free and its C# source code is provided with AddFlow. Its use is demonstrated in the **afEdit** sample provided with AddFlow.

Notice that we do not provide any support for PrnFlow.

Notice also the AddFlow **PageGrid** property which returns/sets a Grid object allowing to set the properties of the grid used to display printing pages.

## 5.10 Exporting the diagram

### 5.10.1 The Render method

The Render method can be used to display the diagram in any GDI+ drawing area. This method could be used for instance to implement a “bird view” window.

### 5.10.2 Metafile support

You can export an AddFlow diagram as a Metafile with the **ExportMetafile** method. The exported image can be saved in a file, copied in a picture box or in the clipboard. This method is demonstrated in the **afEdit** sample provided with AddFlow.

### 5.10.3 SVG support

The **SVGFlow** dll allows exporting an AddFlow diagram in SVG document.

SVG (Scalable Vector Graphics) is a standard for authoring and deploying two-dimensional vector graphics using XML documents. SVGs can be used as an element of a HTML document.

Currently, browsers do not natively support SVG. Therefore, to view an SVG document within web pages, an SVG viewer needs (like the Adobe SVG Viewer or the Corel SVG Viewer) to be downloaded and installed in the client browser.

The following code is all you need to do to export an AddFlow diagram in SVG format.

```
// C#
System.Xml.XmlDocument oDocument =
     Lassalle.Flow.SVG.SVGFlow.FlowToSVG(AddFlow1);
oDocument.Save(fileName);
System.Diagnostics.Process.Start(fileName);

' VB
Dim oDocument As System.Xml.XmlDocument
oDocument = Lassalle.Flow.SVG.SVGFlow.FlowToSVG(AddFlow1)
oDocument.Save(fileName)
System.Diagnostics.Process.Start(fileName)
```

The last line allows running the browser in order to see the SVG file.

# 6 Avanced topics

## 6.1 Undo/Redo

AddFlow provides a powerful multilevel Undo/Redo feature. The history length is limited only by available memory. However, you can limit it yourself with the **UndoSize** property. You can also enable/disable the undo/redo with the **CanUndoRedo** property.

**Updating the user interface**

Some properties and methods allow you to properly update the user interface. The **CanUndo** and **CanRedo** methods will tell you if there is something to undo or redo and therefore will allow you to grey out the menu options. The **RedoCode** and **UndoCode** properties return a code that describes the action waiting to be redone or undone. This will allow your application to give descriptions of the actions on the undo and redo history.

**Grouping basic actions**

Every basic action has a code. However, the **BeginAction** and **EndAction** methods allow you to define a group of actions and to assign a code to this group. This is useful if for instance, in your application, the user can open a dialog box allowing changing several properties of a node (for instance, its text, its shape and its filling color). You will certainly wish to allow the user to undo these 3 basic actions in one time.

Notice that you can also stop recording actions with the **SkipUndo** method and also clear the Undo/Redo buffer with the **ResetUndoRedo** buffer.

Another interesting method is the **AddToLastAction** method. For instance, it allows grouping some actions with the last recorded action or group of actions.. This is demonstrated in the **Tables2** sample: after having resized a node, therefore when receiving the AfterResize event, you wish to adjust the shape of its links. However, you wish to group this reshaping actions with the node resizing action. For that, you use the AddToLastAction method:

```
' VB
AddFlow1.AddToLastAction()
AdjustAllLinkSegments(node)
AddFlow1.EndAction()
```

Notice that you have to call the **EndAction** to terminate the group of actions.

**Undo/Redo customization**

The undo/redo can be customized. For that, you have to create a custom Task class by deriving the Task class and then you can insert it in the undo list with the **SubmitTask** method.

**What can be undone and redone?**

The rule is the following: every action that changes a diagram can be undone or redone. This includes actions like moving or resizing nodes or stretching links or changing a text, a color, a picture or a font. This includes also actions that only change the internal state of the document without having any visible effect, for instance, changing the XMoveable property.

However, making a selection does not change the document so you will not be able to undo a selection. Changing properties of the AddFlow control (zoom, grid, default filling color, etc) does not change the document too. Therefore, it will not be possible to undo these actions. And finally, file, print and export operations are clearly not undoable.

| | |
|---|---|
| **AddToLastAction** | Add the following actions in the last group of actions |
| **BeginAction** | Start a group of actions that can be undone in one time. |
| **CanRedo** | Indicates if there is an action that can be redone. |
| **CanUndo** | Indicates if there is an action that can be undone. |
| **CanUndoRedo** | Determines whether undo/redo is allowed. |
| **EndAction** | Terminate a group of actions that can be undone in one time. |
| **Redo** | Redo, if possible, the last action. |
| **RedoCode** | Returns the code of the next redoable action. |
| **ResetUndoRedo** | Clears the undo/redo buffer. |
| **SkipUndo** | Determines whether the following actions are recorded in the undo manager. |
| **SubmitTask** | Submit a task (or action) that can be undone and redone. |
| **Undo** | Undo, if possible, the last action. |
| **UndoCode** | Returns the code of the next undoable action. |
| **UndoSize** | Sets and returns the number of undo commands that can be performed. |

## 6.2 Performance tuning

To maintain performance while items are added to the AddFlow control, you should call the **BeginUpdate** method. The BeginUpdate method prevents the control from painting until the **EndUpdate** method is called.

Also, between the calls to BeginUpdate and EndUpdate, the size of the diagram is not updated. It is updated only when the EndUpdate method is called. If you need the size of the diagram before the call to EndUpdate, you can use the **GetDiagramSize** method.

The two methods BeginUpdate and EndUpdate are used in the **Stress** sample provided with AddFlow (and also in the **Demo** sample)

**Other tips for better performances**

- Disable the Undo/Redo! If you are creating programmatically a big diagram (as in the **Stress** sample), it is also important to disable the undo/redo with the **CanUndoRedo** property.
- Do not create link jumps! You should use jumps only for small diagrams (less than 200 nodes) because the algorithm used to find intersections requires considerable computational resources. By default however, link jumps are not created (See the **Jump** property of links).

- If you are assigning many property values to nodes and links, don't use the "Old ActiveX" method to create items. (See the paragraph Another way to create the diagram)

## 6.3 Automatic Graph Layout

The primary purpose of an automatic graph layout feature is to offer a way to display graphs or flow charts in a reasonable manner, following some aesthetic rules.

AddFlow does not provide directly any automatic graph layout feature. However, we propose **LayoutFlow**, a set of 5 graph layout components:

| | |
|---|---|
| o **HFlow** | *Hierarchic layout* |
| o **OFlow** | *Orthogonal layout* |
| o **SFlow** | *Symmetric layout* |
| o **SPFlow** | *Series Parallel layout* |
| o **TFlow** | *Tree layout* |

Each of these graph layout components performs a layout on a graph. Performing a layout automatically positions its nodes (also called vertices) and links (also called edges).

Typically, you can first create your nodes and links inside AddFlow, using the AddFlow API, giving each node a random or a (0,0) position. Then you call the layout method of the graph layout control of your choice. This method will position the nodes and the links in a **reasonable** manner in the AddFlow control, following some aesthetic rules that depend on the chosen control (hierarchical with HFlow, symmetric with SFlow, orthogonal with OFlow...).

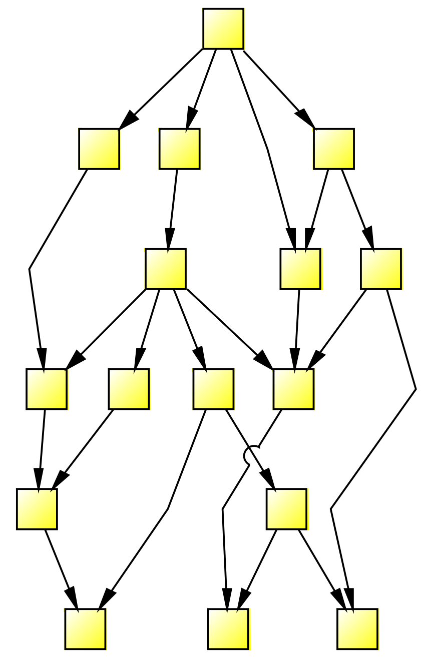
**Remarks**

- Currently, HFlow, OFlow, SFlow, SPFlow and TFlow are AddFlow extensions and you cannot use them without AddFlow. If you just want to perform a layout on a graph without displaying them in an AddFlow control (for instance because you have already a way to display the diagram), then you can use both a hidden AddFlow control and the graph layout control, for instance HFlow, to do that. In such a case, AddFlow is just used to store the logical structure of the graph and to retrieve via its API, the resulting positions of its nodes and links.
- The **DemoLayout** sample installed with AddFlow shows how to use each graph layout component.
- Only the **Logical** nodes and links of the AddFlow control are involved in each layout. This will allow you to apply the layout only to important nodes. For instance, you can exclude a node just used to display a label by setting its **Logical** property to false.
- Reflexive links are not taken into account by layout algorithms. Reflexive links are just translated to follow their origin (and also destination) node.

### 6.3.1 HFlow (Hierarchic layout)

#### *6.3.1.1 Purpose*

HFlow for WinForms is a component that performs a hierarchical layout on a graph. The hierarchical layout arranges vertices in horizontal layers. The order of the nodes on the layers is chosen so that the number of crossings is kept as small as possible.



**- HFlow layout -**

#### *6.3.1.2 Code example*

The following VB code is all you need to do to perform a hierarchical layout:

```
Dim hflow As HFlow = New HFlow()
hflow.LayerDistance = 50  ' Sets the distance between adjacent levels
hflow.VertexDistance = 50 ' Sets the distance between adjacent nodes
hflow.Orientation = North
hflow.LayerWidth = 0      ' No limitation in the number of nodes in a level

hflow.Layout(AddFlow1)    ' Perform the hierarchical layout
```

This code supposes that you have a form containing an AddFlow control. You create the graph in the AddFlow control, either interactively, either programmatically (in this case, giving each node a random position or a (0,0) position). Then you apply the HFlow layout to this graph. And each bode will be placed at a reasonable position.

#### *6.3.1.3 Limitation*

HFlow works with any graph, connected or not.

#### *6.3.1.4 Side Effect*

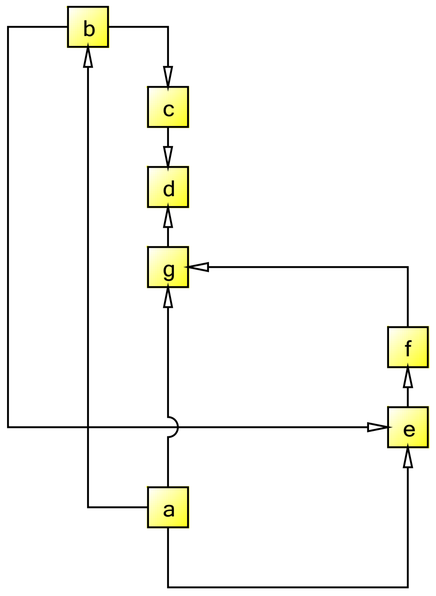
After the layout execution:

- the line style of the links is Polyline
- the AdjustOrg and AdjustDst properties are true.

### 6.3.2 OFlow (Orthogonal layout)

#### *6.3.2.1 Purpose*

OFlow for WinForms is a component that performs an orthogonal layout on a graph. The layout is orthogonal since it produces an orthogonal drawing where each link is drawn as a polygonal chain of alternating horizontal and vertical segments. The algorithm used is the Biedl and Kant algorithm.



**- OFlow layout -**

#### *6.3.2.2 Code example*

The following VB code is all you need to do to perform an orthogonal layout:

```
Dim oflow As OFlow = New OFlow()
oflow.xGrid = 50        ' Sets the grid size
oflow.yGrid = 50
oflow.Orientation = North
oflow.Layout(AddFlow1) ' Perform the orthogonal layout
```

#### *6.3.2.3 Limitation*

OFlow works with any graph, connected or not.

Note however that this algorithm is making generous use of space and the resulting layout is good only with small graphs.

#### *6.3.2.4 Side Effect*

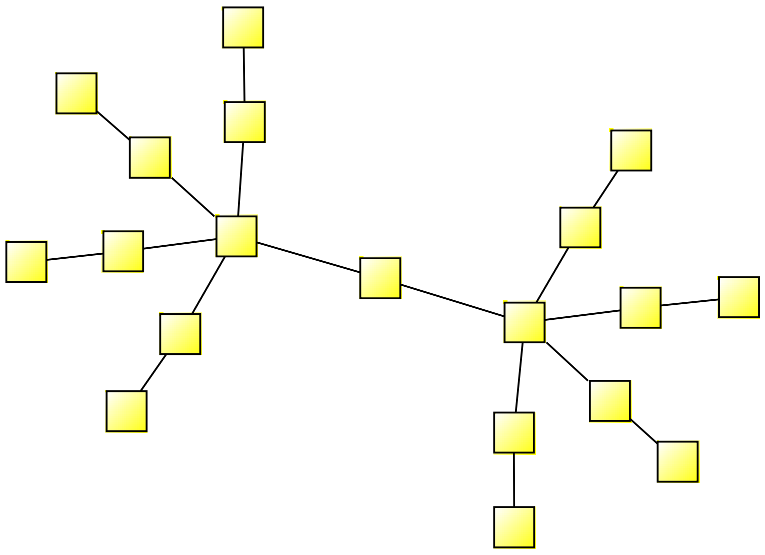
After the layout execution:

- the size of the nodes is changed. If the graph is a graph of maximum degree four, then each node has the same size (determined by the **GridSize** property). If the degree of a node is higher than four, then the height of the node is expanded.
- the line style of the links is Polyline.
- the AdjustOrg and AdjustDst properties are true.

### 6.3.3 SFlow (Symmetric layout)

#### *6.3.3.1 Purpose*

SFlow is a component that performs a symmetric layout on a graph. This layout produces a high degree of symmetry and is particularly useful for undirected graphs, where the directions of the links are not important. SFlow is using a force-directed algorithm (the GEM method of Frick, Ludwig and Mehldau) where a graph is viewed as a system of bodies with forces acting between the bodies.



**- SFlow layout -**

#### *6.3.3.2 Code example*

The following VB code is all you need to do to perform a symmetric layout on a graph:

```
Dim sflow As SFlow = New SFlow()
sflow.Distance = 50       ' Sets the distance between nodes
sflow.Layout(AddFlow1)    ' Perform the symmetric layout
```

If the **Animation** property is true, then you can see how the layout is working. It is just for fun.

#### *6.3.3.3 Limitation*

SFlow works with any graph, connected or not. However, it is recommended to work only with small graphs (less than 200 nodes) because SFlow is using a force-directed method and force-directed methods are using considerable computational resources.

#### *6.3.3.4 Side Effect*

After the layout execution:

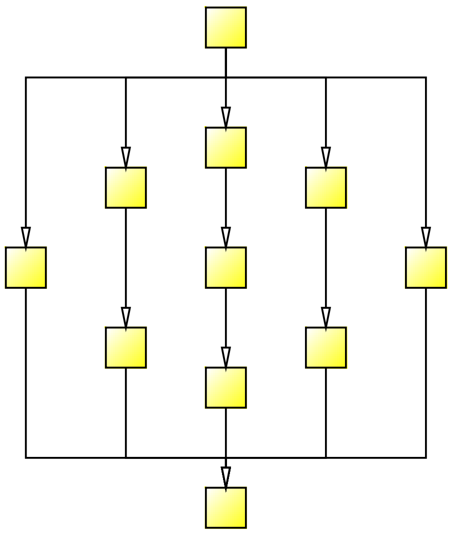
- the line style of the links is Polyline and each link is composed of only one segment.
- the AdjustOrg and AdjustDst properties are true.

### 6.3.4 SPFlow (Series-parallel layout)

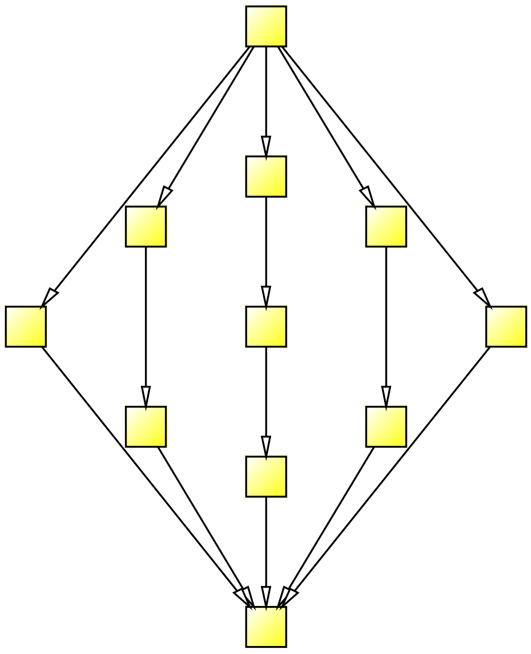
#### *6.3.4.1 Purpose*

SPFlow is a component that performs a series-parallel layout on a graph. The SP layout applies only to a specific subset of graphs: series-parallel digraph (more precisely, a set of series-parallel diagraphs). A series-parallel digraph is defined recursively as follows.
A digraph consisting of two nodes, a source s and a sink t joined by a single link is a series-parallel digraph.
If G1 and G2 are series-parallel digraphs, so are the digraphs constructed by each of the following operations:
- the parallel composition: identify the source of G1 with the source of G2 and the sink of G1 with the sink of G2.
- the series composition: identify the sink of G1 with the source of G2.

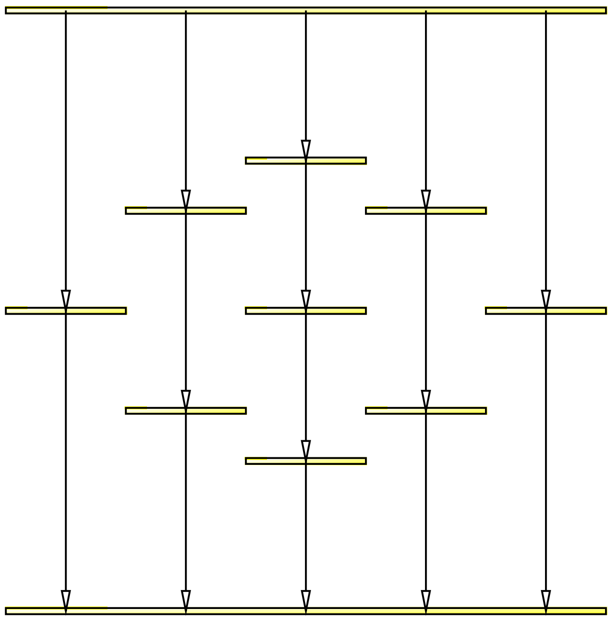
We use an algorithm (described in the book "Drawing Graphs" Michael Kaufmann - Dorothea Wagner) that allows drawing series-parallel digraphs with as much symmetry as possible.



**- SPFlow layout: DrawingStyle = BusOrthogonalDrawing**



**- SPFlow layout: DrawingStyle = StraightLine -**



**- SPFlow layout: DrawingStyle = VisibilityDrawing -**

### *6.3.4.2 Code example*

The following VB code is all you need to do to perform a series-parallel layout on a graph:

```
Dim spflow As SPFlow = New SPFlow()
spflow.LayerDistance = 80    ' Sets the distance between adjacent levels
spflow.VertexDistance = 80  ' Sets the distance between adjacent nodes
spflow.Orientation = Orientation.North
spflow.DrawingStyle = DrawingStyle.BusOrthogonalDrawing
spflow.Layout(AddFlow1)      ' Perform the series-parallel layout
```

If the graph is not a set of series-parallel digraph, an exception is generated.

### *6.3.4.3 Limitation*

The layout applies only to a specific subset of graphs: series-parallel digraphs. One of the requirements is that this diagram has only one starting node and only one ending node. However, it is not actually a limitation. If, for instance, the number of ending nodes is greater than one, then a workaround is to create a dummy node and create a link from each ending node to this dummy node, then execute the layout and then delete the dummy node (which causes all the dummy links to be deleted too).

### *6.3.4.4 Side Effect*

After the layout execution:

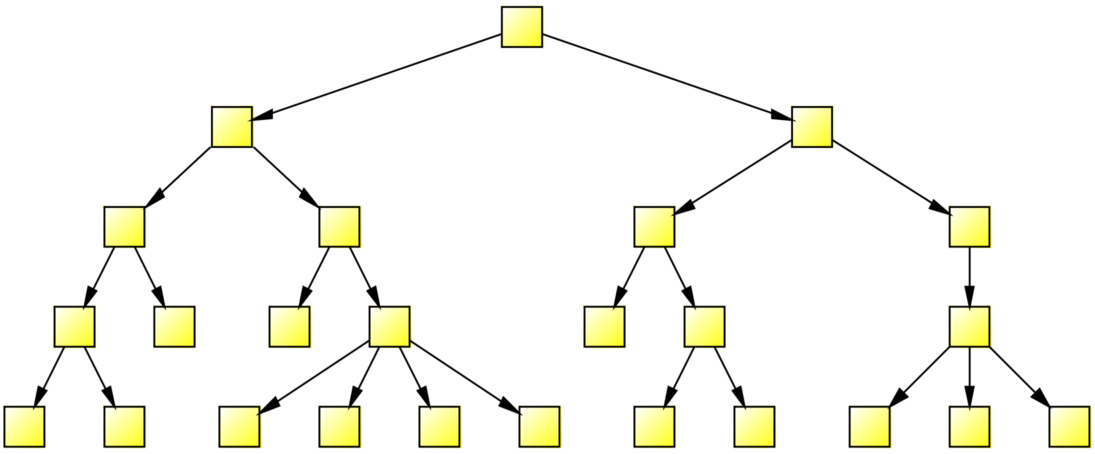
- the line style of the links is Polyline. Moreover, if the DrawingStyle property is not BusOrthogonalDrawing, then each link is composed of only one segment.
- the AdjustOrg and AdjustDst properties are true.

### 6.3.5 TFlow (Tree layout)

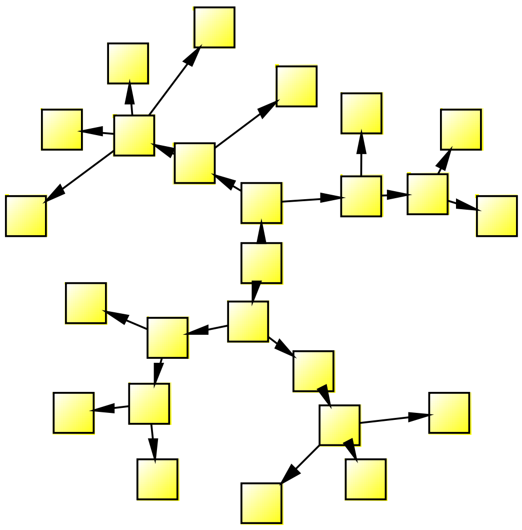
#### *6.3.5.1 Purpose*

TFlow is a component that performs a tree layout on a graph. This layout applies only to a specific subset of graphs: rooted trees. In such a graph, no node may have more than one parent. TFlow offers two drawing styles (**DrawingStyle** property).

- If the DrawingStyle is **Layered**, then the drawing of the tree occupies as little space as possible while satisfying certain aesthetics: nodes at the same level of the tree are placed on the same line and a parent is centred over its children.
- If the DrawingStyle is **Radial**, then the root of the tree is placed at the origin and the layers are concentric circles centred at the origin.



**- TFlow layout: DrawingStyle = Layered -**



**- TFlow layout: DrawingStyle = Radial -**

#### *6.3.5.2 Code example*

The following VB code is all you need to do to perform a tree layout on a graph:

```
Dim tflow As TFlow = New TFlow()
tflow.LayerDistance = 1000  ' Sets the distance between adjacent levels
tflow.VertexDistance = 1000 ' Sets the distance between adjacent nodes
tflow.Orientation = Orientation.North
tflow.Layout(AddFlow1)      ' Perform the tree layout
```

If the graph is not a forest of rooted trees, an exception is generated.

#### *6.3.5.3 Limitation*

The layout applies only to a specific subset of graphs: rooted trees. More precisely, the layout applies to forests (sets of rooted trees).

#### *6.3.5.4 Side Effect*

After the layout execution:

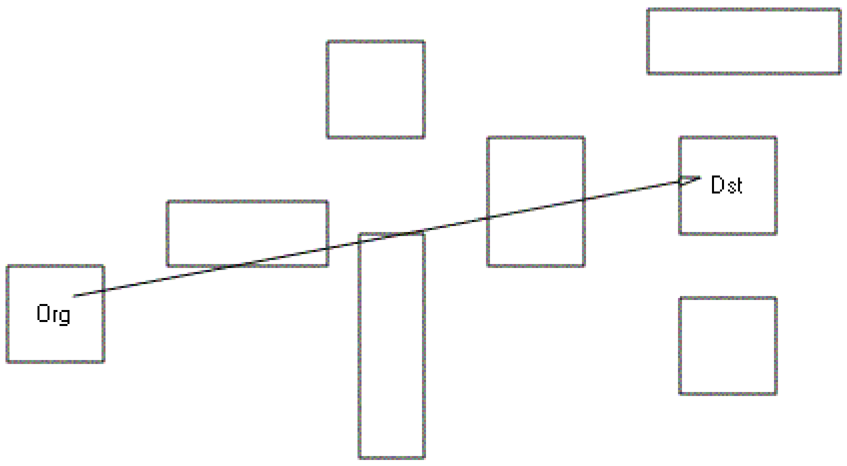
- the line style of the links is Polyline
- the AdjustOrg and AdjustDst properties are true.

## 6.4 Link auto-routing

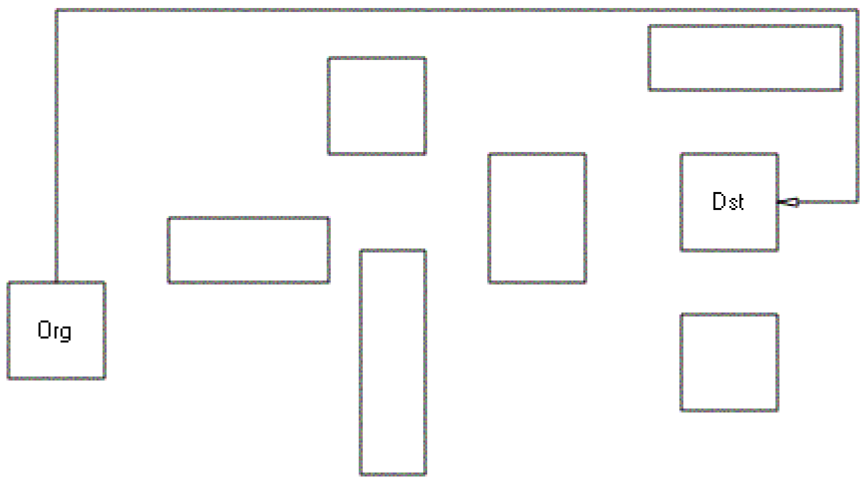
### 6.4.1 Introduction

The **RouteFlow** dll allows link auto-routing. It is an AddFlow extension and you cannot use it without AddFlow.

By default, when the user creates a link between 2 nodes, the link is drawn as a straight line which may traverse other nodes, as in the following example.



The RouteFlow algorithm allows finding a route between the other nodes which are considered as obstacles, as illustrated in the following picture.



### 6.4.2 Method

RouteFlow uses the following rules:

**1) Orthogonality rule**

The path is composed of orthogonal segments. A new segment is always orthogonal to the previous one.

**2) Distance rule** (we try to decrease the distance towards the destination node) When we move in a segment:

- If the distance towards the destination node is increasing, then we stop the current segment and create a new segment allowing decreasing the distance towards the destination node.
- Else we continue until:
    - Either we hit an obstacle (another node)
    - Either we reach the X or Y coordinates of the destination.

**3) Obstacle rule**

- If we hit an obstacle, we backtrack to the previous segment and we continue to move.
- If this is not possible, we move back one step and we create a new segment.

**4) Start rule**

The first segment is orthogonal to a side of the origin node rectangle. Several methods to select the starting side are provided via the **StartingMethod** property.

### 6.4.3 Code sample

The following VB code is all you need to do to perform a route finding for a link. This code should be included in the handler of the AfterAddLink event of AddFlow.

```
' VB
Dim routeFlow As Lassalle.Flow.Router.RouteFlow
routeFlow = New Lassalle.Flow.Router.RouteFlow
routeFlow.StartingMethod = Lassalle.Flow.Router.StartingMethod.AllSides
routeFlow.Grain = 8
routeFlow.MinDistance = 16
routeFlow.FindRoute(e.Link)

// C#
Lassalle.Flow.Router.RouteFlow routeFlow = new
                             Lassalle.Flow.Router.RouteFlow();
routeFlow.StartingMethod = Lassalle.Flow.Router.StartingMethod.AllSides;
routeFlow.Grain = 8;
routeFlow.MinDistance = 16;
routeFlow.FindRoute(e.Link);
```

The **RouteLink** sample provided with AddFlow demonstrates the link auto-routing performed by RouteFlow. It may provide a little animation, using the **Step** event (If the **SendStepEvent** property is true, the **Step** event is sent at each step of the algorithm: this feature is used by the RouteLink sample).

### 6.4.4 Limitations

There are some limitations. The auto-routing is performed only if:

- The link is not reflexive
- The line style is Polyline
- The AdjustDst and AdjustOrg properties are true.

Moreover, only logical nodes can be obstacles.

**And finally, in some cases (pathological diagrams), the auto-routing algorithm may fail to provide a correct solution.** In such cases, it will just draw only one segment joining the origin and destination nodes.

## 6.5 Customization

### 6.5.1 Overview

There are several ways to customize AddFlow and you can customize the AddFlow behaviour, the AddFlow drawings and also the data associated to nodes and links.

*a) Behaviour customization*

- **AddFlow capabilities**. AddFlow offers you a set of properties allowing customizing its behavior.
- **AddFlow class derivation**.

*b) Drawing customization*

- **Custom shapes**. You can customize the AddFlow drawings by using the possibility to create custom shapes for nodes and links.
- **OwnerDraw property**.You can also customize the AddFlow drawings using the OwnerDraw property.

*c) Data customization*

There are 3 ways to associate custom data to a node or link:
- **Tag property**. You can use it to associate an object to a node or a link
- **Property bag**. You can use it to easily add new custom properties to nodes and links.
- **Derivation of Node and Link classes**.

### 6.5.2 Behavior customization

#### *6.5.2.1 AddFlow capabilities*

Following properties allow to set capabilities for an AddFlow control and therefore to customize it. For instance, if you wish to allow only one link between two nodes, you have just to unset the **CanMultiLink** property.

| | |
|---|---|
| **CursorSetting** | Determines whether default cursors are displayed. |
| **CanChangeDst** | Determines whether the user can interactively change the destination of a link. |
| **CanChangeOrg** | Determines whether the user can interactively change the origin of a link. |
| **CanDragScroll** | Determines whether drag scrolling is allowed or not. |
| **CanDrawNode** | Determines whether interactive creation of nodes is allowed or not. |
| **CanDrawLink** | Determines whether interactive creation of links is allowed or not. |
| **CanFireError** | Determines if the error event is fired or not. |

| | |
|---|---|
| **CanLabelEdit** | Determines whether the user can edit the text of nodes. |
| **CanReflexLink** | Determines whether interactive creation of reflexive links is allowed or not. |
| **CanMoveNode** | Determines whether interactive dragging of nodes  is allowed or not. |
| **CanSizeNode** | Determines whether interaction resizing of nodes  is allowed or not. |
| **CanStretchLink** | Determines whether interactive stretching of links  is allowed or not. |
| **CanMultiLink** | Determines whether you can create several links between two nodes. |
| **CanUndoRedo** | Determines whether undo/redo is allowed. |
| **CycleMode** | Determines whether cycles are accepted or not. |
| **DefLinkProp** | Defines the default property values for links. |
| **DefNodeProp** | Defines the default property values for nodes. |
| **DisplayHandles** | Determines whether the handles used for selection are displayed or not. |
| **Grid** | Returns/sets a Grid object allowing to set the properties of the grid. |
| **JumpSize** | Returns/sets the size of the jumps at the intersection of links. |
| **LinkCreationMode** | Used to specify how the user can interactively create a link. |
| **LinkHandleSize** | Defines the size of the linking handle at the center of selected node. |
| **LinkSelectionAreaWidth** | Ddetermines the width of the area where the user has to click to select a link. |
| **MultiSel** | Determines whether multiselection of nodes is allowed or not. |
| **OwnerDraw** | Determines whether you want to provide custom drawing for the diagram. |
| **RemovePointAngle** | Returns/sets the angle that causes a link point to be removed when stretching a link. |
| **RoundedCornerSize** | Returns/sets the size of the rounded corners of the link segments. |
| **SelectionHandleSize** | Defines the size of the selection handles of the selected node. |

#### *6.5.2.2 Deriving the AddFlow class*

Using inheritance, you can create a new class by adding to or otherwise modifying an existing class. Of course we can do that. In the following example, we create a new class derived from the AddFlow class where nodes have a fixed size.

Using Visual Studio .NET, select the menu item **File | Add Project | New Project**. In the Add New Project dialog box, select a project type of **Visual C# Projects**, then choose **Empty Project**. Then, right-click the project name in Solution Explorer and select **Add New Item** from the context menu. In the Add New Item dialog box, in the **Categories** list, choose **Local Project Items**. In the **Template** section, choose **Code File** (Using this template, Visual Studio will not generate code for you).

Right-click in the References item underneath the project name and select Add Reference from the context menu. Select these three items from the list in the dialog box that you're presented with: Sytem.dll, System.Drawing.dll, System.Windows.Forms.dll. Then, use the Browse... button to search for the Lassalle.Flow.dll and select it.

Finally, copy and paste the following C# code:

```
// C#
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using Lassalle.Flow;

namespace DeriveFlow
{
  public class DeriveFlow : System.Windows.Forms.Form
  {
    public static void Main()
    {
      Application.Run(new DeriveFlow());
    }

    public DeriveFlow()
    {
      Text = "DeriveFlow";
      Size = new Size(750, 550);
      MyFlow myflow = new MyFlow();
      myflow.Parent = this;
    }
  }

  internal class MyFlow : AddFlow
  {
    public MyFlow()
    {
      BackColor = Color.White;
      Dock = DockStyle.Fill;
      AutoScroll = true;
      CanSizeNode = false;
    }

    protected override void OnAfterAddNode(AfterAddNodeEventArgs e)
    {
        e.Node.Size = new SizeF(30, 30);
    }
  }
}

' VB
Imports Lassalle.Flow

Public Class DeriveFlow
    Private Sub DeriveFlow_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Text = "DeriveFlow"
        Size = New Size(750, 550)
        Dim myflow As MyFlow = New MyFlow()
        myflow.Parent = Me
    End Sub
End Class

Public Class MyFlow
    Inherits AddFlow

    Public Sub New()
        BackColor = Color.White
        Dock = DockStyle.Fill
        AutoScroll = True
        CanSizeNode = False
    End Sub

    Protected Overrides Sub OnAfterAddNode(ByVal e As
Lassalle.Flow.AfterAddNodeEventArgs)
        e.Node.Size = New SizeF(30, 30)
    End Sub
End Class
```

The MyFlow class is derived from the AddFlow class. In its constructor, we set four properties. The first 3 settings are usual when using AddFlow. The fourth setting (`CanSizeNode = false`) disable the possibility to resize nodes. Moreover, we override the method OnAfterAddNode to force the size of nodes to be limited to 30 in width and also in heigth.

### 6.5.3 Drawing customization

#### *6.5.3.1 Custom Shapes*

We have seen that the shape of the node is determined by the Shape property of the node and more precisely by the Style property of the Shape object associated to a node. For instance:

```
node2.Shape.Style = ShapeStyle.Rectangle;
```

As indicated in the help file, the ShapeStyle enumeration has 44 predefined shapes. You can view each of them with the *Shapes* sample provided with AddFlow. One of these styles is *ShapeStyle.Custom*. If you set the Style property of node Shape object equal to *ShapeStyle.Custom*, and if you use the GraphicsPath property, you can associate a custom shape to a node.

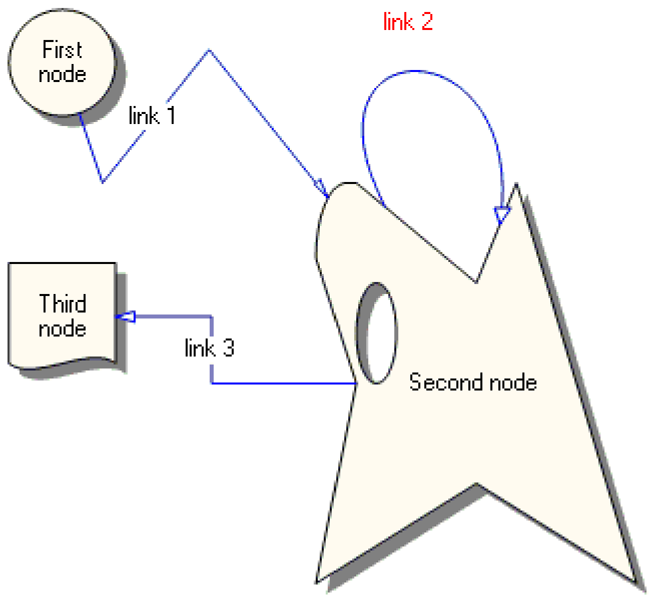
The following CreateDiagram8 method gives an example.

```
// C#
void CreateDiagram8(AddFlow addflow)
{
  // Create a graphics path which will be used for the custom shape
  GraphicsPath path = new GraphicsPath();
  path.AddArc(0, 0, 15, 15, 180, 90);
  path.AddLine(10, 0, 40, 10);
  path.AddLine(40, 10, 50, 0);
  path.AddLine(50, 0, 80, 40);
  path.AddLine(80, 40, 40, 30);
  path.AddLine(40, 30, 0, 40);
  path.AddLine(0, 40, 10, 20);
  path.CloseFigure();
  path.AddEllipse(10,10,10,10);
  path.FillMode = FillMode.Alternate;

  Node dn = (Node)addflow.DefNodeProp.Clone();
  Link dl = (Link)addflow.DefLinkProp.Clone();

  // Default property values for nodes created programmatically
  dn.FillColor = Color.LightYellow;
  dn.Shadow.Style = ShadowStyle.RightBottom;

  // Default property values for links created programmatically
  dl.DrawColor = Color.Blue;
  dl.BackMode = BackMode.Opaque;

  // Create 3 nodes and assign them some property values
  Node node1 = new Node(5, 5, 40, 40, "First node", dn);

  Node node2 = new Node(120, 70, 120, 150, "Second node", dn);
  node2.Shape.Style = ShapeStyle.Custom;
  node2.Shape.GraphicsPath = path;

  Node node3 = new Node(5, 100, 40, 40, "Third node", dn);
  node3.Shape.Style = ShapeStyle.Document;

  // Create 3 links
  Link link1 = new Link("link 1", dl);

  Link link2 = new Link("link 2", dl);
  link2.ArrowDst.Angle = ArrowAngle.deg30;
  link2.Line.Style = LineStyle.Bezier;
  link2.TextColor = Color.Red;

  Link link3 = new Link("link 3", dl);
  link3.Line.Style = LineStyle.HVH;

  // Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1);
  addflow.Nodes.Add(node2);
  addflow.Nodes.Add(node3);
  node1.OutLinks.Add(link1, node2);
  node2.OutLinks.Add(link2, node2);
  node2.OutLinks.Add(link3, node3);

  // Add 2 points (therefore 2 segments) to the first link
  link1.Points.Add(new PointF(40, 70));
  link1.Points.Add(new PointF(80, 20));

  // Stretch the reflexive link
  link2.Points[1] = new PointF(110, 10);
  link2.Points[2] = new PointF(200, 10);
}

' VB
Private Sub CreateDiagram8(ByVal addflow As AddFlow)
  ' Create a graphics path which will be used for the custom shape
  Dim path As GraphicsPath = New GraphicsPath()
  path.AddArc(0, 0, 15, 15, 180, 90)
  path.AddLine(10, 0, 40, 10)
  path.AddLine(40, 10, 50, 0)
  path.AddLine(50, 0, 80, 40)
  path.AddLine(80, 40, 40, 30)
  path.AddLine(40, 30, 0, 40)
  path.AddLine(0, 40, 10, 20)
  path.CloseFigure()
  path.AddEllipse(10, 10, 10, 10)
  path.FillMode = FillMode.Alternate

  Dim dn As Node = addflow.DefNodeProp.Clone()
  Dim dl As Link = addflow.DefLinkProp.Clone()

  ' Default property values for nodes created programmatically
  dn.FillColor = Color.LightYellow
  dn.Shadow.Style = ShadowStyle.RightBottom

  ' Default property values for links created programmatically
  dl.DrawColor = Color.Blue
  dl.BackMode = BackMode.Opaque

  ' Create 3 nodes and assign them some property values
  Dim node1 As Node = New Node(5, 5, 40, 40, "First node", dn)

  Dim node2 As Node = New Node(120, 70, 50, 40, "Second node", dn)
  node2.Shape.Style = ShapeStyle.Custom
  node2.Shape.GraphicsPath = path

  Dim node3 As Node = New Node(5, 100, 40, 40, "Third node", dn)
  node3.Shape.Style = ShapeStyle.Document

  ' Create 3 links
  Dim link1 As Link = New Link("link 1", dl)

  Dim link2 As Link = New Link("link 2", dl)
  link2.ArrowDst.Angle = ArrowAngle.deg30
  link2.Line.Style = LineStyle.Bezier
  link2.TextColor = Color.Red

  Dim link3 As Link = New Link("link 3", dl)
  link3.Line.Style = LineStyle.HVH

  ' Add the nodes and the links to the diagram
  addflow.Nodes.Add(node1)
  addflow.Nodes.Add(node2)
  addflow.Nodes.Add(node3)
  node1.OutLinks.Add(link1, node2)
  node2.OutLinks.Add(link2, node2)
  node2.OutLinks.Add(link3, node3)

  ' Add 2 points (therefore 2 segments) to the first link
  link1.Points.Add(New PointF(40, 70))
  link1.Points.Add(New PointF(80, 20))

  ' Stretch the reflexive link
  link2.Points(1) = New PointF(110, 10)
  link2.Points(2) = New PointF(200, 10)
End Sub
```

Notice that it is necessary to add the following line at the beginning of the program:

```
using System.Drawing.Drawing2D;
```

If we compile and execute this program, it will create the following diagram (We have increased the size of the second node):


link 2
First node
link 1
Third node
link 3
Second node

#### *6.5.3.2 OwnerDraw property*

Even though there are many properties offering a lot of control over the appearance of nodes and links (colors, font, styles, etc), sometimes that is not enough. You may want to use a gradient background inside a node, or draw some custom graphics (for instance swim lines) directly in the background of the AddFlow control.

In these cases, you can use the "OwnerDraw" properties and events to gain total control over how each item is drawn. You can customize the drawing of a node, a link or the entire AddFlow control.

| **Object** | **Property** | **Event** | **Event parameter** |
|---|---|---|---|
| AddFlow | OwnerDraw | DiagramOwnerDraw | DiagramOwnerDrawEventArgs |
| Node | OwnerDraw | NodeOwnerDraw | NodeOwnerDrawEventArgs |
| Link | OwnerDraw | LinkOwnerDraw | LinkOwnerDrawEventArgs |

If the OwnerDraw property is true, then the corresponding event is fired each time the object needs to be redrawn, giving the possibility to replace the default drawing made by AddFlow by a custom drawing. Each event parameter class contains a **Flags** property which specifies how the drawing is made.

This is demonstrated in the **OwnerDraw** sample provided with AddFlow. For instance, following is the code used to draw a node with a hatched background:

```
' VB
Private Sub AddFlow1_NodeOwnerDraw(ByVal sender As System.Object, ByVal e
As Lassalle.Flow.NodeOwnerDrawEventArgs) Handles AddFlow1.NodeOwnerDraw
    Dim grfx As Graphics = e.Graphics
    Dim node As Node = e.Node

    ' Save the graphics state because it does not belong to us
    ' (although in this case, it not necessary because we do not
    ' alter the Graphics state)
    Dim gs As GraphicsState = grfx.Save()

    Dim hBrush1 As HatchBrush = New HatchBrush(HatchStyle.BackwardDiagonal,
Color.White, Color.Silver)
    grfx.FillRectangle(hBrush1, node.Rect)

    ' Restore the graphics state
    grfx.Restore(gs)

    ' Tell AddFlow not to fill the node (because this is already done)
    e.Flags = Not NodeDrawFlags.Fill
End Sub

// C#
private void node_OwnerDraw(object sender, NodeOwnerDrawEventArgs e)
{
    Graphics grfx = e.Graphics;
    Node node = e.Node;
    // Save the graphics state because it does not belong to us
    // (although in this case, it not necessary because we do not
    // alter the Graphics state)
    GraphicsState gs = grfx.Save();


    HatchBrush hBrush1 = new HatchBrush(HatchStyle.BackwardDiagonal,
                                        Color.White, Color.Silver);
    grfx.FillRectangle(hBrush1, node.Rect);

    // Restore the graphics state
    grfx.Restore(gs);

    // Tell AddFlow not to fill the node (because this is already done)
    e.Flags = e.Flags & ~(NodeDrawFlags.Fill);
}
```

Of course, your program should have the following line of code:

```
addflow.NodeOwnerDraw +=
                   new AddFlow.NodeOwnerDrawEventHandler(node_OwnerDraw);
```

## 6.5.4 Data customization

### *6.5.4.1 Tag property*

The **Tag** property allows associating an object to a node or a link.

### *6.5.4.2 Property bag*

The property bag allows extending the functionality of nodes and link by adding new properties. You can do that with the **Properties** property.

For instance, if you wish to add a new property "Author" to a node and assign the value "Alice" to this property, you can do that using a single line of code:

```
' VB
node.Properties("Author").Value = "Alice"


// C#
node.Properties["Author"].Value = "Alice";
```

In this example, this property is a string but it could be any kind of objects.

The Property bag feature is demonstrated in the **PropertyBag** sample provided with AddFlow.

This sample shows also how to save and load these custom data in a XML file, using the **ReadXMLNodeExtraData** and **WriteXMLNodeExtraData** events.

### *6.5.4.3 Derivation of Node and Link classes*

The **DeriveNode** sample application provided with AddFlow shows how to create a new class MyNode, derived from the Node class, and how to save and load these custom data in a XML file, using the following events:

- **BeforeReadXMLNode**
- **BeforeWriteXMLNode**
- **ReadXMLNodeExtraData**
- **WriteXMLNodeExtraData**

In this paragraph, we are going to describe how to use these four events, using the DeriveNode sample.

Of course the same kind of events can be used for the links:

- **BeforeReadXMLLink**
- **BeforeWriteXMLLink**
- **ReadXMLLinkExtraData**
- **WriteMLLinkExtraData.**

#### *6.5.4.3.1 The derived class*

The class MyNode, derived from the Node class, contains two new string properties "Author" and "Comment".

```
' VB
Public Class MyNode
    Inherits Node

    Private m_author As String = Nothing
    Private m_comment As String = Nothing

    Public Sub New(ByVal defnode As Node, ByVal author As String, ByVal
comment As String)
        MyBase.New(defnode)
        m_author = author
        m_comment = comment
    End Sub

    Public Property Author() As String
        Get
            Return m_author
        End Get
        Set(ByVal Value As String)
            m_author = Value
        End Set
    End Property

    Public Property Comment() As String
        Get
            Return m_comment
        End Get
        Set(ByVal Value As String)
            m_comment = Value
        End Set
    End Property
End Class

// C#
internal class MyNode : Node
{
    private string m_author = null;
    private string m_comment = null;

    public MyNode(Node defnode, string author, string comment) :
                                                          base(defnode)
    {
        m_author = author;
        m_comment = comment;
    }

    public string Author
    {
        get { return m_author; }
        set { m_author = value; }
    }

    public string Comment
    {
        get { return m_comment; }
        set { m_comment = value; }
    }
}
```

### *6.5.4.3.2 Interactive creation of of a derived node*

The user creates interactively (with the mouse) a MyNode object exactly as he would to create a node. To implement that, we use the **BeforeAddNode** event which is fired just before a node is created interactively (or also programmaticaly if the InteractiveEventsOnly property is false).

In the handler of this event, the following tasks are done:

- Cancel the node creation
- Instead of creating a node, we create a MyNode object. However, as the MyNode class inherits form the Node class, our MyNode object is also a Node object.
- The MyNode object is placed at the same place as the node whose creation has been aborted.
- The MyNode object is then added to the diagram.

The code is the following:

```
' VB
Private Sub AddFlow1_BeforeAddNode(ByVal sender As System.Object, ByVal e
As Lassalle.Flow.BeforeAddNodeEventArgs) Handles AddFlow1.BeforeAddNode
    e.Cancel.Cancel = True
    Dim mynode As MyNode = New MyNode(AddFlow1.DefNodeProp,Nothing,Nothing)
    mynode.Location = e.Location
    mynode.Size = e.Size
    AddFlow1.Nodes.Add(mynode)
    mynode.Selected = True
End Sub

// C#
private void AddFlow1_BeforeAddNode(object sender,BeforeAddNodeEventArgs e)
{
    e.Cancel.Cancel = true;
    MyNode mynode = new MyNode(AddFlow1.DefNodeProp, null, null);
    mynode.Location = e.Location;
    mynode.Size = e.Size;
    AddFlow1.Nodes.Add(mynode);
    Mynode.Selected = true;
}
```

Notice however that the **BeforeAddNode** event is not fired during a de-serialization process. In this last case, you should use the **BeforeReadXMLNode** event (described later).

#### *6.5.4.3.3 Add Custom data*

In the DeriveNode sample, when the user double click on a node (or click on the "**<u>N</u>ode custom data…**" menu) and if this node is in fact a MyNode object (which is always the case in our DeriveNode sample), a dialog box is displayed to allow entering the MyNode object custom data (here the "Author" and the Comment" strings).

```
' VB
Private Sub EnterCustomData()
    If TypeOf(AddFlow1.SelectedItem) Is Node Then
        Dim node As Node = AddFlow1.SelectedItem
        If (TypeOf (node) Is MyNode) Then
            Dim mynode As MyNode = node
            Dim nd As NodeData = New NodeData
            nd.MyNode = mynode
            If nd.ShowDialog() = Windows.Forms.DialogResult.OK Then
                AddFlow1.SubmitTask(New MyTask(mynode))
                mynode.Author = nd.TextBoxAuthor.Text
                mynode.Comment = nd.textBoxComment.Text
            End If
        End If
    End If
End Sub
```

```
// C#
private void EnterCustomData()
{
    if (AddFlow1.SelectedItem is Node)
    {
        Node node = (Node)AddFlow1.SelectedItem;
        if (node is MyNode)
        {
            MyNode mynode = (MyNode)node;
            NodeData nd = new NodeData();
            nd.MyNode = mynode;
            if (nd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                AddFlow1.SubmitTask(new MyTask(mynode));
                mynode.Author = nd.TextBoxAuthor.Text;
                mynode.Comment = nd.textBoxComment.Text;
            }
        }
    }
}
```

(Notice the call to **SubmitTask** to include the custom data change in the undo/redo buffer. You can find the code of the MyTask class in the DeriveNode sample code)

#### *6.5.4.3.4 Serialization of derived nodes*

We have to serialize the type of nodes and the custom data.

How to inform AddFlow of your custom type in the serialization process? You have to use the **BeforeWriteXMLNode** event. It is sent just before serializing a node. It allows saving the value of an attribute named “Custom”. You have to just to pass its value to AddFlow. In our case, we pass the type of the node (if this type is not the default "Node" type). AddFlow will save this type in the “Custom” attribute of the “Node” XML element.

```
' VB
Private Sub AddFlow1_BeforeWriteXMLNode(ByVal sender As System.Object,
ByVal e As Lassalle.Flow.BeforeWriteXMLNodeEventArgs) Handles
AddFlow1.BeforeWriteXMLNode
        If (TypeOf (e.Node) Is MyNode) Then e.Custom = "MyNode"
End Sub

// C#
private void AddFlow1_BeforeWriteXMLNode(object sender,
                                          BeforeWriteXMLNodeEventArgs e)
{
      if (e.Node is MyNode)
            e.Custom = "MyNode";
}
```

How to include your custom data in the serialization process? You have just to use the **WriteMLNodeExtraData** event which is fired when serializing a node to allow the serialization of custom data. In our case, we use it to serialize the Author and Comment properties if the node is in fact a MyNode object.

```
' VB
Private Sub AddFlow1_WriteXMLNodeExtraData(ByVal sender As System.Object,
ByVal e As Lassalle.Flow.WriteXMLNodeExtraDataEventArgs) Handles
AddFlow1.WriteXMLNodeExtraData
   If (TypeOf (e.Node) Is MyNode) Then
     Dim mynode As MyNode = e.Node
     If (Not (mynode.Author = Nothing) And mynode.Author.Length > 0) Then
         e.Writer.WriteElementString("Author", mynode.Author)
     End If
     If (Not (mynode.Comment = Nothing) And mynode.Comment.Length > 0) Then
         e.Writer.WriteElementString("Comment", mynode.Comment)
     End If
   End If
End Sub

// C#
private void AddFlow1_WriteXMLNodeExtraData(object sender,
                                    WriteXMLNodeExtraDataEventArgs e)
{
    if (e.Node is MyNode)
    {
        MyNode mynode = (MyNode)e.Node;
        if (mynode.Author != null && mynode.Author.Length > 0)
            e.Writer.WriteElementString("Author", mynode.Author);
        if (mynode.Comment != null && mynode.Comment.Length > 0)
            e.Writer.WriteElementString("Comment", mynode.Comment);
    }
}
```

The following XML file is an example of  a file saved with the **DeriveNode** sample. Notice the “Custom” attribute and the “Author” and “Comment” new elements.

```
<?xml version="1.0"?>
<!--AddFlow.net diagram-->
<AddFlow Nodes="2" Links="0">
  <Version>2.1.0.0</Version>
  <DefaultNode>
    <FillColor>-32</FillColor>
    <Shape Style="Document" Orientation="so_0" />
  </DefaultNode>
  <Node Index="0" Left="49" Top="35" Width="94" Height="66"
Custom="MyNode">
    <Author>Alice</Author>
    <Comment>Interesting</Comment>
  </Node>
  <Node Index="1" Left="190" Top="93" Width="94" Height="72"
Custom="MyNode">
    <Author>Paul</Author>
    <Comment>Serious</Comment>
  </Node>
</AddFlow>
```

### *6.5.4.3.5 De-serialization of derived nodes*

When a node is created during the de-serialization of a XML stream, the **BeforeAddNode** event is not fired. Instead, the **BeforeReadXMLNode** event is fired and it is fired just before the node is de-serialized.

The event argument object contains the value of the “Custom” attribute of the node. Remember that in our case, this attribute contains the type of the node. This type allows knowing what contructor to use to instantiate the node. Here, there is only one possibility: the MyNode constructor.

The event argument object has also a Node parameter which contains the default values of the node properties. These default values have been previously read in the XML file.

The Node object corresponding to the MyNode object is returned to AddFlow since it is AddFlow that will perform the de-serialization of the node. (The custom properties of the MyNode object will be read with the **ReadXMLNodeExtraData** event)

```
' VB
Private Sub AddFlow1_BeforeReadXMLNode(ByVal sender As System.Object, ByVal
e As Lassalle.Flow.BeforeReadXMLNodeEventArgs)
                                    Handles AddFlow1.BeforeReadXMLNode
    If (e.Custom = "MyNode") Then
        Dim mynode As MyNode = New MyNode(e.DefNode, Nothing, Nothing)
        e.Node = mynode
    End If
End Sub

// C#
private void AddFlow1_BeforeReadXMLNode(object sender,
                                    BeforeReadXMLNodeEventArgs e)
{
    if (e.Custom == "MyNode")
    {
        MyNode mynode = new MyNode(e.DefNode, null, null);
        e.Node = mynode;
    }
}
```

The **ReadXMLNodeExtraData** event is fired when de-serializing a node and each time AddFlow encounters a custom XML element. In this example, there are two possible custom elements: "Author" and "Comment".

```
' VB
Private Sub AddFlow1_ReadXMLNodeExtraData(ByVal sender As System.Object,
ByVal e As Lassalle.Flow.ReadXMLNodeExtraDataEventArgs) Handles
AddFlow1.ReadXMLNodeExtraData
    If (TypeOf (e.Node) Is MyNode) Then
        Dim mynode As MyNode = e.Node
        If (e.Reader.Name = "Author") Then
            mynode.Author = e.Reader.ReadElementContentAsString()
        End If
        If (e.Reader.Name = "Comment") Then
            mynode.Comment = e.Reader.ReadElementContentAsString()
        End If
    End If
End Sub

// C#
private void AddFlow1_ReadXMLNodeExtraData(object sender,
                                          ReadXMLNodeExtraDataEventArgs e)
{
    if (e.Node is MyNode)
    {
        MyNode mynode = (MyNode)e.Node;
        if (e.Reader.Name == "Author")
            mynode.Author = e.Reader.ReadElementContentAsString();
        else if (e.Reader.Name == "Comment")
            mynode.Comment = e.Reader.ReadElementContentAsString();
    }
}
```

## 6.6 Conversion guide from the ActiveX Control

AddFlow for WinForms is NOT compatible with the ActiveX version. However, it provides approximately the same features as the ActiveX version (and also many new features!).

The following lists all properties, methods and events of the AddFlow ActiveX Control and its corresponding implementations in AddFlow for WinForms, grouped by classes (AddFlow control, Node class, Link class, etc).

**Note**: we provide a free tool **XToNetFlow** that allows converting a diagram created with the ActiveX version of AddFlow (version 5.4) to a diagram usable by the .NET version of AddFlow. You may find it at http://www.lassalle.com/download/XToNetFlow.zip. The source code is included.

### 6.6.1 AddFlow properties

| **ActiveX** | **.NET corresponding feature** |
|---|---|
| AdjustOrg | AddFlow1.DefLinkProp.AdjustOrg |
| AdjustDst | AddFlow1.DefLinkProp.AdjustDst |
| Alignment | AddFlow1.DefNodeProp.Alignment |
| AllowArrowKeys | None |
| ArrowDst | AddFlow1.DefLinkProp.ArrowDst.Style |
| ArrowMid | AddFlow1.DefLinkProp.ArrowMid.Style |
| ArrowOrg | AddFlow1.DefLinkProp.ArrowOrg.Style |
| Autorouting | See the paragraph Link Autorouting |
| AutoSize | AddFlow1.DefNodeProp.AutoSize |
| AutoScroll | AddFlow1.CanDragScroll |
| BackColor | AddFlow1.BackColor |
| BackMode | AddFlow1.DefNodeProp.BackMode<br>AddFlow1.DefLinkProp.BackMode |
| BackPicture | None (use instead the more powerful OwnerDraw property) |
| BorderStyle | None (workaround: place the AddFlow control in a Panel control) |
| CanChangeDst | AddFlow1.CanChangeDst |
| CanChangeOrg | AddFlow1.CanChangeOrg |
| CanDrawNode | AddFlow1.CanDrawNode |
| CanDrawLink | AddFlow1.CanDrawLink |
| CanFireError | AddFlow1.CanFireError |
| CanMoveNode | AddFlow1.CanMoveNode |
| CanMultiLink | AddFlow1.CanMultiLink |
| CanReflexLink | AddFlow1.CanReflexLink |
| CanSizeNode | AddFlow1.CanSizeNode |
| CanStretchLink | AddFlow1.CanStretchLink |
| CanUndoRedo | AddFlow1.CanUndoRedo |
| CustomShapeIndex | See the paragraph Custom Shapes |
| CustomShapes | See the paragraph Custom Shapes |
| DisplayHandles | AddFlow1.DisplayHandles |
| DrawColor | AddFlow1.DefNodeProp.DrawColor<br>AddFlow1.DefLinkProp.DrawColor |

| | |
|---|---|
| DrawStyle | AddFlow1.DefNodeProp.DashStyle<br>AddFlow1.DefLinkProp.DashStyle |
| DrawWidth | AddFlow1.DefNodeProp.DrawWidth<br>AddFlow1.DefLinkProp.DrawWidth |
| EditMode | AddFlow1.CanLabelEdit<br>AddFlow1.DefNodeProp.LabelEdit |
| Ellipsis | AddFlow1.DefNodeProp.Trimming |
| FillColor | AddFlow1.DefNodeProp.FillColor |
| Font | AddFlow1.DefNodeProp.Font<br>AddFlow1.DefLinkProp.Font |
| ForeColor | AddFlow1.DefNodeProp.TextColor<br>AddFlow1.DefLinkProp.TextColor |
| GridColor | AddFlow1.Grid.Color |
| GridStyle | AddFlow1.Grid.Style |
| Hidden | AddFlow1.DefNodeProp.Hidden<br>AddFlow1.DefLinkProp.Hidden |
| JumpSize | AddFlow1.JumpSize |
| LastUserAction | None |
| LinkCreationMode | AddFlow1.LinkCreationMode |
| LinkingHandleSize | AddFlow1.LinkHandleSize |
| LinkStyle | AddFlow1.DefLinkProp.Line.Style |
| LogicalOnly | None |
| MaxDegree | None |
| MaxInDegree | None |
| MaxOutDegree | None |
| MouseIcon | AddFlow1.Cursor |
| MousePointer | AddFlow1.Cursor |
| MultiSel | AddFlow1.MultiSel |
| Nodes | AddFlow1.Nodes |
| NoPrefix | None |
| OrientedText | AddFlow1.DefLinkProp.OrientedText |
| OrthogonalDynamic | AddFlow1.DefLinkProp.Line.OrthogonalDynamic |
| PicturePosition | AddFlow1.DefNodeProp.ImagePosition |
| Pictures | AddFlow1.Images |
| PointedArea | AddFlow1.PointedArea |
| PointedLink | AddFlow1.PointedItem |
| PointedNode | AddFlow1.PointedItem |
| ProportionalBars | None |
| ReadOnly | AddFlow1.Enabled |
| RedoCode | AddFlow1.RedoCode |
| RemovePointAngle | AddFlow1.RemovePointAngle |
| Repaint | Use the methods BeginUpdate and EndUpdate |
| Rigid | AddFlow1.DefLinkProp.Rigid |
| RoundedCorner | AddFlow1.DefLinkProp.Line.RoundedCorner |
| RoundedCornerSize | AddFlow1.RoundedCornerSize |
| RouteGrain | See the paragraph Link Autorouting |
| RouteMinDistance | See the paragraph Link Autorouting |
| RouteStartMethod | See the paragraph Link Autorouting |
| ScrollBars | None. Use AddFlow1.AutoScroll |
| ScrollTrack | None |

| | |
|---|---|
| ScrollWheel | None |
| SelectAction | AddFlow1.MouseAction |
| SelectedLink | AddFlow1.SelectedItem |
| SelectedNode | AddFlow1.SelectedItem |
| SelectionHandleSize | AddFlow1.SelectionHandleSize |
| SelectMode | None |
| SelLinks | AddFlow1.SelectedItems |
| SelNodes | AddFlow1.SelectedItems |
| Shadow | AddFlow1.DefNodeProp.Shadow.Style |
| ShadowColor | AddFlow1.DefNodeProp.Shadow.Color |
| Shape | AddFlow1.DefNodeProp.Shape.Style |
| ShapeOrientation | AddFlow1.DefNodeProp.Shape.Orientation |
| ShowGrid | AddFlow1.Grid.Draw |
| ShowJump | None. Use instead AddFlow1.DefLink.Jump |
| ShowPropertyPages | None |
| ShowToolTip | AddFlow1.ShowTooltips |
| SkipUndo | AddFlow1.SkipUndo |
| SnapToGrid | AddFlow1.Grid.Snap |
| SizeArrowDst | AddFlow1.DefLinkProp.ArrowDst.Size |
| SizeArrowMid | AddFlow1.DefLinkProp.ArrowMid.Size |
| SizeArrowOrg | AddFlow1.DefLinkProp.ArrowOrg.Size |
| StretchingPoint | AddFlow1.StretchingPoint |
| Tag | AddFlow1.Tag |
| Transparent | AddFlow1.DefNodeProp.Transparent |
| UndoCode | AddFlow1.UndoCode |
| UndoSize | AddFlow1.UndoSize |
| XExtent | AddFlow1.Extent.Width |
| XGrid | AddFlow1.Grid.Size.Width |
| XScroll | AddFlow1.ScrollPosition.X |
| XShadowOffset | AddFlow1.DefNodeProp.Shadow.Size.Width |
| XZoom | AddFlow1.Zoom.X |
| YExtent | AddFlow1.Extent.Height |
| YGrid | AddFlow1.Grid.Size.Height |
| YScroll | AddFlow1.ScrollPosition.Y |
| YShadowOffset | AddFlow1.DefNodeProp.Shadow.Size.Height |
| YZoom | AddFlow1.Zoom.Y |

### 6.6.2 AddFlow methods

| **ActiveX** | **.NET corresponding feature** |
|---|---|
| BeginAction | AddFlow1.BeginAction |
| CanPaste | None |
| CanRedo | AddFlow1.CanRedo |
| CanUndo | AddFlow1.CanUndo |
| Copy | None |
| DeleteSel | AddFlow1.DeleteSel |
| DeleteMarked | None |
| DisplayPropertyPage | None |
| EndAction | AddFlow1.EndAction |

| | |
|---|---|
| ExportPicture | AddFlow1.ExportMetafile |
| GetLinkAtPoint | GetItemAt |
| GetNodeAtPoint | GetItemAt |
| GetVersion | None |
| IsChanged | AddFlow1.IsChanged |
| IsSelChanged | AddFlow1.IsSelChanged |
| LoadFile | None (use XMLFlow) |
| LoadMemory | None (use XMLFlow) |
| Paste | None |
| Redo | AddFlow1.Redo |
| Refresh | AddFlow1.Refresh |
| SaveFile | None (use XMLFlow) |
| SaveImage | AddFlow1.ExportMetafile |
| SaveMemory | None (use XMLFlow) |
| SelectAll | `For Each item As Item In AddFlow1.Items`<br>`  item.Selected = True`<br>`Next` |
| SelectRectangle | AddFlow1.GetItemsInRectangle |
| SetChangedFlag | AddFlow1.SetChangedFlag |
| SetSelChangedFlag | AddFlow1.SetSelChangedFlag |
| StartEdit | node.BeginEdit and node.EndEdit |
| Undo | AddFlow1.Undo |
| ZoomRectangle | AddFlow1.ZoomRectangle |

### 6.6.3 AddFlow events

| **ActiveX** | **.NET corresponding feature** |
|---|---|
| AfterAddLink | AddFlow1.AfterAddLink |
| AfterAddNode | AddFlow1.AfterAddNode |
| AfterEdit | AddFlow1.AfterEdit |
| AfterMove | AddFlow1.AfterMove |
| AfterResize | AddFlow1.AfterResize |
| AfterSelect | AddFlow1.AfterSelect |
| AfterStretch | AddFlow1.AfterStretch |
| BeforeAddLink | AddFlow1.BeforeAddLink |
| BeforeAddNode | AddFlow1.BeforeAddNode |
| BeforeChangeDst | AddFlow1.BeforeChangeDst |
| BeforeChangeOrg | AddFlow1.BeforeChangeOrg |
| BeforeEdit | AddFlow1.BeforeEdit |
| Error | None |
| Scroll | AddFlow1.Scroll |

### 6.6.4 Node properties

| **ActiveX** | **.NET** |
|---|---|
| Alignment | node.Alignment |
| AutoSize | node.AutoSize |
| BackMode | node.BackMode |

| | |
|---|---|
| DrawColor | node.DrawColor |
| DrawStyle | node.DashStyle |
| DrawWidth | node.DrawWidth |
| EditMode | node.LabelEdit |
| FillColor | node.FillColor |
| Font | node.Font |
| ForeColor | node.TextColor |
| Height | node.Rect.Height or node.Size.Height |
| Hidden | node.Hidden |
| Index | node.Index |
| InLinks | node.InLinks |
| Key | None (use class derivation or the property bag) |
| Left | node.Rect.Left or node.Location.X |
| Links | node.Links |
| Logical | node.Logical |
| Marked | None (use class derivation or the property bag) |
| MaxDegree | None |
| MaxInDegree | None |
| MaxOutDegree | None |
| Moveable | None (use xMoveable and yMoveable) |
| OutLinks | node.OutLinks |
| Picture | None (use the ImageIndex property instead) |
| PictureIndex | node.ImageIndex |
| PicturePosition | node.ImagePosition |
| Selectable | node.Selectable |
| Selected | node.Selected |
| Shadow | node.Shadow.Style |
| Shape | node.Shape.Style |
| ShapeOrientation | node.Shape.Orientation |
| Sizable | None (use xSizeableand ySizeable) |
| Tag | None (use class derivation or the property bag) |
| TagVariant | None (use the new Tag object property instead) |
| Text | node.Text |
| Tooltip | node.Tooltip |
| Top | node.Rect.Top or node.Location.Y |
| Transparent | node.Transparent |
| UserData | None (use class derivation or the property bag) |
| Width | node.Rect.Width or node.Size.Width |
| xMoveable | node.XMoveable |
| xTextMargin | node.TextMargin.Width |
| xScrollable | None |
| xSizeable | node.XSizeable |
| yMoveable | node.YMoveable |
| yTextMargin | node.TextMargin.Height |
| yScrollable | None |
| ySizeable | node.YSizeable |
| ZOrder | node.ZOrder |
| ZOrderIndex | node.ZOrder |

### 6.6.5 Node methods

| **ActiveX** | **.NET corresponding feature** |
|---|---|
| Clone | node.Clone |
| EnsureVisible | node.BringIntoView |
| GetLinkedNode | node.GetLinkedNode |
| PropertyPage | None (Use the DlgFlow component instead) |

### 6.6.6 Link properties

| **ActiveX** | **.NET** |
|---|---|
| AdjustDst | link.AdjustOrg |
| AdjustOrg | link.AdjustDst |
| ArrowDst | link.ArrowDst.Style |
| ArrowMid | link.ArrowMid.Style |
| ArrowOrg | link.ArrowOrg.Style |
| BackMode | link.BackMode |
| DrawColor | link.DrawColor |
| DrawStyle | link.DashStyle |
| DrawWidth | link.DrawWidth |
| ExtraPoints | link.Points |
| Dst | link.Dst |
| Font | link.Font |
| ForeColor | link.TextColor |
| Hidden | link.Hidden |
| InIndex | None |
| Key | None (use class derivation or the property bag) |
| LinkStyle | link.Line.Style |
| Logical | link.Logical |
| Marked | None (use class derivation or the property bag) |
| Org | link.Org |
| OrientedText | link.OrientedText |
| OrthogonalDynamic | link.Line.OrthogonalDynamic |
| OutIndex | None |
| Rigid | link.Rigid |
| RoundedCorner | link.Line.RoundedCorner |
| Selectable | link.Selectable |
| Selected | link.Selected |
| ShowJump | link.Jump |
| SizeArrowDst | link.ArrowDst.Size |
| SizeArrowMid | link.ArrowMid.Size |
| SizeArrowOrg | link.ArrowOrg.Size |
| Stretchable | Link.Stretchable |
| Tag | None (use class derivation or the property bag) |
| TagVariant | None (use the new Tag object property instead) |
| Text | link.Text |
| TextSegment | None. (use the OwnerDraw property) |
| Tooltip | link.Tooltip |

| | |
|---|---|
| UserData | None (use class derivation or the property bag) |
| ZOrder | link.ZOrder |
| ZOrderIndex | link.ZOrder |

## 6.6.7 Link methods

| **ActiveX** | **.NET corresponding feature** |
|---|---|
| Clone | link.Clone |
| EnsureVisible | link.BringIntoView |
| PropertyPage | None (Use the DlgFlow component instead) |
| Reverse | link.Reverse |

# 7 Frequently Asked Questions

## 7.1 General Questions

1. **What is AddFlow for WinForms?**

AddFlow for WinForms is a .NET Windows Forms Custom control. It provides approximately the same features as the ActiveX version and also many new features, more flexibility and a simpler and more powerful object model. It has been completely rewritten in C# to take advantage of the infrastructure provided by .NET. It is 100% Managed Code.

2. **Is it compatible with the ActiveX version?**

AddFlow for WinForms is NOT compatible with the current ActiveX version. For instance, if you are porting an application from VB6, you will have to review almost all the code that uses AddFlow (however this is just a matter of a few hours).
If you need a strict compatibility, you can use the current ActiveX version (5.4) as a COM component in a .NET application. However we encourage you to use the .NET version which is far more powerful and easier to use.

3. **What has changed?**

AddFlow for WinForms is still easier to use since its programmatic interface has been changed towards more simplicity.
The "user data" properties (Marked, UserData, Tag, VariantTag, Key) are removed. We just keep a Tag property allowing attaching any object to a node or a link. However, if you still need such properties, you could add them by deriving the Node or the Link class. We provide the "DeriveNode" sample illustrating how to do. Another easier method is to use the PropertyBag.
Also the Picture property is removed. We just keep the PictureIndex property which is renamed ImageIndex.
The persistence methods (LoadFile, SaveFile, Copy, Paste, CanPaste) are removed. Instead, we encourage you to use either the IXMLSerializable interface, either the XMLFlow component.
Also some properties are grouped: for instance, the Shape and ShapeOrientation properties of ActiveX version are grouped in only one property, Shape, which is an object of type Shape which contains 2 properties: Style (Rectangle, Ellipse, Hexagon, ...) and Orientation (North, East, ...)
And finally, there are also some new properties (Items collection, SelItems collection, AntiAliasing property, etc) and also all the inherited properties, methods and events.
See the conversion guide in this present document.

4. **What are the concrete benefits of using the .Net version instead of the ActiveX version?**

The ActiveX version of AddFlow can be used as a COM component in a .NET application because COM objects are supported in .NET (however just as 16-bit programs are supported under 32-bit Windows!). We think anyway that, in the .NET environment, it is better to use a .NET component. Moreover AddFlow for WinForms offers some concrete benefits:

- The possibility to derive the Node and the Link classes to obtain new objects that fit your needs
- Anti-aliasing technology to obtain smoother diagrams
- Custom shapes for nodes and arrows
- SVG support

## 5. What is the license agreement for AddFlow, LayoutFlow?

The key points are the following:

- Each product is licensed **per individual developer**
- Each product is **runtime royalty free**
- The evaluation version of each product has a **nag screen**. You may use it for up to **60 days** for trials and design-time evaluation purposes only.

## 6. Can I use it in a web page?

AddFlow for WinForms is a windows form control, not an ASP control. It executes on the client, not on the server.
However, Internet Explorer supports the OBJECT tag for hosting ActiveX controls. In .NET, the OBJECT tag can be used to also host Windows Form controls and provide accesses to the properties of the hosted control.
For information about licensing, see the paragraph Licensing when using AddFlow in a web page

## 7. What platforms does AddFlow for WinForms support?

Our product works on those platforms with .NET support, namely, the following platforms,

Microsoft Windows® 98
Microsoft Windows NT® 4.0 (SP 6a required)
Microsoft Windows Millennium Edition (Windows Me)
Microsoft Windows 2000 (SP2 Recommended)
Microsoft Windows XP Professional
Microsoft Windows XP Home Edition
Microsoft Windows Vista
Microsoft Windows 7

## 8. What are the benefits of using AddFlow for WinForms

Small deployment assembly. The size of the Lassalle.Flow.dll file is just 332 Kb.
Small programming interface: we have always preferred the quality to the quantity. We refuse to provide an inflation of classes and properties.
Full integration with the .NET environment.
Great Flexibility.
Runtime royalty free.

## 7.2 Technical Questions

### 1. How to associate a context menu to a node?

Under AddFlow, a node is not implemented as a control and it has not any context menu associated. Therefore, I would use the context menu of AddFlow as in the following example:

```
// C#
private void AddFlow1_MouseDown(object sender, MouseEventArgs e)
{
  if (e.Button == MouseButtons.Right)
  {
    if (AddFlow1.ContextMenu != null)
      AddFlow1.ContextMenu.Dispose();
    Item item = AddFlow1.PointedItem;
    if (item is Node)
    {
      AddFlow1.ContextMenu = new ContextMenu();
      AddFlow1.ContextMenu.MenuItems.Add(new MenuItem("item1"));
      AddFlow1.ContextMenu.MenuItems.Add(new MenuItem("item2"));
      AddFlow1.ContextMenu.Show(this, new System.Drawing.Point(e.X, e.Y));
    }
  }
}

' VB
Private Sub AddFlow1_MouseDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles AddFlow1.MouseDown
    If (e.Button = MouseButtons.Right) Then
      If Not AddFlow1.ContextMenu Is Nothing Then
        AddFlow1.ContextMenu.Dispose()
      End If
        If TypeOf (AddFlow1.PointedItem) Is Lassalle.Flow.Node Then
          AddFlow1.ContextMenu = New ContextMenu()
          AddFlow1.ContextMenu.MenuItems.Add(New MenuItem("item1"))
          AddFlow1.ContextMenu.MenuItems.Add(New MenuItem("item2"))
          AddFlow1.ContextMenu.Show(Me, New System.Drawing.Point(e.X, e.Y))
        End If
    End If
End Sub
```

### 2. How to save a diagram in a file?

See the paragraph Serialization

### 3. Is it possible to display an image in a node?

Yes. See the paragraph Displaying an image in a node

### 4. Is it possible to use a custom shape for a node?

Yes. See the paragraph Custom Shapes

## 5. How can we tell when nodes or links receive Mouse events?

By design, we have decided to not associate any click events to nodes and links. It means that you will have to use the events of the control itself and in the handler of these events, use the AddFlow properties to know if you have clicked on a node or a link. There is an example in the afEdit sample provided with AddFlow:

```
' VB
Private Sub AddFlow1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles AddFlow1.MouseUp
    Dim item As Item = AddFlow1.SelectedItem
    If Not (item Is Nothing) Then
        If TypeOf (item) Is Node Then
            Dim node As Node = CType(item, Node)
            PropertyGrid1.SelectedObject = node
            Label1.Text = "Selected Node"
        ElseIf TypeOf (item) Is Link Then
            Dim link As Link = CType(item, Link)
            PropertyGrid1.SelectedObject = link
            Label1.Text = "Selected Link"
        End If
    Else
        PropertyGrid1.SelectedObject = AddFlow1
        Label1.Text = "AddFlow control"
    End If
End Sub

// C#
private void AddFlow1_MouseUp(object sender,
                     System.Windows.Forms.MouseEventArgs e)
{
      Item item = AddFlow1.SelectedItem;
      if (item != null)
      {
            if (item is Node)
            {
                  Node node = (Node)item;
                  PropertyGrid1.SelectedObject = node;
                  Label1.Text = "Selected Node";
            }
            else if (item is Link)
            {
                  Link link = (Link)item;
                  PropertyGrid1.SelectedObject = link;
                  Label1.Text = "Selected Link";
            }
      }
      else
      {
            PropertyGrid1.SelectedObject = AddFlow1;
            Label1.Text = "AddFlow control";
      }
}
```

## 6. How to make the node's border transparent?

You have to use a transparent color:

```
node.DrawColor = Color.Transparent
```

7. **How to autofit the diagram; i.e. how to adjust the zoom to its maximum while still keeping all the shapes (nodes, links etc.) in view?**

The **ZoomRectangle** method and the **Extent** property should allow implementing this feature:

```
Dim rc As RectangleF = New RectangleF(New PointF(0, 0), AddFlow1.Extent)
AddFlow1.ZoomRectangle(rc, ZoomType.Isotropic)
```

8. **What is the purpose of HFlow, OFlow, SFlow, SPFlow, TFlow?**

See the paragraph Automatic Graph Layout

9. **How to execute a layout asynchronously?**

This is an important question, in particular when using SFlow which is not very quick in comparison with OFlow, SPFlow, TFlow or HFlow.
Under .NET, it is the consumer (of a component or a service) that specifies whether a method is to be run asynchronously or not. The creator of the component does not have to provide an explicit mechanism to allow the consumer to execute a method asynchronously.
How to call the layout method of SFlow asynchronously? You can use a background thread. You will have to create the thread, assign it a starting method and start the thread to begin its work.

Declare a thread in your class:

```
Thread thread = null;
```

Suppose you launch the layout from the menu. Following is the code you could use for this purpose:

```
void MenuLayoutOnClick(object obj, EventArgs ea)
{
  if (thread != null && thread.IsAlive)
    return;

  // Create the thread and pass it the SFlowLayout method
  ThreadStart ts = new ThreadStart(SFlowLayout);
  thread = new Thread(ts);
  thread.IsBackground = true;

  // Start the thread of execution
  thread.Start();
}

void SFlowLayout()
{
  // Create the SFlow component and perform the force directed Layout
  SFlow sflow = new SFlow();
  sflow.VertexDistance = 50;
  sflow.Layout(addflow);
}
```

10. **How to manage so that the layout algorithm applies only to a subset of the graph?**

Only the nodes and links whose **Logical** property is true are taken into account by the Layout method of HFlow, OFlow, SFlow, SPFlow and TFlow.

By default, the Logical property of an item (node or link) is true. You have just to set it to false to cause the Layout algorithm to ignore the item.

This will allow you to apply the layout only to important nodes. For instance, you can exclude the nodes that are just used to display a label by setting their Logical property to false. The layout will position only the important (logical) nodes. The secondary (not logical) nodes will not be moved by the layout.
Therefore, after the execution of the layout, you will have to programmatically move the secondary nodes so that they retrieve their correct position relatively to the main nodes.