AddFlow for Silverlight V 2.0 Tutorial

January 2014

Lassalle Technologies

http://www.lassalle.com

CONTENTS

1) Introduction

AddFlow for Silverlight is a general purpose Flowcharting/Diagramming Silverlight component, which lets you quickly build flowchart-enabled Silverlight applications.

AddFlow for Silverlight 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, …

AddFlow is runtime royalty free.

It has been created with **VS 2012** and **Silverlight 5**.

Purpose of this tutorial

This tutorial provides information on:

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

Who should use this tutorial?

This guide is intended for application programmers building Silverlight applications.

Samples

AddFlow for Silverlight is installed with one demo sample written in C#: DemoFlow. The source code of DemoFlow is provided.

2) Last Version enhancements

2.1 Version 2.0

2.1.1 A major change

In previous versions, AddFlow was a control consisting in a Border element containing a Scrollviewer element containing a Canvas panel.

Now, it is just a Canvas panel. And you have the possibility to place it yourself in a Scrollviewer element.

2.1.1.1 C # changes

Programmatically, the consequences are minor in your C# code. In fact it just means that the Diagram property is removed. Instead of writing for instance:

```
IEnumerable<Node> nodes = addflow.Diagram.Children.OfType<Node>();
```

you will write:

```
IEnumerable<Node> nodes = addflow.Children.OfType<Node>();
```

Therefore you replace addflow.Diagram.Children by addflow.Children that is all.

2.1.1.2 XAML changes

In the xaml definition of AddFlow, you will have to replace:

```
                <af:AddFlow x:Name="addflow"
                            Background="White" />
```

by:

```
<Border x:Name="Border"
                BorderBrush="Red"
                BorderThickness="2"
                CornerRadius="2" >
            <ScrollViewer x:Name="ScrollViewer"
                          HorizontalScrollBarVisibility="Auto"
                          VerticalScrollBarVisibility="Auto"
                          Padding="1"
                          Background="White"
                          BorderBrush="Transparent"
                          BorderThickness="0"
                          Margin="1"
                          IsTabStop="False"
                          TabNavigation="Once">
                <af:AddFlow x:Name="addflow"
                            Background="White" />
            </ScrollViewer>
</Border>
```

2.1.2 New features

2.1.2.1 Virtualization

The new property **IsVirtualizing** determines whether virtualizing mode is allowed. This is a UI virtualization, which collapses all items outside the viewable area, thus enhancing the performance while manipulating the items. This feature is demonstrated in the “Stress” example in the demo.

2.1.2.2 Bird view

The new **BirdView** property returns an object managing a bird view popup window allowing scrolling and zooming the diagram (see paragraph #5.9.3.Bird view|outline)

2.1.2.3 IsFixedSize

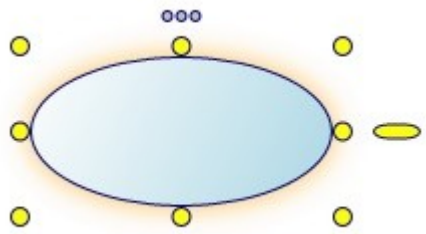
The new property **IsfixedSize** determines whether the size of the addflow canvas is fixed or if it depends on the size of the diagram. It is false by default.

2.1.2.4 Grid

Now, you may show or hide a grid (**GridDraw**, **GridColor**, GridSnap, GridSize properties)

2.1.2.5 IsContextHandle

The Node and Link **IsContextHandle** property determines if a context handle ooo is displayed when the node or the link is selected.



When the context handle is clicked, a **ContextHandleClicked** event is fired, allowing you for instance to display a context menu or a dialog box.

By default, the IsContextHandle property is false. Also, if the addflow **CanShowContextHandle** property is false, context handles will not be displayed, whatever the value of the isContexthandle property of each item may be.

There is an example of the use of this event in the "State diagram" example and in the "Social network" example of the demo. In the second example, only nodes have a context handle.

2.2 Version 1.5

The demo shows how to make blincking items.

The algorithm used for the license checking is now FIPS compliant.

2.3 Version 1.4

2.3.1 Version 1.4.2

This new release offers new properties (**TextBlockBorder**, **TextAngleMode**, **TextXOffset, TextYOffset**) allowing greater flexibility in the way link labels are displayed. See the paragraph 5.5.

2.3.2 Version 1.4.1

The version 1.4.1 adds the possibility to zoom the diagram interactively with the mouse if the **MouseSelection** property of the AddFlow control is set to **MouseSelection.Zoom**.

It adds also two properties **SelectionBoxStyle** and **ZoomingBoxStyle** allowing defining the style of the rectangle used when selection items or zooming interactively with the mouse (demonstrated in the “Stress” example in the demo)

2.3.3 Version 1.4.0

This new release provides a **rotation feature**. Nodes can be rotated.

If the **CanRotateNode** property of AddFlow is true (which is the case by default) and if the **IsRotatable** property of a node is true (which is the case by default), a node can be rotated programmatically, using the node **RotationAngle** property or interactively, using a handle placed at the right of the node.

You may change the look of this rotation handle using the **RotateHandleStyle** property.

When rotating nodes interactively, the **SelectedNodesRotating** event is fired and the IsRotatingNode property of AddFlow is true.

2.4 Version 1.3

This new release has been created with VS 2010 and Silverlight 4.

2.5 Version 1.2

This new release provides the following new features:

- **Panning mode**

  If you set the MouseSelection property to **MouseSelection.Pan**, you enter in panning mode: you can scroll the diagram directly with the mouse (don't need to use the scrollbars)

- **New properties**

  PinDst and **PinOrg** properties allowing to get the origin and destination pin of a link.

- **New events**

  SelectedNodesLayouting and **SelectedLinkStretching** events. The first event is used in the file **PageRigidLinks.xaml.cs**

2.6 Version 1.1

This new release provides the following new features:

- **Link intersections**

  AddFlow uses a powerful algorithm to find intersections between the link segments. If the **CanShowJumps** property of the AddFlow control is true then jumps will be displayed at the intersection of this link with other links.

- **In-place edition for nodes**

  If the **CanEditMode** property of the AddFlow control is true and if the **IsEditable** property of the node is also true (which is the case by default), then you can interactively edit the text inside the node. If it is the case, then the **IsInEditMode** property of the node is true.

- **Dash line for links**

  The **StrokeDashArray** property allows drawing a link with a dashes line style.

3) Getting Started

3.1 Installation

The AddFlow for Silverlight 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 Silverlight installation folder, it creates 3 subdirectories: Bin, Doc and Demo:

- The **Bin** subdirectory contains the AddFlow and LayoutFlow assemblies.
- The **Doc** subdirectory contains the help file, the tutorial, the readme file and the license agreement.
- The **Demo** subdirectory contains the C# source code of the DemoFlow sample that demonstrates AddFlow for Silverlight. It contains also the source code of:

  - **Lassalle.AlgoFlow**, a dll providing some graph algorithms.

  - **Lassalle.Silverlight.Arrows**, a dll providing some predefined arrow geometries.

TIP: The source code of each of these dll (AlgoFlow, Arrow) is installed with AddFlow.

3.2 AddFlow Assemblies

Following is the list of the AddFlow for Silverlight extensions, including AddFlow itself. All these assemblies are installed with AddFlow for Silverlight.

Assembly	**Description**
Lassalle.Silverlight.Flow.dll	The AddFlow for Silverlight control
Lassalle.Silverlight.Flow.Layout.dll	The LayoutFlow dll is a set of Graph layout algorithms. It is an extension of AddFlow. It can be purchased separately.

TIP: Is the source code of AddFlow or LayoutFlow available?

The source code, written in C# is not provided. However you can purchase a license of the source code.

3.3 Licensing

3.3.1 Type of licenses

Notice that you can purchase either:

- an AddFlow for Silverlight **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 Silverlight **Professional** license. This license includes LayoutFlow. You can execute the graph layout algorithms provided by LayoutFlow without any restriction.

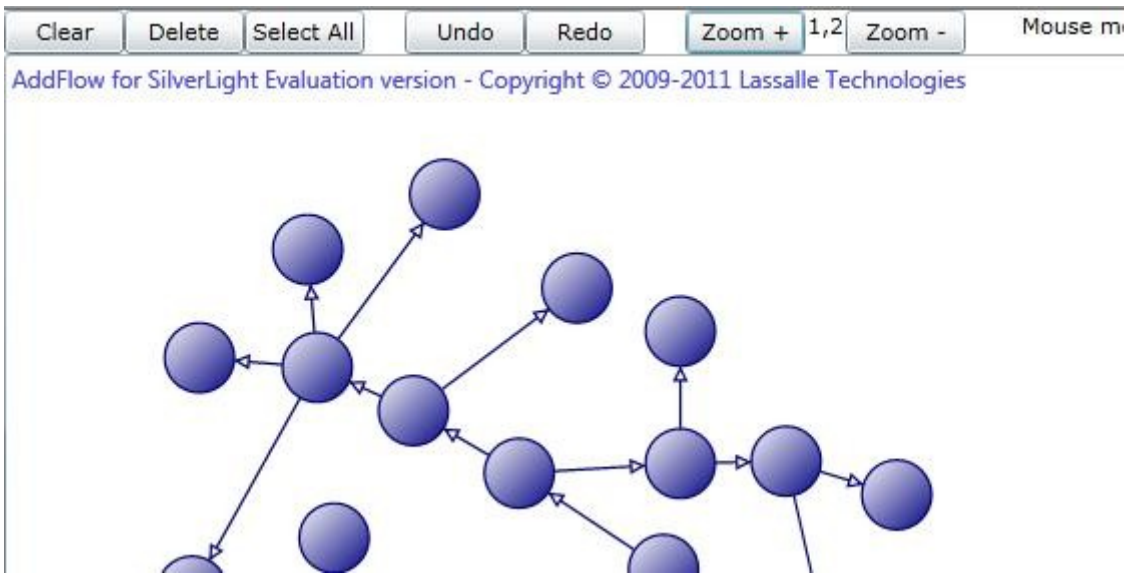
3.3.2 How it works?

The evaluation version

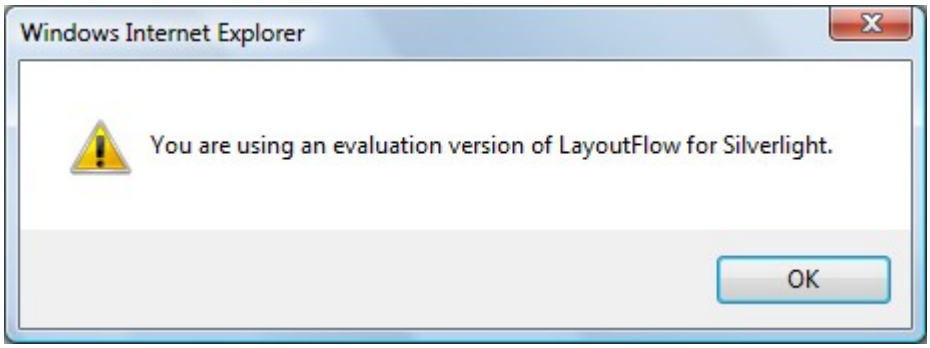
When you install AddFlow for Silverlight, you install in fact an evaluation version of AddFlow. (And you install also an evaluation version of LayoutFlow)

If you generate ("compile") an application that uses this evaluation version of AddFlow for Silverlight, 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.

In the following example, you can see the evaluation label displayed at the top of the diagram.



And if you execute one of the graph layout methods provided by the LayoutFlow dll, 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).

Now, it depends of the type of license you have purchased.

- If you have just purchased a Standard license of AddFlow for Silverlight, the license key will alllow you removing the evaluation label. However you will continue facing the nag screen if you try to execute a graph layout method.

- If you have just purchased a Professional license of AddFlow for Silverlight, no nag screen will be displayed if you execute a graph layout method.

WARNING: How to use the license key ?

You will receive the instructions about how to use the license key when you will order the product. If you do not receive them, do not hesitate to contact us to obtain them.

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
- In place edition for nodes

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.

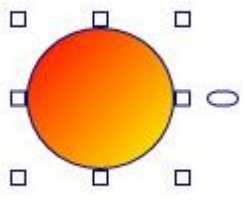
And a set of methods and properties allow implementing a powerful Undo/Redo feature.

4.2 Create a diagram interactively

4.2.1 Draw a node

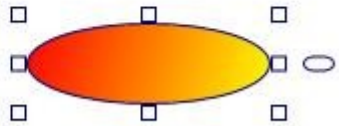
Bring the mouse cursor into the control, press the left button, move the mouse and release the left button. You have created an elliptic node. This node is selected: that's why 8 handles (little squares) are displayed.

The 8 handles allow **resizing the node** (the handle at the right allows rotating it). If you want to **move the node,** you bring the mouse cursor into the node (but not in the center), press the left button, move the mouse and release the left button.

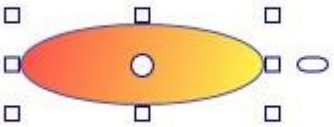


4.2.2 Draw a link

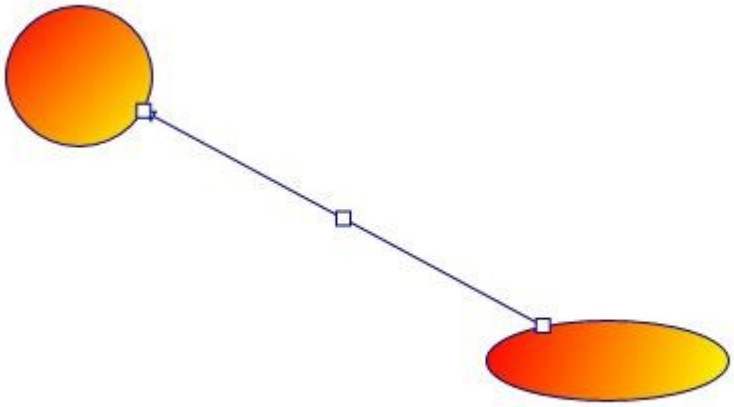
Draw a second node.



Then bring the mouse cursor above the second node. A small circle handle is then displayed at the center of the selected node.



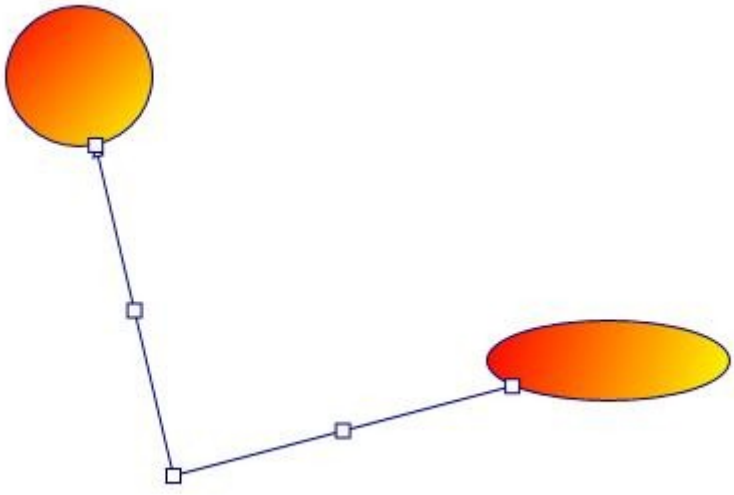
Bring the mouse over this small circle handle, press the left button, move the mouse towards the other node. When the mouse cursor is into the other node, release the left button. The link has been created. And it is selected: 3 handles are displayed in the link.



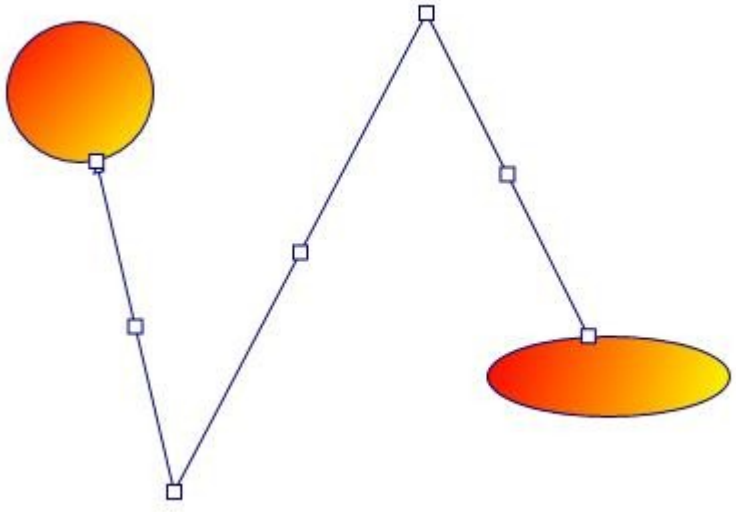
As you can see, the link stretching handles are also displayed as little rectangles. By default, those handles are small rectangles as for the nodes above. But we can change the style of those handles. The DemoFlow sample provided with AddFlow shows many distinct ways to display the node resizing handles and the link stretching handles.

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).

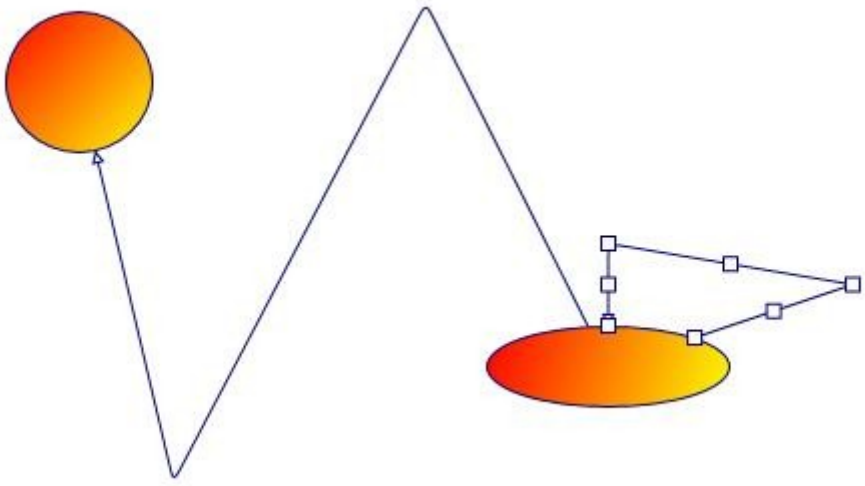


Create another segment



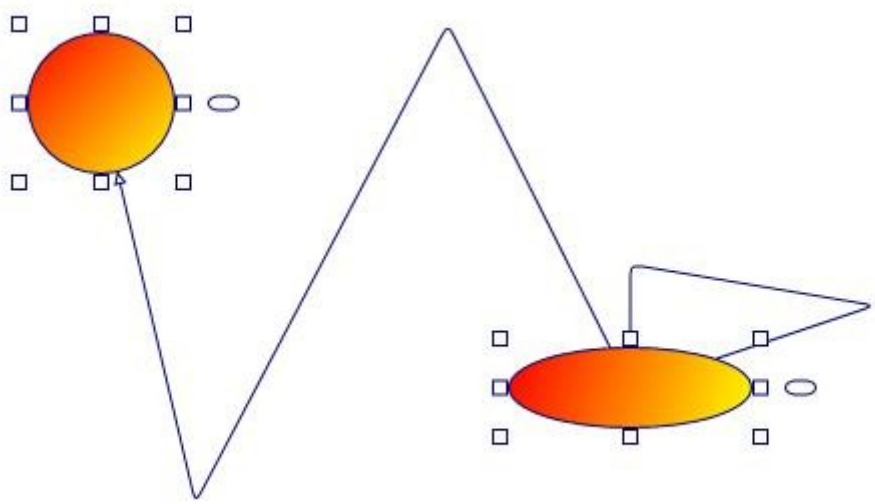
4.2.4 Draw a reflexive link

Select a node by clicking on it. Then bring the mouse cursor above the small diamond handle at the center of 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.

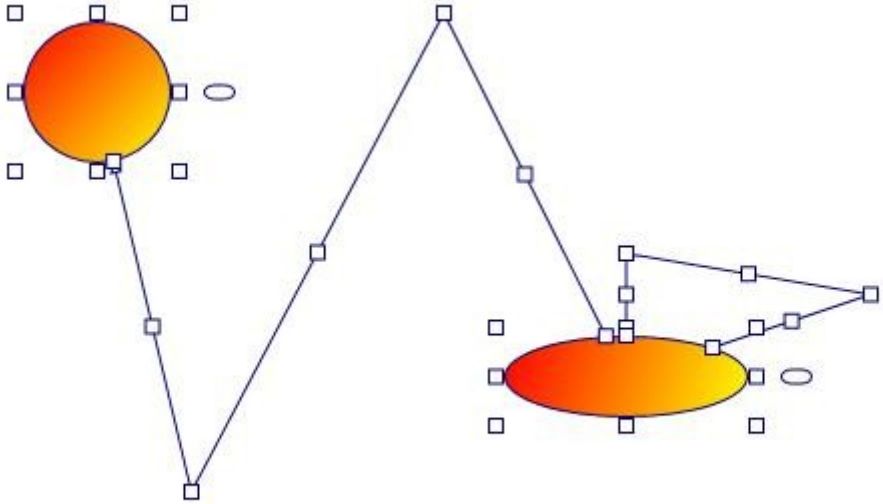


4.2.5 Multiselection

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



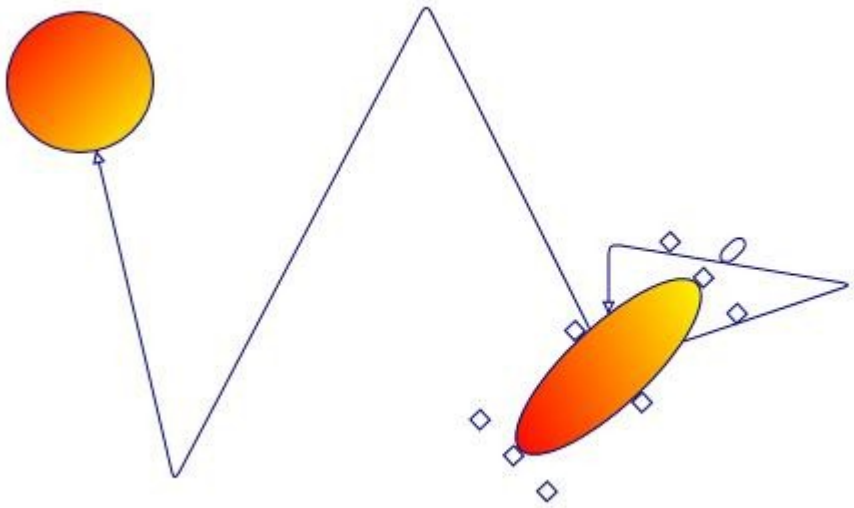
You can also select links or nodes and links.



There is another way to perform multiselection, using the **MouseSelection** property and assigning it the **MouseSelection.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 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.

4.2.6 Node rotation

Bring the mouse cursor into the handle iplaced at the right of a node, press the left button, move the mouse and release the left button. You have rotated the node.



4.2.7 Change properties of a node or a link

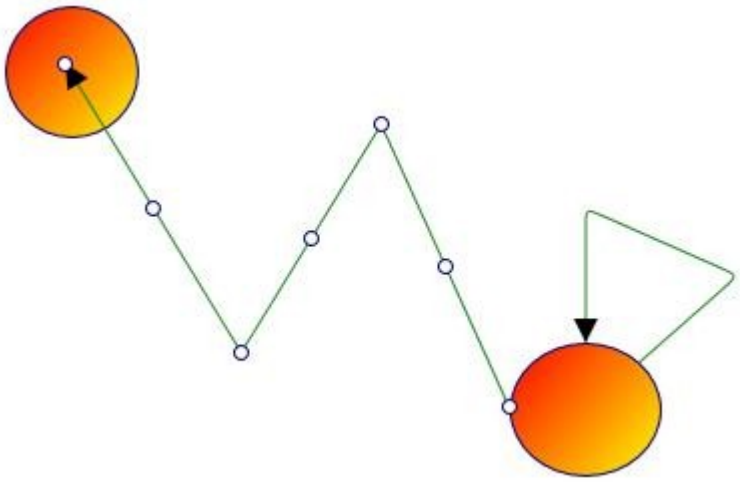
Interactively, without adding any code, you can change the position and the size of a node. You can add segments to a link or remove them. To change other properties (shape, styles, colors, behaviors, etc) of a node or a link, you have to write some code.

4.2.8 Adjust the link origin and destination points

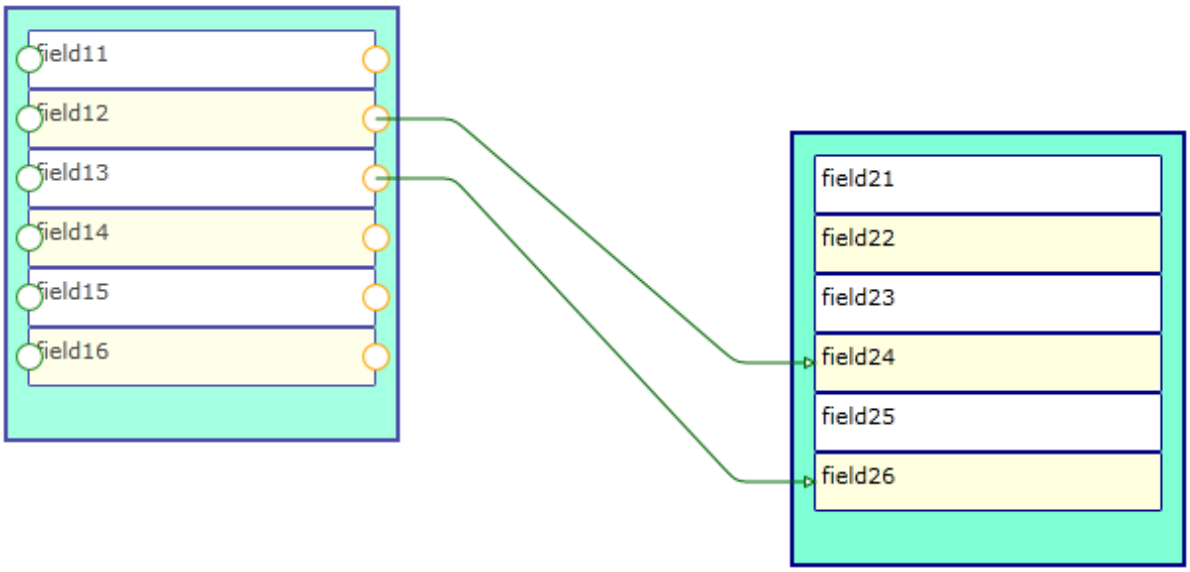
By default, you cannot adjust the extremities of the links. For instance, if you select a link and bring the mouse cursor into the last (or the first) handle of this link, press the left button, move the mouse in another place then relinquish the mouse button, the link springs back again, retrieving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

However, you can change this behavior by using the **InConnectionMode** property of the destination node of the link and the **OutConnectionMode** property of the origin node of the link.

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.

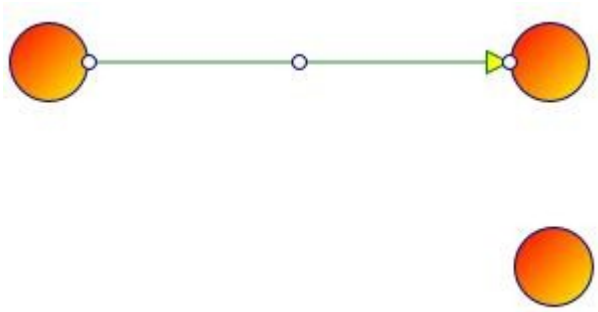


But, as you will see later in this tutorial, you can also use another way to create links by using pins.



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.You bring the mouse cursor into the



third link handle (near the arrow head), press the left button, move the mouse until the isolated node and release the left button.

The new destination of the link has changed.

5) Programmatic creation of a diagram

5.1 Overview

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

The main class is the **AddFlow** class that derives from the Canvas class.

An AddFlow diagram contains two kinds of objects, **Node** objects and **Link** objects. A Link object allows linking 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.

The Node class derives from the ContentControl class whereas the Link class derives from the Control class.

You can obtain the collection of the objects contained in the canavs by using the **Children** property that includes all the nodes, all the links and also all other UIElement objects that could be included in the panel.

This Children property is important since it is the only way to obtain the list nodes and links of AddFlow. The good news is that you can use the new **LINQ** technology to select the set of objects you wish. For instance,

if you wish to select all nodes in an IEnumerable object:

```
IEnumerable<Node> nodes = addflow.Children.OfType<Node>();
```

then, if for instance, you wish to draw them with a red color:

```
foreach (Node node in nodes)
    this.EmphasizeNode(node, new SolidColorBrush(Colors.Red));
```

or if you prefer selecting all the nodes in an array:

```
var nodes = addflow.Children.OfType<Node>().ToArray();
foreach (Node node in nodes)
    this.EmphasizeNode(node, new SolidColorBrush(Colors.Red));
```

Same thing for the links:

```
IEnumerable<Link> links = addflow.Children.OfType<Link>();
foreach (Link link in links)
    this.EmphasizeLink(link, new SolidColorBrush(Colors.Red));
```

If you wish selecting all the nodes that are selectable:

```
var nodes = addflow.Children.OfType<Node>().Where(
                                            node => node.IsSelectable == true);
foreach (Node node in nodes)
    this.EmphasizeNode(node, new SolidColorBrush(Colors.Red));
```

or if you wish selecting all the links that are stretchable:

```
var links = addflow.Children.OfType<Link>().Where(
                                            link => link.IsStretchable == true);
foreach (Link link in links)
    this.EmphasizeLink(link, new SolidColorBrush(Colors.Red));
```

5.2 Diagram creation

5.2.1 Our first program

Using Visual Studio .NET, select the menu item **File | New | Project...**. In the Add New Project dialog box, select the **Visual C#** project type, then choose **Silverlight Application,** change the project name to **AddFlowSilverlightTutorial1** and click **ok**.

In the "**Add Silverlight Application"** dialog box, uncheck the checkbox.

Right-click in the **References** item underneath the project name and select **Add Reference** from the context menu. Then, use the **Browse...** button to search for the Lassalle.Silverlight.Flow.dll and select it.

The initial XAML description of the main application window **MainPage.xaml** is the following:

```
<UserControl x:Class="SilverlightApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">

    </Grid>
</UserControl>
```

Let us replace it by the following XAML code:

```
<UserControl x:Class="SilverlightApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 xmlns:af="clr-namespace:Lassalle.Silverlight.Flow;assembly=Lassalle.Silverlight.Flow"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
        <af:AddFlow x:Name="addflow" Background="White" />
    </Grid>
</UserControl>
```

If you compile and run this application, you will see that it allows to create interactively nodes and links. Nodes can be moved and resized. Links can be stretched.

However you cannot scroll the diagram. So let us add a ScrollViewer containing AddFlow and also a red Border containg the ScrollViewer.

```
<UserControl x:Class="SilverlightApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:af="clr-
namespace:Lassalle.Silverlight.Flow;assembly=Lassalle.Silverlight.Flow"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
```

```
    <Grid x:Name="LayoutRoot" Background="White">
        <Border x:Name="Border"
                BorderBrush="Red"
                BorderThickness="2"
                CornerRadius="2" >
            <ScrollViewer x:Name="ScrollViewer"
                          HorizontalScrollBarVisibility="Auto"
                          VerticalScrollBarVisibility="Auto"
                          Padding="1"
                          Background="White"
                          BorderBrush="Transparent"
                          BorderThickness="0"
                          Margin="1"
                          IsTabStop="False"
                          TabNavigation="Once">
                <af:AddFlow x:Name="addflow"
                            Background="White" />
            </ScrollViewer>
        </Border>
    </Grid>
</UserControl>
```

Now, as soon as you create some nodes, scrollbars are appearing.

Now, we are going to create a diagram with C# code. Let us open the code-behind file **MainPage.xaml.cs**.

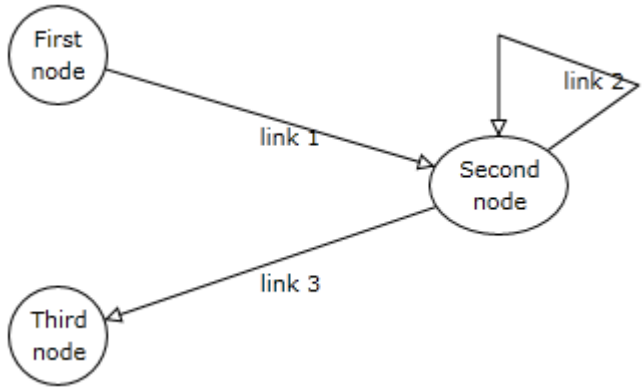
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

Let us replace it by the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Lassalle.Silverlight.Flow;
```

```
namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
            this.CreateDiagram1();
        }

        private void CreateDiagram1()
        {
            // Create 3 nodes
            Node node1 = this.addflow.AddNode(20, 20, 50, 50, "First node");
            Node node2 = this.addflow.AddNode(135, 85, 62, 50, "Second node");
            Node node3 = this.addflow.AddNode(20, 115, 50, 50, "Third node");

            // Create 3 links
            Link link1 = this.addflow.AddLink(node1, node2, "link 1");
            Link link2 = this.addflow.AddLink(node2, node2, "link 2");
            Link link3 = this.addflow.AddLink(node2, node3, "link 3");
        }
    }
}
```

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 color is black. The text color is black.

We are going to enhance this diagram.

However, let us focus on the way nodes and links are created. First we create the nodes then we create the links. This is because a link cannot exist without its origin and destination nodes.

To create a node, you have to use the **AddNode** method. There is no other method. However, there are 4 versions of this method. There is a version of AddNode that takes only the first 4 parameters. For instance, to create the first node, you could have written:

```
Node node1 = this.addflow.AddNode(20, 20, 50, 50);
node1.Text = "First node";
```

There is also a version of AddNode that takes just one parameter of Node type.

```
Node node1 = new Node(20, 20, 50, 50, "First node");
this.addflow.AddNode(node1);
```

To create a link, you have to use the **AddLink** method. There is no other method. However there is 3 versions of AddLink. There is a version of AddLink that takes only the 2 first parameters. For instance, to create the first link, you could have written:

```
Link link1 = this.addflow.AddLink(node1, node2);
link1.Text = "link 1";
```

There is also a version of AddLink that takes just one parameter of Link type.

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

Notice that we are not forced to memorize each node reference. Using LINQ, we could rewrite CreateDiagram1 as follows:

```
 private void CreateDiagram1()
 {
      // Create 3 nodes
      this.addflow.AddNode(20, 20, 50, 50, "First node");
      this.addflow.AddNode(230, 85, 70, 50, "Second node");
      this.addflow.AddNode(20, 160, 50, 50, "Third node");

      // We use LINQ to select in an array all the nodes.
      var nodes = this.addflow.Children.OfType<Node>().ToArray();

      // Create 3 links
      this.addflow.AddLink(nodes[0], nodes[1], "link 1");
      this.addflow.AddLink(nodes[1], nodes[1], "link 2");
      this.addflow.AddLink(nodes[1], nodes[2], "link 3");
 }
```

Notice that you could also write it using the “one parameter versions of AddNode and Addlink”:

```
private void CreateDiagram11()
{
      // Create 3 nodes
      Node node1 = new Node(20, 20, 50, 50, "First node");
      Node node2 = new Node(230, 85, 70, 50, "Second node");
      Node node3 = new Node(20, 160, 50, 50, "Third node");
      this.addflow.AddNode(node1);
      this.addflow.AddNode(node2);
      this.addflow.AddNode(node3);

      // Create 3 links
      Link link1 = new Link(node1, node2, "link 1");
      Link link2 = new Link(node2, node2, "link 2");
      Link link3 = new Link(node2, node3, "link 3");
      this.addflow.AddLink(link1);
      this.addflow.AddLink(link2);
      this.addflow.AddLink(link3);
 }
```

or, using a more compact style:

```
  private void CreateDiagram12()
  {
      // Create 3 nodes
      this.addflow.AddNode(new Node(20, 85, 50, 50, "First node"));
      this.addflow.AddNode(new Node(230, 85, 70, 50, "Second node"));
      this.addflow.AddNode(new Node(20, 160, 50, 50, "Third node"));

      // We use LINQ to select in an array all the nodes of the canvas.
      var nodes = this.addflow.Children.OfType<Node>().ToArray();

      // Create 3 links
      this.addflow.AddLink(new Link(nodes[0], nodes[1], "link 1"));
```

```
        this.addflow.AddLink(new Link(nodes[1], nodes[1], "link 2"));
        this.addflow.AddLink(new Link(nodes[1], nodes[2], "link 3"));
    }
```

In the DemoFlow sample provided with AddFlow, we generally use this method.

5.2.2 Node Properties

The node properties are described in the **Lassalle.Silverlight.Flow** namespace in the AddFlow help file. Many of them are inherited from the ContentControl class.

We can group them in 5 categories:

- Layout properties: **Location**, **Size, RotationAngle**
- Appearance properties: **Content, ImageUri, ImageHorizontalAlignment, ImageVerticalAlignment**
- Node label properties: **Text, TextHorizontalAlignment, TextVerticalAlignment, TextBlock**
- Graph properties: the **Links** collection property that allow getting all the links (in and out) of the node.
- Behaviour properties: **IsSelected, IsSelectable, IsXMoveable, IsYMoveable, IsXSizeable, IsYSizeable, IsLinkingOver, IsEditable, IsRotatable, IsContext.**
- Connection properties defining how a link is attached to a node: **InConnectionMode**, **OutConnectionMode** and **Connector**

5.2.3 Link Properties

The link properties are described in the **Lassalle.Silverlight.Flow** namespace in the AddFlow help file. Many of them are inherited from the Control class.

We can group them in 4 categories:

- Appearance properties: **PathGeometry**, **HitPathGeometry**, **LineStyle RoundedCornerSize, Stroke, StrokeThickness, StrokeDashArray, JumpSize.**
- Link label properties: **Text, TextPlacementMode, TextAngleMode, TextXPosition, TextYPosition, TextXOffset, TextYOffset, TextBlockBorder, TextBlock**
- Arrow head properties: **ArrowDstGeometry, ArrowOrgGeometry, ArrowDstFill, ArrowOrgFill, ArrowDstX**, **ArrowDstY**, **ArrowOrgX**, **ArrowOrgY**, **ArrowDstWidth**, **ArrowDstHeight**, **ArrowOrgWidth**, **ArrowOrgHeight**, **ArrowOrgAngle**, **ArrowDstAngle**
- 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: **IsSelected, IsSelectable, IsStretchable, IsContext**

WARNING: There is also the **Points** collection. A link is composed of several segments defined by a collection of points. However, you should not use this collection directly except for serialization purposes. To manipulate the collection the link points, you should use instead the methods **AddPoint, RemovePoint, ClearPoints, SetPoint, GetPoint** and **CountPoints.**

5.2.4 Changing property values

Now let us replace the CreateDiagram1 method by the following CreateDiagram2 method:

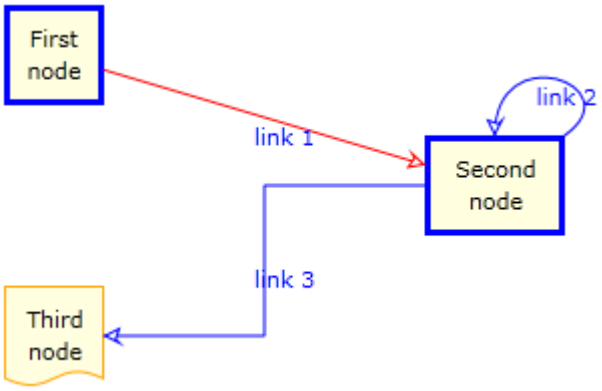
```
        private void CreateDiagram2()
        {
            string xamlRectangle =
                        "<Rectangle
xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Stroke='Blue'
StrokeThickness='3' Fill='LightYellow' />";
            string xamlDocument =
                        "<Path
xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Stroke='Orange'
Fill='LightYellow' Stretch='Fill' Data='M 0,0 H 60 V 40 C 30,30 30,50 0,40 Z' />";

            // Create 3 yellow nodes.
            // The first and second nodes are rectangular
            // and the third one has a Document shape style.
            this.addflow.AddNode(20, 20, 50, 50, "First node",
            XamlReader.Load(xamlRectangle));
            this.addflow.AddNode(230, 85, 70, 50, "Second node",
            XamlReader.Load(xamlRectangle));
            this.addflow.AddNode(20, 160, 50, 50, "Third node",
            XamlReader.Load(xamlDocument));

            // We use LINQ to select in an array all the nodes of the canvas.
            var nodes = this.addflow.Children.OfType<Node>().ToArray();

            // Create 3 links with distinct colors and arrows
            // The first link has a green arrow.
            // The second link has a Bezier style
            // The third link has a "HVH" style.
            Link link = this.addflow.AddLink(nodes[0], nodes[1], "link 1");
            link.Foreground = new SolidColorBrush(Colors.Blue);
            link.Stroke = new SolidColorBrush(Colors.Red);
            link.ArrowDstGeometry = MakeMyArrowGeometry();

            link = this.addflow.AddLink(nodes[1], nodes[1], "link 2");
            link.Foreground = new SolidColorBrush(Colors.Blue);
            link.Stroke = new SolidColorBrush(Colors.Blue);
            link.LineStyle = LineStyle.Bezier;
            link.ArrowDstGeometry = MakeMyArrowGeometry();

            link = this.addflow.AddLink(nodes[1], nodes[2], "link 3");
            link.Foreground = new SolidColorBrush(Colors.Blue);
            link.Stroke = new SolidColorBrush(Colors.Blue);
            link.LineStyle = LineStyle.HVH;
            link.ArrowDstGeometry = MakeMyArrowGeometry();
        }

        static Geometry MakeMyArrowGeometry()
        {
            PathGeometry pathGeometry = new PathGeometry();

            PathFigure pathFigure = new PathFigure();
            pathFigure.IsClosed = true;
            pathFigure.StartPoint = new Point(4, 4);

            LineSegment lineSegment = new LineSegment();
            lineSegment.Point = new Point(0, 0);
            pathFigure.Segments.Add(lineSegment);

            LineSegment lineSegment2 = new LineSegment();
            lineSegment2.Point = new Point(16, 8);
            pathFigure.Segments.Add(lineSegment2);

            LineSegment lineSegment3 = new LineSegment();
```

```
            lineSegment3.Point = new Point(0, 16);
            pathFigure.Segments.Add(lineSegment3);

            pathGeometry.Figures.Add(pathFigure);
            return pathGeometry;
        }
```

and add the namespace delcaration at the beginning of the file :

```
using System.Windows.Markup;
```

If we compile and execute this program, you will see that now, our nodes and links have distinct appearances (colors, shapes, styles, etc).



Notice the call to the AddNode method with 6 parameters.

```
this.addflow.AddNode(20, 20, 50, 50, "First node", XamlReader.Load(xamlRectangle));
```

The last parameter is the content of the node (do not forget that the Node class derives from the ContentControl class). If you omit this parameter (as in the CreateDiagram1 example), then the content is an ellipse with a black stroke and a white filling color.

Notice however that to specify the content of each node, we had to do it for each node, even if the content is the same.

It is the same thing for the links. For instance, in the previous example, we have defined a blue color for each link.

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.

5.2.5 Default property values

Firs of all, let us replace the xaml code in the file **MainPage.xaml** by the following XAML code:

```
<UserControl x:Class="SilverlightApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:af="clr-
namespace:Lassalle.Silverlight.Flow;assembly=Lassalle.Silverlight.Flow"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
```

```
    <UserControl.Resources>
        <Style TargetType="af:Link" x:Key="linkStyle">
            <Setter Property="Stroke" Value="Blue"/>
            <Setter Property="Foreground" Value="Blue"/>
            <Setter Property="ArrowDstGeometry" Value="M3,4 L0,0 12,4 0,8 Z"/>
        </Style>
    </UserControl.Resources>

    <Grid x:Name="LayoutRoot" Background="White">
        <Border x:Name="Border"
                BorderBrush="Red"
                BorderThickness="2"
                CornerRadius="2" >
            <ScrollViewer x:Name="ScrollViewer"
                          HorizontalScrollBarVisibility="Auto"
                          VerticalScrollBarVisibility="Auto"
                          Padding="1"
                          Background="White"
                          BorderBrush="Transparent"
                          BorderThickness="0"
                          Margin="1"
                          IsTabStop="False"
                          TabNavigation="Once">
                <af:AddFlow x:Name="addflow"
                          Background="White"
                          LinkStyle="{StaticResource linkStyle}" />
            </ScrollViewer>
        </Border>
    </Grid>
</UserControl>
```

As you can see we have added a Resources section containing the definition of a style for links. This style is defined in the AddFlow declaration: `LinkStyle="{StaticResource linkStyle}"`

Now let us replace the CreateDiagram2 method by the following CreateDiagram3 method:

```
private void CreateDiagram3()
{
     string xamlRectangle =
               "<Rectangle
       xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
       Stroke='Blue' StrokeThickness='3' Fill='LightYellow' />";
     string xamlDocument =
               "<Path
       xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
       Stroke='Orange' Fill='LightYellow' Stretch='Fill'
       Data='M 0,0 H 60 V 40 C 30,30 30,50 0,40 Z' />";

      this.addflow.XamlNodeContent = xamlRectangle;

      // Create 3 yellow nodes.
      // The second node is rectangular
      // and the third one has a Document shape style.
      this.addflow.AddNode(20, 20, 50, 50, "First node");
      this.addflow.AddNode(230, 85, 70, 50, "Second node");
      this.addflow.AddNode(20, 160, 50, 50, "Third node",
                           XamlReader.Load(xamlDocument));

      // We use LINQ to select in an array all the nodes of the canvas.
      var nodes = this.addflow.Children.OfType<Node>().ToArray();

      // Create 3 links with distinct colors and arrows
      // The first link is red.
      // The second link has a Bezier style
      // The third link has a "HVH" style.
      Link link = this.addflow.AddLink(nodes[0], nodes[1], "link 1");
      link.Stroke = new SolidColorBrush(Colors.Red);
```

```
        link = this.addflow.AddLink(nodes[1], nodes[1], "link 2");
        link.LineStyle = LineStyle.Bezier;

        link = this.addflow.AddLink(nodes[1], nodes[2], "link 3");
        link.LineStyle = LineStyle.HVH;
    }
```

If we compile and execute this new program, it will create the same diagram. However, our program is smaller because we have defined default property values for nodes and links.

For instance, writing:

```
this.addflow.XamlNodeContent = xamlRectangle;
```

indicates that all the nodes that have the content defined in the xaml string xamlRectangle.

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

Notice that these default properties have also an interactive effect. Not only the nodes created programmatically will have the content defined in the xaml string xamlRectangle but also the nodes created interactively with the mouse. This may be interesting or not, depending on what you intend to do.

For the links, we have defined a style in the file page.xaml:

```
<Style TargetType="af:Link" x:Key="linkStyle">
      <Setter Property="Stroke" Value="Blue"/>
      <Setter Property="Foreground" Value="Blue"/>
      <Setter Property="ArrowDstGeometry" Value="M3,4 L0,0 12,4 0,8 Z"/>
</Style>
```

This style is the default style for the links, as indicated in the Addflow xaml definition

```
<af:AddFlow x:Name="addflow"
            Background="White"
            LinkStyle="{StaticResource linkStyle}" />
```

When you create a link programmatically or interactively, it will have a blue color and its associated label will have also a blue color.

In the DemoFlow sample, there are also many examples of node styles. For instance, in the PageFlowChart.xaml, we define the following style:

```
<Style TargetType="af:Node" x:Key="decisionStyle">
    <Setter Property="InConnectionMode" Value="Pin" />
    <Setter Property="OutConnectionMode" Value="Pin" />
    <Setter Property="ContentTemplate">
        <Setter.Value>
            <DataTemplate>
                <Polygon Stroke="Black" Stretch="Fill"
                                        Points="0,20 30,0 60,20 30,40">
                    <Polygon.Fill>
                        <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStop Color="White" Offset="0" />
                                <GradientStop Color="Yellow" Offset="1" />
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Polygon.Fill>
                </Polygon>
            </DataTemplate>
        </Setter.Value>
    </Setter>
```

```
</Style>
```

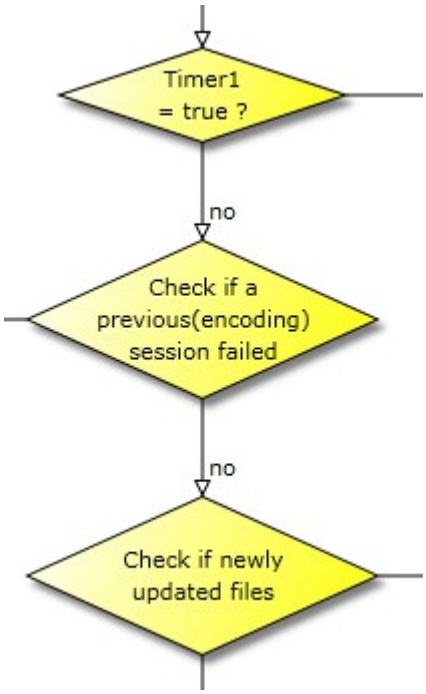
We could have decided to set this style as the default node style in the AddFlow control xaml definition:

```
<af:AddFlow x:Name="addflow"
            Background="White"
            LinkStyle="{StaticResource linkStyle}"
           NodeStyle="{StaticResource decisionStyle}" />
```

However, as other nother node styles are defined, we have preferred to use this style in the code:

```
this.addflow.NodeStyle = this.Resources["decisionStyle"] as Style;
this.addflow.AddNode(112, 128, 144, 48, "Timer1\r\n= true ?",
                     XamlReader.Load(xamlLosange));
this.addflow.AddNode(96, 224, 176, 80,
                     "Check if a\r\nprevious(encoding)\r\nsession failed",
                     XamlReader.Load(xamlLosange));
this.addflow.AddNode(96, 352, 176, 80,
                     "Check if newly\r\nupdated files",
                     XamlReader.Load(xamlLosange));
```

We set the default node style and then we create 3 nodes. Those 3 nodes will have this style as demonstrated in the following figure:

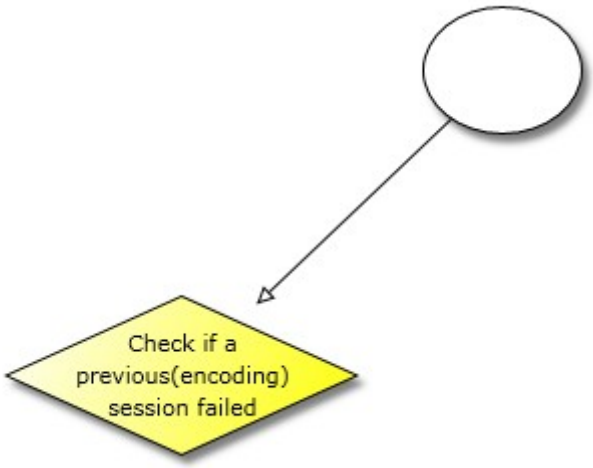


Notice that we could have also choosen to assign the style to each node instead of using the AddFlow node style property:

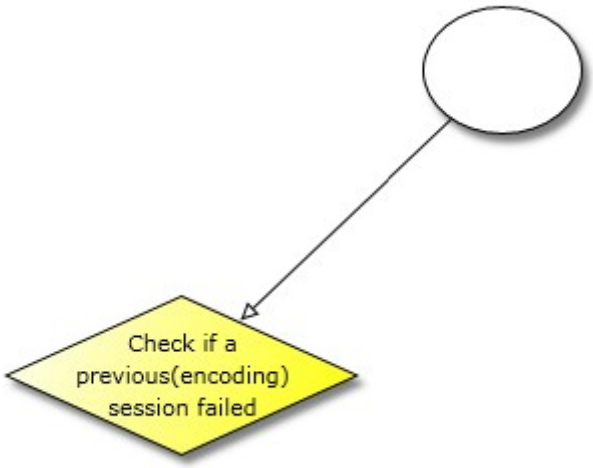
```
Node node1 = this.addflow.AddNode(112, 128, 144, 48,
             "Timer1\r\n= true ?",
             XamlReader.Load(xamlLosange));
node1.Style = this.Resources["decisionStyle"] as Style;
Node node2 = this.addflow.AddNode(96, 224, 176, 80,
             "Check if a\r\nprevious(encoding)\r\nsession failed",
             XamlReader.Load(xamlLosange));
node2.Style = this.Resources["decisionStyle"] as Style;
Node node3 = this.addflow.AddNode(96, 352, 176, 80,
             "Check if newly\r\nupdated files",
             XamlReader.Load(xamlLosange));
```

```
node3.Style = this.Resources["decisionStyle"] as Style;
```

As the style defines the node shape to be a losange, you may think that it is not necessary to define the content to be losange. And in this case, it is true. But it is not always the case. In the case where you do not use pins to create links (node.InConnectionMode = ConnectionMode.Center), a link should be directed towards the destination node center and should stop at the node border. But if you do not define the node content to be a losange, its content is considered to be an ellipse (the default content) and you obtain a link as in the the following diagram,



If you define the content to be a losange (you don't need to specify brush), the you would obtain:



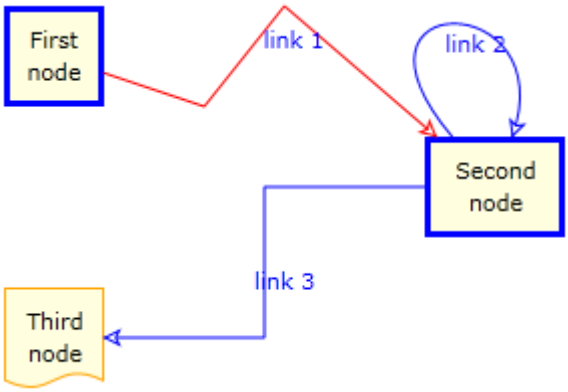
5.2.6 Stretching the links

We would like to add segments to our links. The following CreateDiagram4 method demonstrates how to do that.

```
private void CreateDiagram4()
 {
     string xamlRectangle =
               "<Rectangle
       xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
       Stroke='Blue' StrokeThickness='3' Fill='LightYellow' />";
     string xamlDocument =
```

```
            "<Path
         xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
         Stroke='Orange' Fill='LightYellow' Stretch='Fill'
         Data='M 0,0 H 60 V 40 C 30,30 30,50 0,40 Z' />";

      this.addflow.XamlNodeContent = xamlRectangle;

      // Create 3 yellow nodes.
      // The second node is rectangular
      // and the third one has a Document shape style.
      this.addflow.AddNode(20, 20, 50, 50, "First node");
      this.addflow.AddNode(230, 85, 70, 50, "Second node");
      this.addflow.AddNode(20, 160, 50, 50, "Third node",
                           XamlReader.Load(xamlDocument));

      // We use LINQ to select in an array all the nodes of the canvas.
      var nodes = this.addflow.Children.OfType<Node>().ToArray();

      // Create 3 links with distinct colors and arrows
      // The first link is red.
      // The second link has a Bezier style
      // The third link has a "HVH" style.
      Link link = this.addflow.AddLink(nodes[0], nodes[1], "link 1");
      link.Stroke = new SolidColorBrush(Colors.Red);

      // Add 2 points (therefore 2 segments) to this first link
      link.AddPoint(new Point(120, 70));
      link.AddPoint(new Point(160, 20));

      link = this.addflow.AddLink(nodes[1], nodes[1], "link 2");
      link.LineStyle = LineStyle.Bezier;

      // Stretch this reflexive link
      link.SetPoint(new Point(180, 10), 1);
      link.SetPoint(new Point(300, 10), 2);

      link = this.addflow.AddLink(nodes[1], nodes[2], "link 3");
      link.LineStyle = LineStyle.HVH;
}
```

If we compile and execute this program, you will see that first link has now 3 segments and the reflexive link is bigger.



To add segments to a link or to alter its shape, you have to use the **AddPoint** method collection of the link.

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, using the **SetPoint** method.

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

- After its insertion in the diagram, a link has at least 2 points.
- You cannot remove these 2 points. The Count property of the Points collection is always superior or equal to 2.
- You can add or delete points only if the link line style is Polyline or Spline. In other case, the number of link points is fixed. For instance, if the link line style is Bezier, then it has 4 points in any case.
- You can change the first point of the Points collection only if the **OutConnectionMode** property of the origin node is NOT set to **ConnectionMode.Cente**r.
- You can change the last point of the Points collection only if the **InConnectionMode** property of the destination node is NOT set to **ConnectionMode.Cente**r.
- 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 **ImageUri** property.

The WorkFlow example in the DemoFlow sample shows how to that. It first creates 3 Image objects (see the CreateWorkFlowDiagramExample method)

```
Uri imageUri1 = new Uri("../Images/phone.png", UriKind.Relative);
Uri imageUri2 = new Uri("../Images/mail.png", UriKind.Relative);
Uri imageUri3 = new Uri("../Images/disc.png", UriKind.Relative);
```

Then, to assign an image to a node, you have just to write for instance:

```
node.ImageUri = imageUri1;
```

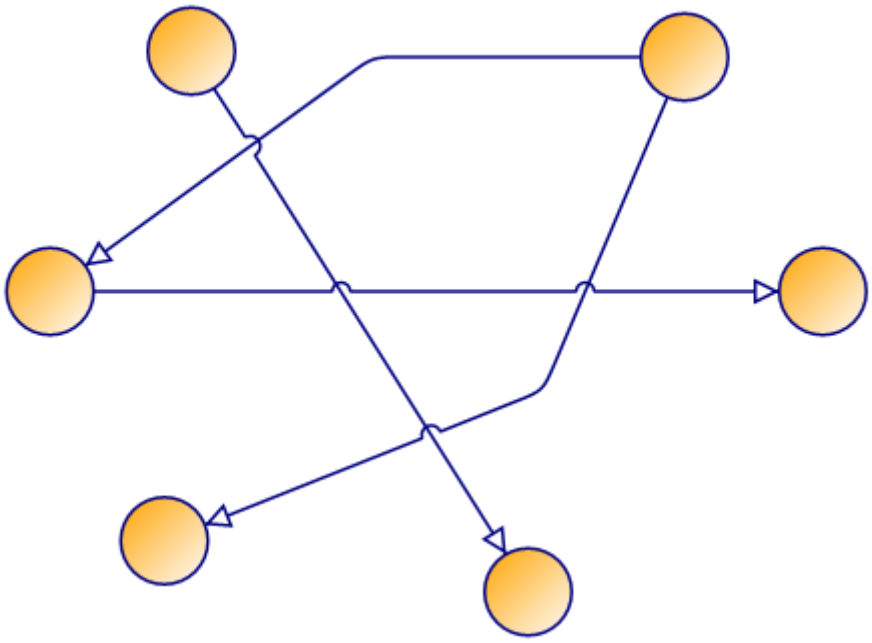
5.4 Displaying link intersections

AddFlow uses a poweful algorithm to find intersections between the link segments.

If the **CanShowJumps** property of the AddFlow control is true then jumps will be displayed at the intersection of this link with other links.

However, this will not work if the link is a curved link (Bezier or Spline).

The size of jumps is determined by the value of the **JumpSize** property of the link. If this value is 0, then no jump will be displayed.



5.5 Displaying link labels

The following table gives the list of all properties available to manage the link labels.

Text	Sets or returns the text label associated with the link.
TextBlock	Returns the TextBlock associated with the link and used to display the link label
TextBlockBorder	Returns the Border containing the link TextBlock
TextPlacementMode	Returns/sets the TextPlacementMode style of the text of the link.
TextAngleMode	Sets or returns the TestAngleMode style of the label on the link.
TextXPosition	Returns/sets the x coordinate of the position of the link text.
TextYPosition	Returns/sets the y coordinate of the position of the link text.
TextXOffset	Returns/sets the x value of the offset of the link text.
TextYOffset	Returns/sets the y value of the offset of the link text.

In the default xaml description of a link, the following code defines the link label:

```
<Border Name="TextBlockBorder" Canvas.Left="{TemplateBinding TextXPosition}"
                               Canvas.Top="{TemplateBinding TextYPosition}">
  <TextBlock Name="TextBlock" Text="{TemplateBinding Text}" IsHitTestVisible="True" />
</Border>
```

As you can see, the label string (given by the **Text** property) is displayed in a TextBlock (**TextBlock** property) contained in a Border control (**TextBlockBorder** property). The Border control is placed at a position given by the **TextXPosition** and **TextYPosition** properties.

By default, the TextBlockBorder is not visible so you just see the text. But you may choose to make it visible to emphasize the label or make it opaque instead of transparent. This is demonstrated in the «State Diagram» example of the demo (see the files **PageStateDiagram.xaml** and **PageStateDiagram.xaml.cs**)

The values of the TextXPosition and TextYPosition properties is determined by the **TextPlacementMode, TextAngleMode, TextXOffset and TextYOffset properties.**

The TextPlacementMode property may have two values:

- **MiddleSegment**. The text is placed at the middle of the medium segment of the link.
- **MiddleLine**. The text is placed at the middle of the link line (default)

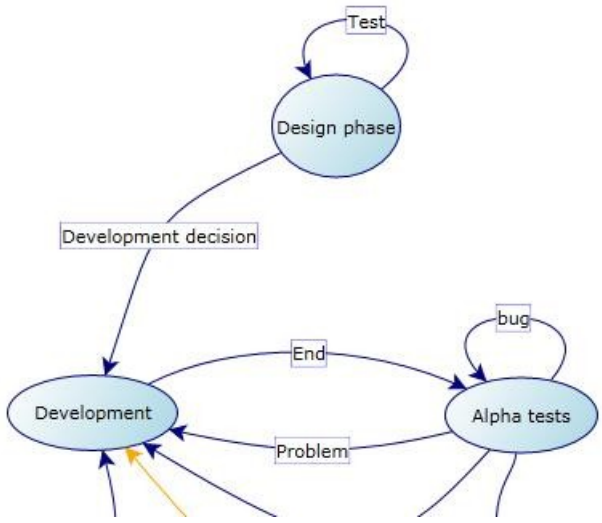
The TextAngleMode property may have two values:

- **Horizontal**. The text is displayed horizontally, whatever the angle of the link may be (default).
- **LineAngle**. The text is rotated to follow the link angle.

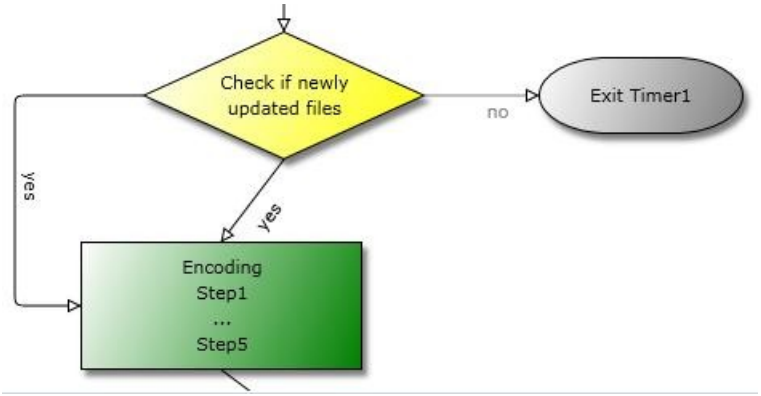
Both TextPlacementMode, TextAngleMode properties determine a position for link label. Then, you may alter this position using the **TextXOffset** and **TextYOffset** properties.

Note that the default values for TextXOffset and TextYOffset is 10. If you wish to display the label above the link, then should set both properties to 0.

In the demo program provided with AddFlow, only the first example («State Diagram»: see the files **PageStateDiagram.xaml** and **PageStateDiagram.xaml.cs**) is using values distinct from the default values for the **TextXOffset** and **TextYOffset** properties.



In the second example («Flowchart Diagram»: see the files **PageFlowchart.xaml** and **PageFlowchart.xaml.cs**), the property TextAngleMode is to LineAngle so that the labels are rotated to follow the link angle.



5.6 Selection of items

5.6.1 Interactive selection

You can select an item (a node or a link) interactively by clicking it with the mouse.

You can also select several items interactively 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 **MouseSelection** property is set to MouseSelection.Selection. 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.

FAQ: How to select 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 or a Spline curve, then you have just to click near the curve.

5.6.2 Programmatic selection: ISelectable interface

Both Node and Link classes implement the **ISelectable** interface:

- A node or a link can be selected either interactively, either programmatically using its **IsSelected** property, for instance:

```
node1.IsSelected = true;
node2.IsSelected = true;
link1.IsSelected = true;
```

- If the **IsSelectable** property of a node or a link is false, then it is no more possible to select it.

5.6.3 SelectedItems collection

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

```
// Make each selected nodes red
IEnumerable<Node> selnodes = this.addflow.SelectedItems.OfType<Node>();
foreach (Node node in selnodes)
       node.BorderBrush = new SolidColorBrush(Colors.Red);
```

WARNING: there is a difference with previous versions (the ActiveX version or the Windows Form version). In the Silverlight version, there is not any notion of a “current” (or “active”) item and therefore there is not any property like SelectedNode, SelectedLink or SelectedItem. When several items (nodes or links) are selected, you cannot designate one that could be the current one. However, you can implement a “current item” feature. You have just to emphasize the first item of the SelectedItems collection. This is demonstrated in the **State diagram** example of the DemoFlow sample.

5.6.4 Selection event

The **SelectionChanged** event is fired each time the selection status of an item is changed.

However, you can avoid that by setting the **CanSendSelectionChangedEvent** property to false.

5.7 In-place edition for nodes

If the **CanEditMode** property of the AddFlow control is true and if the **IsEditable** property of the node is also true (which is the case by default), then you can interactively edit the text inside the node. If it is the case, then the **IsInEditMode** property of the node is true.

Notice that the Node class offers also 3 new methods for doing in-place edition: **BeginEdit**, **EndEdit** and **CancelEdit**.

And the Addflow class offers also 3 new events: **BeforeEdit**, **AfterEditEdit** and **CancelEdit**.

Any in-place edition can be undone and redone.

5.8 Diagram navigation

AddFlow provides four and only four properties to navigate in a diagram (“Network traversals”). Notice that these properties described here are demonstrated in the “**Navig**” diagrams in the **DemoFlow** sample provided with AddFlow.

We have already spoken of these properties.

- **Children**. It is the collection of all the objects contained in the AddFlow canvas. Therefore it includes all nodes and links but also all other objects you may add in the diagram. See the paragraph 5.1 to see how to retrieve a list of nodes or links using the LINQ technology.
- **Links**. It is the collection of links coming to or leaving a node.
- **Org**. It is the origin node of a link.
- **Dst**. It is the destination node of a link.

5.9 Zooming

5.9.1 Programmatic zoom

The **Zoom** property allows zooming a diagram. It is a double value representing the zoom factor. Its default value is equal to 1. Notice that the zoom is isotropic ensuring a 1:1 aspect ratio.

The **ZoomRectangle** method allows zooming and scrolling a view to fit a specified rectangular portion of the diagram. The zoom is isotropic.

TIP: 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?

You can implement this feature using the **ZoomRectangle** method and the **Extent** property:

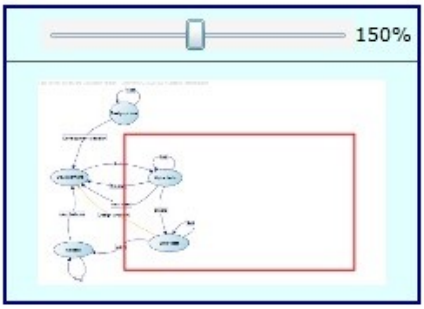
```
Size size = this.addflow.Extent;
this.addflow.ZoomRectangle(new Rect(0, 0, size.Width, size.Height));
```

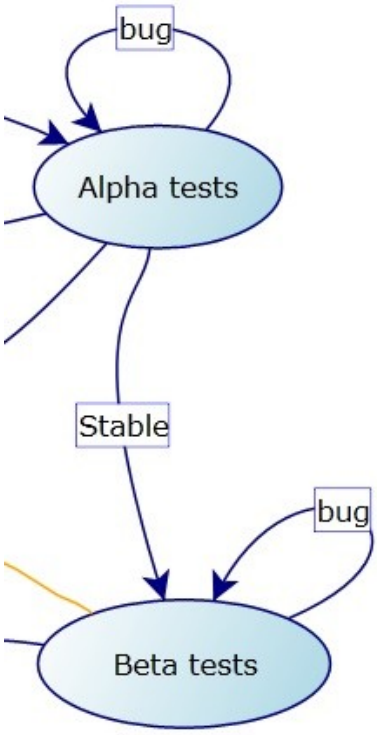
5.9.2 Interactive zoom

Notice that you may also zoom the diagram interactively with the mouse if the **MouseSelection** property of the AddFlow control is set to **MouseSelection.Zoom**. The user brings the mouse cursor into the AddFlow control, press the left button, move the mouse (which draws a rectangular area) and then release the left button: this has the effect of zooming and scrolling the view to fit the rectangular area.

5.9.3 Bird view

The **BirdView** property returns an object managing a bird view popup window allowing scrolling and zooming the diagram.





The **Show** and **Hide** methods of the BirdView object allow showing or hiding the bird view popup window.

5.10 Serialization

AddFlow does not provide any serialization feature. However, the “Save XAML” example in the DemoFlow sample shows how to deals with serialization (see the file **PageSerialize.xaml.cs**). More precisely, it shows how to store data on the Client, using:

- the **Isolated Storage** area
- XML
- LINQ

You know that LINQ can be used to perform queries against XML. For that, you need to add a refererence to the System.Xml.Linq DLL.

The methods to save/load a diagram are the followings:

```
static void SaveLocalFile(AddFlow addflow)
{
    using (IsolatedStorageFile store =
                    IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream fs = store.CreateFile(@"diagram.xml"))
        {
            XElement xElement = PageSaveXaml.SaveXML(addflow);
            xElement.Save(fs);
        }
    }
}

static void LoadLocalFile(AddFlow addflow)
```

```
{
    using (IsolatedStorageFile store =
                    IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream fs =
                                store.OpenFile(@"diagram.xml", FileMode.Open))
        {
            XElement xElement = XElement.Load(fs);
            PageSaveXaml.LoadXML(addflow, xElement);
        }
    }
}
```

The **SaveXML** file allows serializing the diagram in a XML file. The LoadXML file allows creating the diagram from the XML file.

Notice that in this example, we do not save the xaml content of a nodeor a link. We just save a type that is used when re-creating the diagram from the XML file. We use an attached property to assign a type to a node and another attached property to assign a type to a link.

For instance when creating the diagram, we use the following line of code to assign a type to a node:

```
PageSaveXaml.SetNodeType(node, NodeType.Ellipse);
```

This type is saved when serializing the node as you can see in the GetNodesXML method (for links, it is the method GetLinksXML)

```
static XElement GetNodesXML(AddFlow addflow, IEnumerable<Node> nodes)
{
    return new XElement("Nodes",
        from node in nodes
        select new XElement("Node",
            new XAttribute("Left", node.Location.X),
            new XAttribute("Top", node.Location.Y),
            new XAttribute("Width", node.Size.Width),
            new XAttribute("Height", node.Size.Height),
            new XAttribute("ZOrder", addflow.Children.IndexOf(node)),
            new XAttribute("NodeType", PageSaveXaml.GetNodeType(node)),
            string.IsNullOrEmpty(node.Text) ?
                                null : new XElement("Text", node.Text)
            )
    );
}
```

When de-serializing the diagram, the type is used to associate the correct content to a node. This is demonstrated in the following **DeserializeNode** method.

```
private static Node DeserializeNode(AddFlow addflow, XElement nodeXML)
{
    double width = Double.Parse(nodeXML.Attribute("Width").Value,
        CultureInfo.InvariantCulture);
    double height = Double.Parse(nodeXML.Attribute("Height").Value,
        CultureInfo.InvariantCulture);
    double left = Double.Parse(nodeXML.Attribute("Left").Value,
        CultureInfo.InvariantCulture);
    double top = Double.Parse(nodeXML.Attribute("Top").Value,
        CultureInfo.InvariantCulture);
    int zorder = int.Parse(nodeXML.Attribute("ZOrder").Value);
    NodeType nodeType = (NodeType)Enum.Parse(typeof(NodeType),
                            nodeXML.Attribute("NodeType").Value, false);

    Node node = addflow.AddNode(left, top, width, height, null);
    if (node != null)
    {
        PageSaveXaml.SetZorderSerial(node, zorder);
        PageSaveXaml.SetNodeType(node, nodeType);
```

```
            if (nodeXML.Element("Text") != null)
                node.Text = nodeXML.Element("Text").Value;
            switch (nodeType)
            {
                case NodeType.Ellipse:
                    node.Content = XamlReader.Load(PageSaveXaml.xamlEllipse);
                    break;
                case NodeType.Losange:
                    node.Content = XamlReader.Load(PageSaveXaml.xamlLosange);
                    break;
                case NodeType.Document:
                    node.Content = XamlReader.Load(PageSaveXaml.xamlDocument);
                    break;
            }
        }
        return node;
    }
```

5.11 Exporting a diagram in XAML

AddFlow itself does not provide any exporting feature.

However, Silverlight allows you rendering the content of a control to a bitmap. The key to generating a bitmap is the WriteableBitmap class found in System.Windows.Media.Imaging as in the following code:

```
WriteableBitmap wb = new WriteableBitmap(this.addflow, null);
this.image1.Source = wb;
```

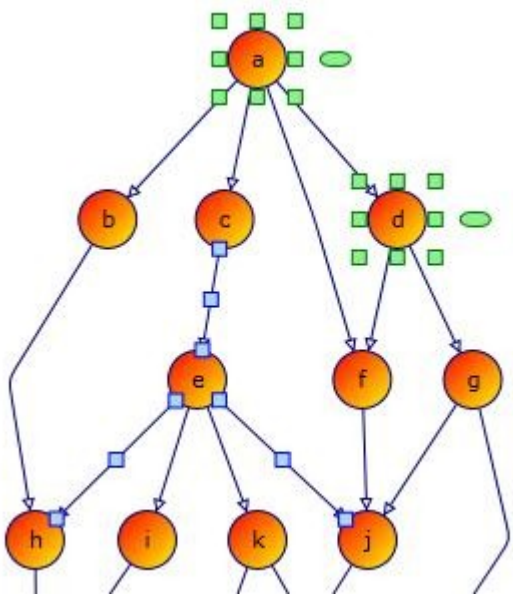
5.12 Printing

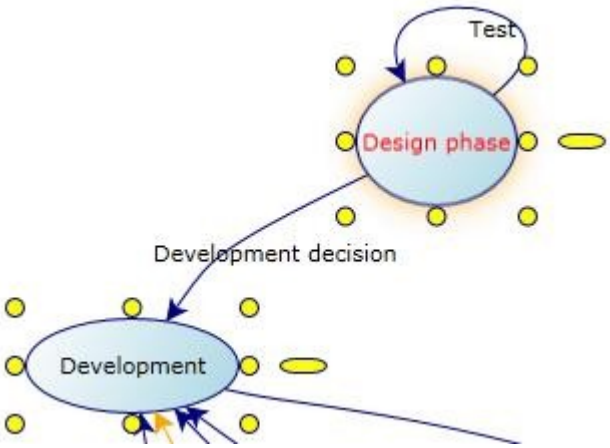
AddFlow itself does not provides any printing feature. However, the DemoFlow sample shows how to print a diagram in one page.

5.13 Customizing the user interface

5.13.1 Customizing the handles

If you look at the different diagram examples provided in the DemoFlow sample, you will see that the handles used to resize a node or to stretch a link are not same.





For changing the style of the node resizing handles, you can use the **ResizeHandleStyle** property of AddFlow. For changing the style of the rotation handle, you can use the **RotateHandleStyle** property of AddFlow. And for changing the style of the link stretching handles, you can use the **StretchHandleStyle** property of AddFlow. The type of these properties is **Style**. And you can define a style in xaml.

For instance the following styles are defined in the file PageStateDiagram.xaml:

```
<Style TargetType="af:ResizeHandle" x:Key="resizeHandle">
```

```
    <Setter Property="Width" Value="10"/>
    <Setter Property="Height" Value="10"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="af:ResizeHandle">
                <Grid>
                    <vsm:VisualStateManager.VisualStateGroups>
                        <vsm:VisualStateGroup x:Name="CommonStates">
                            <vsm:VisualStateGroup.Transitions>
                                <vsm:VisualTransition
                                  GeneratedDuration="00:00:00.1" To="MouseOver"/>
                            </vsm:VisualStateGroup.Transitions>
                            <vsm:VisualState x:Name="Normal"/>
                            <vsm:VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <DoubleAnimationUsingKeyFrames
                                        Storyboard.TargetName="contentPresenter"
                                        Storyboard.TargetProperty="Opacity">
                                        <SplineDoubleKeyFrame
                                              KeyTime="00:00:00" Value="0.5"/>
                                    </DoubleAnimationUsingKeyFrames>
                                </Storyboard>
                            </vsm:VisualState>
                        </vsm:VisualStateGroup>
                    </vsm:VisualStateManager.VisualStateGroups>

                    <ContentPresenter x:Name="contentPresenter">
                        <Ellipse Stretch="Fill" Fill="Yellow" Stroke="Navy"/>
                    </ContentPresenter>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style TargetType="af:RotateHandle" x:Key="rotateHandle">
    <Setter Property="Width" Value="24"/>
    <Setter Property="Height" Value="8"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="af:RotateHandle">
                <Grid>
                    <vsm:VisualStateManager.VisualStateGroups>
                        <vsm:VisualStateGroup x:Name="CommonStates">
                            <vsm:VisualStateGroup.Transitions>
                                <vsm:VisualTransition
                                 GeneratedDuration="00:00:00.1" To="MouseOver"/>
                            </vsm:VisualStateGroup.Transitions>
                            <vsm:VisualState x:Name="Normal"/>
                            <vsm:VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <DoubleAnimationUsingKeyFrames
                                        Storyboard.TargetName="contentPresenter"
                                        Storyboard.TargetProperty="Opacity">
                                        <SplineDoubleKeyFrame
                                              KeyTime="00:00:00" Value="0.5"/>
                                    </DoubleAnimationUsingKeyFrames>
                                </Storyboard>
                            </vsm:VisualState>
                        </vsm:VisualStateGroup>
                    </vsm:VisualStateManager.VisualStateGroups>

                    <ContentPresenter x:Name="contentPresenter">
                        <Path Data="M 2,4 A 2,2 0 0 1 2,0 H 4 A 2,2 0 0 1 4,4 Z"
                              Fill="Yellow" Stroke="Navy" Stretch="Fill" />
                    </ContentPresenter>
                </Grid>
            </ControlTemplate>
```

```
        </Setter.Value>
    </Setter>
</Style>

<Style TargetType="af:StretchHandle" x:Key="stretchHandle">
    <Setter Property="Width" Value="10"/>
    <Setter Property="Height" Value="10"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="af:StretchHandle">
                <Grid>
                    <vsm:VisualStateManager.VisualStateGroups>
                        <vsm:VisualStateGroup x:Name="CommonStates">
                            <vsm:VisualStateGroup.Transitions>
                                <vsm:VisualTransition
                                  GeneratedDuration="00:00:00.1" To="MouseOver"/>
                            </vsm:VisualStateGroup.Transitions>
                            <vsm:VisualState x:Name="Normal"/>
                            <vsm:VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <DoubleAnimationUsingKeyFrames
                                        Storyboard.TargetName="contentPresenter"
                                        Storyboard.TargetProperty="Opacity">
                                        <SplineDoubleKeyFrame
                                            KeyTime="00:00:00" Value="0.5"/>
                                    </DoubleAnimationUsingKeyFrames>
                                </Storyboard>
                            </vsm:VisualState>
                        </vsm:VisualStateGroup>
                    </vsm:VisualStateManager.VisualStateGroups>

                    <ContentPresenter x:Name="contentPresenter">
                        <Ellipse Stretch="Fill" Fill="Yellow" Stroke="Navy"/>
                    </ContentPresenter>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

This style says that the handle is a yellow ellipse and that when the mouse enters over this handle, the opacity of this handle changes.

Then, to use these styles, you have just to assign them to the **ResizeHandleStyle**, **RotateHandleStyle** and **StretchHandleStyle** properties of the AddFlow control:

```
<af:AddFlow x:Name="addflow" Grid.Row="1" Background="White"
            ResizeHandleStyle="{StaticResource resizeHandle}"
            RotateHandleStyle="{StaticResource rotateHandle}"
            StretchHandleStyle="{StaticResource stretchHandle}"
            LinkStyle="{StaticResource linkBezier}"
             SelectionChanged="addflow_SelectionChanged" />
```

5.13.2 Customizing the connection of links to a node

5.13.2.1 Generalities

The way a link is connected to a node is managed by this node. Each node has 4 properties for this purpose: **InConnectionMode**, **OutConnectionMode, Connector and ConnectorStyle**.

The type of the InConnectionMode and OutConnectionMode properties is **ConnectionMode**. It is an enumeration that defines how the link is connected to the node.

Member	Comment

Center	Default. The link is directed towards the centre of the node.
Anywhere	The link last (or first) point can be placed anywhere.
Pin	A pin a must be use to connect a link to the node.

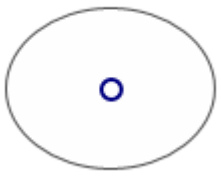
The **Connector property** is read-only. It returns a **Connector** object. This object manages the set of the pins that can be used to connect links. The Connector class exposes the **Pins** property that returns the collection of **Pin** objects used by the node. By default, the node connector has only one pin placed at the center of the node. However, as explained later in this paragraph, you may customize a connector so that it contains several pins placed anywhere in the node.

The Pin class exposes the 3 following properties:

Property	Type	Description
Position	Point	Returns/sets the position of a pin
In	bool	Determines if the pin accepts "in" links
Out	bool	Determines if the pin accepts "out" links

5.13.2.2 Customizing pins

When the mouse is over a node, a pin (connector) is displayed at the center of the node. It allows creating a link leaving this node towards another node. By default this pin is a circle but you can customize it.



For changing the style of the pin, you can use the **PinStyle** property of AddFlow. The type of this property is **Style**. And you can define a style in xaml.

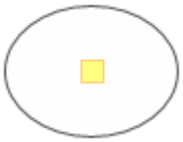
For instance, the following style is defined in the file PageStateDiagram.xaml:

```
<Style TargetType="af:Pin" x:Key="pinStyle">
    <Setter Property="Width" Value="12" />
    <Setter Property="Height" Value="12" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="af:Pin">
                <Grid Name="PART_Root" >
                    <Rectangle Stroke="Orange" Fill="Yellow" Opacity="0.5"/>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Then, to use these styles, you have just to assign them to the PinStyle property of the AddFlow control:

```
<af:AddFlow x:Name="addflow" Grid.Row="1" Background="White"
            ResizeHandleStyle="{StaticResource resizeHandle}"
            StretchHandleStyle="{StaticResource stretchHandle}"
            PinStyle="{StaticResource pinStyle}"
```

```
            LinkStyle="{StaticResource linkStyle}"/>
```



Instead of being a circle, it is a yellow rectangle.

5.13.2.3 Customizing the Connector

By default, the node connector contains only one pin placed at the center of the node. However, you can change this default style, using the ConnectorStyle property of AddFlow. This demonstrated in the **PageFlowChart.xaml** file where you can find the definition of the following style:

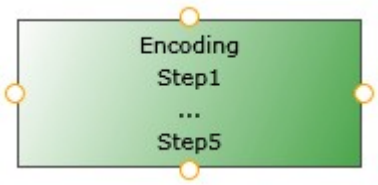
```
<Style TargetType="af:Connector" x:Key="connectorStyle">
    <Setter Property="HorizontalContentAlignment" Value="Stretch" />
    <Setter Property="VerticalContentAlignment" Value="Stretch" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="af:Connector">
                <Grid Margin="-6" Name="PART_Root" >
                    <af:Pin x:Name="pin0"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                    <af:Pin x:Name="pin1"
                        VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin2"
                        VerticalAlignment="Top" HorizontalAlignment="Center" />
                    <af:Pin x:Name="pin3"
                        VerticalAlignment="Bottom" HorizontalAlignment="Center" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

WARNING: The pins must have a name and this name must be “pin0”, “pin1”, “pin2” etc...

Then, to use this style, you have just to assign it to the ConnectorStyle property of the AddFlow control:

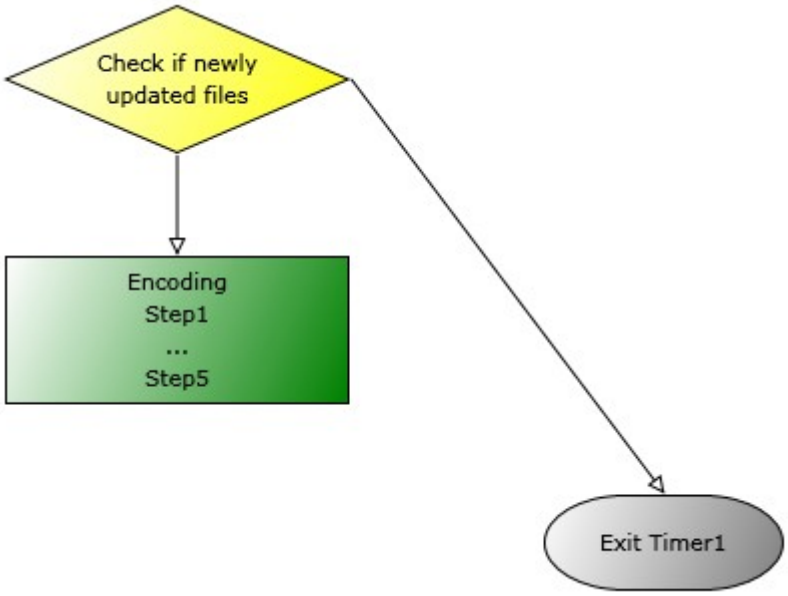
```
<af:AddFlow x:Name="addflow" Grid.Row="1" Background="White"
            ResizeHandleStyle="{StaticResource resizeHandle}"
            StretchHandleStyle="{StaticResource stretchHandle}"
            LinkStyle="{StaticResource linkStyle}"
            ConnectorStyle="{StaticResource connectorStyle}"
            PinStyle="{StaticResource pinStyle}" />
```

The resulting connector has four pins.



Notice however that the InConnectionMode and OutConnectionMode properties of the node must be set to ConnectionMode.Pin to be able to use those pins to create links. For instance, the definition of the node style must contains the lines:

```
<Setter Property="InConnectionMode" Value="Pin" />
<Setter Property="OutConnectionMode" Value="Pin" />
```



(Remember that the pins are visible only when the mouse is over the node)

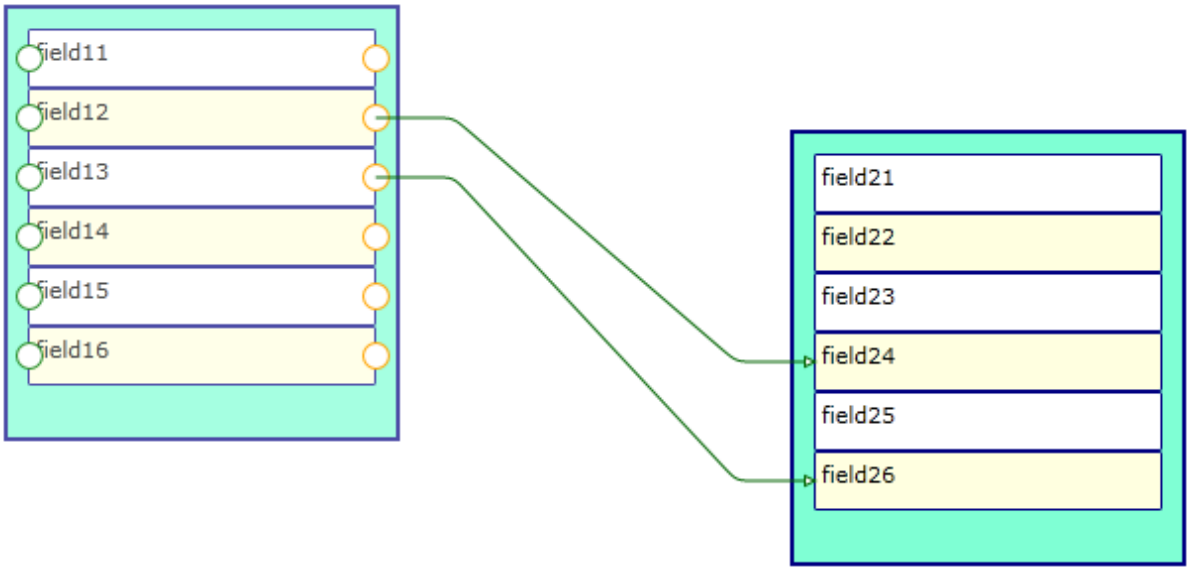
The file **PageConnectors.xaml** provide another example where each node has 6 "out" pins and 6 "in" pins.

```
<Style TargetType="af:Connector" x:Key="connectorStyle">
    <Setter Property="HorizontalContentAlignment" Value="Stretch" />
    <Setter Property="VerticalContentAlignment" Value="Stretch" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="af:Connector">
                <Grid Margin="5.5,12,5.5,12" Name="PART_Root" >
                    <Grid.RowDefinitions>
                        <RowDefinition Height="30" />
                        <RowDefinition Height="30" />
                        <RowDefinition Height="30" />
                        <RowDefinition Height="30" />
                        <RowDefinition Height="30" />
                        <RowDefinition Height="30" />
                    </Grid.RowDefinitions>
                    <af:Pin x:Name="pin0" Grid.Row="0"
                          Style="{StaticResource outPinStyle}"
                          VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin1" Grid.Row="1"
                          Style="{StaticResource outPinStyle}"
                          VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin2" Grid.Row="2"
                          Style="{StaticResource outPinStyle}"
                          VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin3" Grid.Row="3"
                          Style="{StaticResource outPinStyle}"
                          VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin4" Grid.Row="4"
                          Style="{StaticResource outPinStyle}"
                          VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin5" Grid.Row="5"
```

```
                        Style="{StaticResource outPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Right" />
                    <af:Pin x:Name="pin6" Grid.Row="0"
                        Style="{StaticResource inPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                    <af:Pin x:Name="pin7" Grid.Row="1"
                        Style="{StaticResource inPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                    <af:Pin x:Name="pin8" Grid.Row="2"
                        Style="{StaticResource inPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                    <af:Pin x:Name="pin9" Grid.Row="3"
                        Style="{StaticResource inPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                    <af:Pin x:Name="pin10" Grid.Row="4"
                        Style="{StaticResource inPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                    <af:Pin x:Name="pin11" Grid.Row="5"
                        Style="{StaticResource inPinStyle}"
                        VerticalAlignment="Center" HorizontalAlignment="Left" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Here again, the InConnectionMode and OutConnectionMode properties of the node must be set to ConnectionMode.Pin. This can be done in the definition of the node style:

```
<Style TargetType="af:Node" x:Key="nodeStyle">
    <Setter Property="InConnectionMode" Value="Pin" />
    <Setter Property="OutConnectionMode" Value="Pin" />
    <Setter Property="ContentTemplate">
        <Setter.Value> …
```



The “out” pins are orange whereas the “in” pins are green.

6) Avanced topics

6.1 Undo/Redo

6.1.1 General features

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

6.1.2 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.

6.1.3 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 **Clear** method.

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

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

6.1.4 What can be undone and redone?

The rule is the following: every interactive 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.

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.

WARNING: there is a difference with the previous versions (ActiveX version or the Windows Form version). In the Silverlight version, only the interactive actions can be undone and redone. In the previous versions, all actions changing a node or a link could be undone or redone. For instance, you could undo a change of the XMoveable property of a node. In the Silverlight version, you will have to write a custom task to be able to undo a change of the IsXMoveable property. This is explained in the following paragraph.

6.1.5 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/Redo buffer with the **SubmitTask** method. The file **PageCustomUndo.xaml.cs** of the DemoFlow project shows how to do that: if you select a node, you can change its text and its text color. And then, you can undo this action.

For doing that, we have created a class deriving from the Task class. This new class implements the task consisting in changing both the node text and node text color (to make the things simple, you can only select 3 colors: blue, red or green)

```
internal class NodePropertiesTask : Task
{
    private string oldText;
    private Brush oldForeground;
    private Node node;

    private NodePropertiesTask() { }

    internal NodePropertiesTask(Node node, string oldText, Brush oldForeground)
    {
        this.Code = (Lassalle.Silverlight.Flow.Action)1002;
        this.node = node;
        this.oldText = oldText;
        this.oldForeground = oldForeground;
    }

    public override void Redo()
    {
        this.Undo();
    }

    public override void Undo()
    {
        string oldText = this.node.Text;
        this.node.Text = this.oldText;
        this.oldText = oldText;

        Brush oldForeground = this.node.Foreground;
        this.node.Foreground = this.oldForeground;
        this.oldForeground = oldForeground;
    }
}
```

When the user click on the “Submit” button, the text and the text color are assigned to the selected node.The code is the following:

```
private void CommandSubmit_Click(object sender, RoutedEventArgs e)
{
    if (this.addflow.SelectedItems.Count > 0)
    {
        ISelectable item = this.addflow.SelectedItems[0];
        if (item is Node)
        {
            Node node = item as Node;
            this.addflow.TaskManager.SubmitTask(
                    new NodePropertiesTask(node, node.Text, node.Foreground));
            node.Text = textBoxNodeText.Text;
            switch (comboBoxColor.SelectedIndex)
            {
                case 0:
                    node.Foreground = new SolidColorBrush(Colors.Blue);
                    break;
                case 1:
                    node.Foreground = new SolidColorBrush(Colors.Red);
                    break;
```

```
                case 2:
                    node.Foreground = new SolidColorBrush(Colors.Green);
                    break;
            }
        }
    }
}
```

As you can see, before the node receives new values for its Text and Foreground properties, you can find the following code line:

```
this.addflow.TaskManager.SubmitTask(
            new NodePropertiesTask(node, node.Text, node.Foreground));
```

This causes the new custom action to be registered in the list of tasks (undo/redo buffer).

6.1.6 Undo/Redo API

The following table gives the list of all properties and methods available to manage the undo/redo feature.

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.
Clear	Clears the undo/redo buffer.
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.
RedoItem	Returns the item involved in the next redoable action
SkipUndo	Determines whether the following actions are recorded in the undo manager.
SubmitTask	Submit a task (or action) that can be undone and redone.
RemoveLastTask	Remove the last task that has been added in the undo list.
Undo	Undo, if possible, the last action.
UndoCode	Returns the code of the next undoable action.
UndoItem	Returns the item involved in the next undoable action.
UndoLimit	Sets and returns the number of undo commands that can be performed.

6.2 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 which provides** a set of 5 graph layout algorithms:

- *Hierarchic layout*
- *Orthogonal layout*
- *Symmetric layout*
- *Series Parallel layout*
- *Tree layout*

Each of these graph layout algorithms 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, symmetric, orthogonal...).

Remarks

- Currently, LayoutFlow is an AddFlow extension and you cannot use it 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 LayoutFlow 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 **DemoFlow** sample installed with AddFlow shows how to use each graph layout component.
- Reflexive links are not taken into account by layout algorithms. Reflexive links are just translated to follow their origin (and also destination) node.

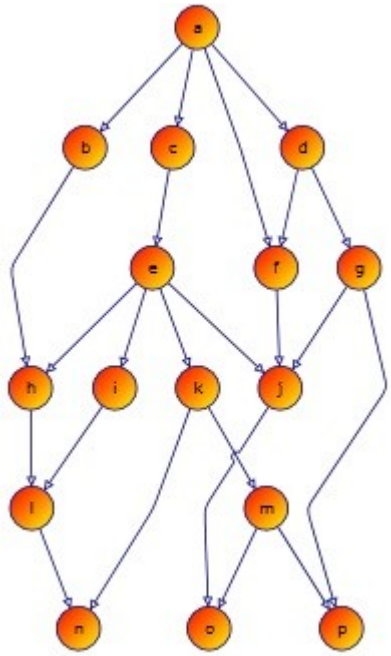
TIP: How to manage so that the layout algorithm applies only to a subset of the graph?

LayoutFlow provides an attached property: **IsLogical**. Only the nodes and links whose IsLogical property is true are involved in each layout. This will allow you to make the layout algorithm to ignore some nodes or links. By default, the IsLogical property is true.

6.2.1 Hierarchic layout

6.2.1.1 Purpose

This algorithm 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.



- Hierarchic layout -

6.2.1.2 Code example

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

```
LayoutFlow.TreeLayout(this.addflow,
             50, // Sets the distance between adjacent levels
             50, // Sets the distance between adjacent nodes
             Orientation.North,
             new Size(20, 20), // Margin
             0); // No limitation in the number of nodes in a level
```

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 layout to this graph. And each bode will be placed at a reasonable position.

6.2.1.3 Limitation

It works with any graph, connected or not.

6.2.1.4 Side Effect

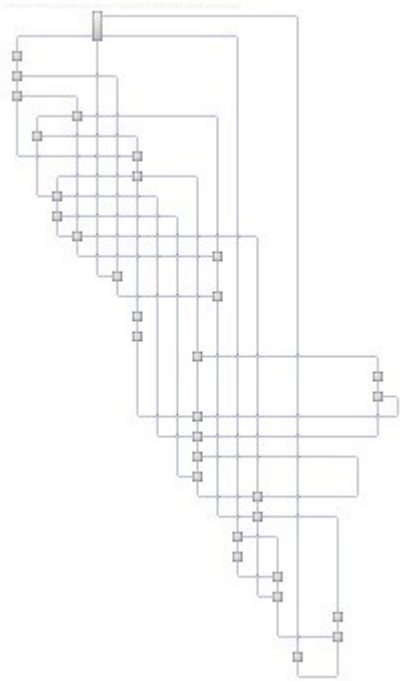
After the layout execution:

- the line style of the links is Polyline
- the InConnectionMode property of the destination node of a link is set to Center.
- the OutConnectionMode property of the origin node of a link is set to Center.

6.2.2 Orthogonal layout

6.2.2.1 Purpose

This algorithm 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.



- Orthogonal layout -

6.2.2.2 Code example

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

```
LayoutFlow.OrthogonalGridLayout(this.addflow,
       Orientation.North,
       new Size(40, 40), // The horizontal and vertical grid size
       nodeSizeRatio, // The node size (in percentage of the grid size)
       new Size(20, 20)); // Margin
```

6.2.2.3 Limitation

It 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.

6.2.2.4 Side Effect

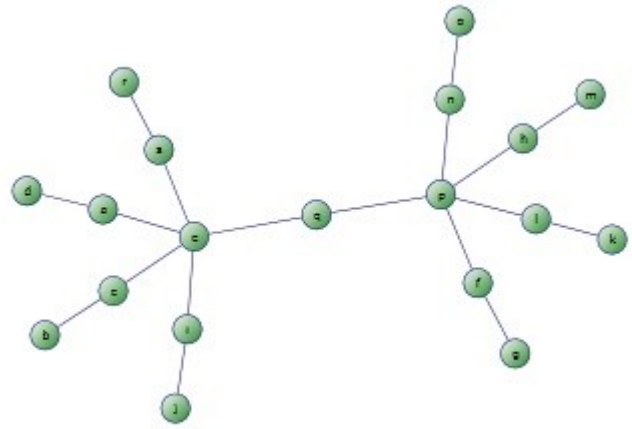
After the layout execution:

- 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 InConnectionMode property of the destination node of a link is set to Anywhere.
- the OutConnectionMode property of the origin node of a link is set to Anywhere.

6.2.3 Symmetric layout

6.2.3.1 Purpose

This algorithm 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. It 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.



- Symmetric layout -

6.2.3.2 Code example

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

```
LayoutFlow.SymmetricLayout(this.addflow,
                    50,                 // Sets the distance between nodes
                    new Size(20, 20)); // Margin
```

6.2.3.3 Limitation

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

6.2.3.4 Side Effect

After the layout execution:

- the line style of the links is Polyline and each link is composed of only one segment.
- the InConnectionMode property of the destination node of a link is set to Center.

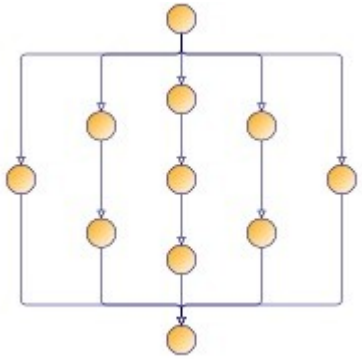
- the OutConnectionMode property of the origin node of a link is set to Center.

6.2.4 Series-parallel layout

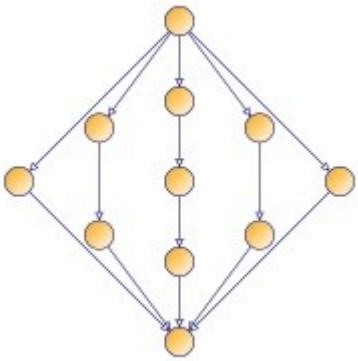
6.2.4.1 Purpose

This algorithm 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.

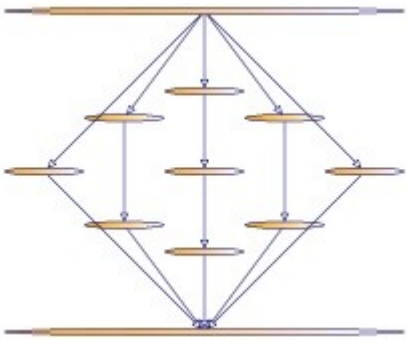
We use an algorithm (described in the book "Drawing Graphs" Michael Kaufmann - Dorothea Wagner) that allows drawing series-parallel digraphs with as much symmetry as possible.



- SP layout: DrawingStyle = BusOrthogonalDrawing



- SP layout: DrawingStyle = StraightLine -



- SP layout: DrawingStyle = VisibilityDrawing -

6.2.4.2 Code example

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

```
LayoutFlow.SeriesParallelLayout(this.addflow,
       DrawingStyle.BusOrthogonalDrawing,
       Orientation.North,
         80,                // Sets the distance between adjacent levels
         80,                // Sets the distance between adjacent nodes
         new Size(30, 30),  // The vertex size
         new Size(20, 20)); // Margins
```

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

6.2.4.3 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).

6.2.4.4 Side Effect

After the layout execution:

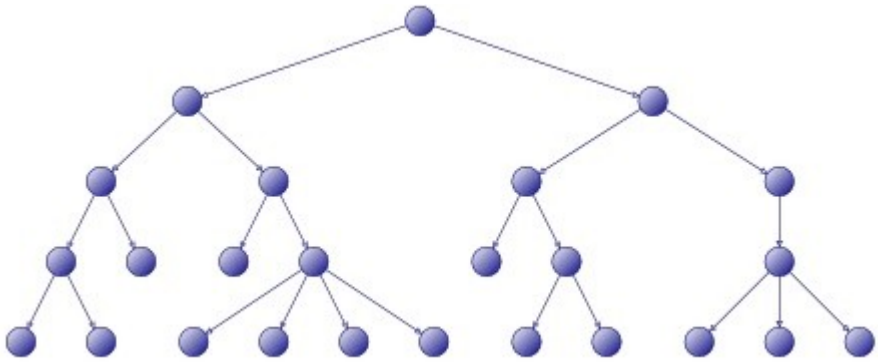
- the line style of the links is Polyline. Moreover, if the DrawingStyle property is not BusOrthogonalDrawing, then each link is composed of only one segment.
- the InConnectionMode property of the destination node of a link is set to Center.
- the OutConnectionMode property of the origin node of a link is set to Center.

6.2.5 Tree layout

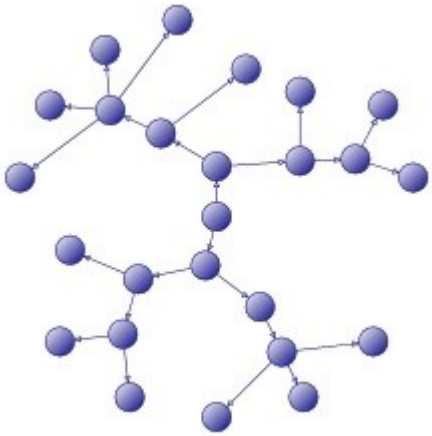
6.2.5.1 Purpose

This algorithm 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. It 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.



- Tree layout: DrawingStyle = Layered -



- Tree layout: DrawingStyle = Radial -

6.2.5.2 Code example

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

```
LayoutFlow.TreeLayout(this.addflow,
                          50, // Layer distance
                          50, // Vertex distance
                          DrawingStyle.Layered,
                          Orientation.North,
                          new Size(20, 20)); // Margin
```

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

6.2.5.3 Limitation

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

6.2.5.4 Side Effect

After the layout execution:

- the line style of the links is Polyline
- the InConnectionMode property of the destination node of a link is set to Center.
- the OutConnectionMode property of the origin node of a link is set to Center.

6.3 Data Customization

6.3.1 Framework's Tag property

Nodes and links provide a **Tag** which can be used to reference any object.

6.3.2 Attached properties

Attached properties is an extensibilty mechanism that can be used for storing any data with each node or each link.

This feature is used in the AlgoFlow component whose source code is installed with AddFlow.

For instance, when you write a depth first search algorithm to visit all the nodes of a graph, you need a boolean variable to mark a node as "visited". You can use an attached property for that. You can declare it as follows:

```
static readonly DependencyProperty IsVisitedProperty =
      DependencyProperty.RegisterAttached("IsVisited", typeof(bool),
              typeof(Connect), new FrameworkPropertyMetadata(false));

public static bool GetIsVisited(DependencyObject item)
 {
      if (item == null)
          return false;
      return (bool)item.GetValue(IsVisitedProperty);
}

public static void SetIsVisited(DependencyObject item, bool value)
{
       if (item != null)
              item.SetValue(IsVisitedProperty, value);
}
```

Then you can use it to mark a node as visited:

```
Connect.SetIsVisited(node, true);
```

6.3.3 Derivation of Node and Link classes

Using inheritance, you can create a new class by adding to or otherwise modifying an existing class. You can do that with the Node and Link classes and add custom data to each class.

6.4 Conversion guide from previous versions

6.4.1 Introduction

As you will see, some features are removed, some features are working differently and some features are renamed.

6.4.1.1 Removed Features

- The link jumps (intersection display) feature will be implemented in the next version (with a faster algorithm).
- Routing (should be added in a future version)
- In Place edition (should be added in a future version)
- Tooltips

6.4.1.2 Features that work differently

- Node shapes and link arrows are implemented using geometries.
- The Nodes and Items collections disappear because we prefer to use the LINQ technology, as explained at the beginning of the paragraph 4.
- The ReadXml and WriteXml methods (used in the .NET version) disappear because AddFlow does not offer any serialization feature: you have to use your own method to serialize a diagram (the good news is that this is demonstrated in the PageSerialize.xaml.cs file)

6.4.1.3 Renamed features

The name of many boolean properties (except the properties that start with the «Can» prefix like CanMoveNode) starts with the «Is» prefix. For instance, the Selected property is renamed IsSelected. This is a trend in the Silverlight/WPF world.

Some events have been also renamed. For instance, AfterAddLink is renamed LinkCreated.

We have done so because it is the wording used in the Silverlight/WPF world.

6.4.2 Conversion guide from AddFlow ActiveX

6.4.2.1 AddFlow properties

Remark: many properties of the ActiveX control allows defining default behaviors or styles for nodes and links. For instance, the FillColor allows defining the default filling color for nodes. In the Silverlight version, you can define default properties for nodes and links using the NodeStyle and LinkStyle properties (of type Style).

AddFlow ActiveX	**AddFlow for Silverlight**
AdjustOrg	addflow.NodeStyle
AdjustDst	addflow.NodeStyle
Alignment	addflow.NodeStyle
AllowArrowKeys	None
ArrowDst	addflow.LinkStyle
ArrowMid	None

ArrowOrg	addflow.LinkStyle
Autorouting	None
AutoSize	None
AutoScroll	addflow.CanDragScroll
BackColor	addflow.Background
BackMode	None
BackPicture	None
BorderStyle	None
CanChangeDst	addflow.CanChangeDst
CanChangeOrg	addflow.CanChangeOrg
CanDrawNode	addflow.CanDrawNode
CanDrawLink	addflow.CanDrawLink
CanFireError	None
CanMoveNode	addflow.CanMoveNode
CanMultiLink	addflow.CanMultiLink
CanReflexLink	addflow.CanReflexLink
CanSizeNode	addflow.CanSizeNode
CanStretchLink	addflow.CanStretchLink
CanUndoRedo	addflow.CanUndoRedo
CustomShapeIndex	None
CustomShapes	None (but see PageShapes;xaml)
DisplayHandles	None (but see PageGenealogy.xaml)
DrawColor	addflow.NodeStyleand addflow.LinkStyle
DrawStyle	None
DrawWidth	addflow.NodeStyleand addflow.LinkStyle
EditMode	None
Ellipsis	None
FillColor	addflow.NodeStyle
Font	addflow.NodeStyle and addflow.LinkStyle
TextColor	addflow.NodeStyle and addflow.LinkStyle
GridColor	None
GridStyle	None
Hidden	None
JumpSize	None
LastUserAction	None
LinkCreationMode	None
LinkingHandleSize	addflow.StretchThumStyle
LinkStyle	addflow.LinkStyle
LogicalOnly	None
MaxDegree	None
MaxInDegree	None
MaxOutDegree	None
MouseIcon	addflow.Cursor
MousePointer	addflow.Cursor
MultiSel	None
Nodes	None (use LINQ)
NoPrefix	None
OrientedText	None
OrthogonalDynamic	None
PicturePosition	addflow.NodeStyle
Pictures	None
PointedArea	None
PointedLink	None

PointedNode	None
ProportionalBars	None
ReadOnly	addflow.IsEnabled
RedoCode	addflow.TaskManager.RedoCode
RemovePointAngle	addflow.RemovePointDistance
Repaint	None
Rigid	None
RoundedCorner	None
RoundedCornerSize	None
RouteGrain	None
RouteMinDistance	None
RouteStartMethod	None
ScrollBars	None
ScrollTrack	None
ScrollWheel	None
SelectAction	addflow.MouseSelection
SelectedLink	addflow.SelectedItem
SelectedNode	addflow.SelectedItem
SelectionHandleSize	addflow.ResizeHandleStyle
SelectMode	None
SelLinks	addflow.SelectedItems
SelNodes	addflow.SelectedItems
Shadow	None
ShadowColor	None
Shape	None
ShapeOrientation	None
ShowGrid	None
ShowJump	None
ShowPropertyPages	None
ShowToolTip	None
SkipUndo	addflow.TaskManager.SkipUndo
SnapToGrid	Addflow.GridSnap
SizeArrowDst	addflow.LinkStyle
SizeArrowMid	None
SizeArrowOrg	addflow.LinkStyle
StretchingPoint	None
Tag	addflow.Tag
Transparent	None
UndoCode	addflow.TaskManager.UndoCode
UndoSize	addflow.TaskManager.UndoLimit
XExtent	addflow.Extent
XGrid	addflow.GridSize
XScroll	None (see ScrollViewer property)
XShadowOffset	None
XZoom	addflow.Zoom
YExtent	addflow.Extent
YGrid	addflow.GridSize
YScroll	None (see ScrollViewer property)
YShadowOffset	None
YZoom	addflow.Zoom

6.4.2.2 AddFlow methods

AddFlow ActiveX	AddFlow for Silverlight
BeginAction	addflow.TaskManager.BeginAction
CanPaste	None
CanRedo	addflow.TaskManager.CanRedo
CanUndo	addflow.TaskManager.CanUndo
Copy	None
DeleteSel	addflow.Delete
DeleteMarked	None
DisplayPropertyPage	None
EndAction	addflow.TaskManager.EndAction
ExportPicture	None
GetLinkAtPoint	None
GetNodeAtPoint	None
GetVersion	None
IsChanged	addflow.IsChanged
IsSelChanged	None
LoadFile	None (see paragraph about Serialization)
LoadMemory	None (see paragraph about Serialization)
Paste	None
Redo	addflow.TaskManager.Redo
Refresh	None
SaveFile	None (see paragraph about Serialization)
SaveImage	None
SaveMemory	None (see paragraph about Serialization)
SelectAll	Addflow.SelectAll
SelectRectangle	None
SetChangedFlag	addflow.ResetChangedFlag
SetSelChangedFlag	None
StartEdit	None
Undo	addflow.TaskManager.Undo
ZoomRectangle	None

6.4.2.3 AddFlow events

AddFlow ActiveX	AddFlow for Silverlight
AfterAddLink	addflow.LinkCreated
AfterAddNode	addflow.NodeCreated
AfterEdit	Addflow.AfterEdit
AfterMove	None
AfterResize	None
AfterSelect	None
AfterStretch	None
BeforeAddLink	None
BeforeAddNode	None
BeforeChangeDst	addflow.DstChanging
BeforeChangeOrg	addflow.OrgChanging
BeforeEdit	addflow.BeforeEdit
Error	Addflow.Error
Scroll	None

6.4.2.4 Node properties

AddFlow ActiveX	AddFlow for Silverlight
Alignment	node.TextHorizontalAlignment, node.TextVerticalAlignment
AutoSize	None
BackMode	None
DrawColor	None (node.Content)
DrawStyle	None (node.Content)
DrawWidth	None (node.Content)
EditMode	Node.IsInEditMode
FillColor	None (node.Content)
Font	node.FontFamily, node.FontSize, etc...
ForeColor	node.Foreground
Height	node.Size.Height
Hidden	Node.Visibility
Index	None
InLinks	None
Key	None
Left	node.Location.X
Links	node.Links
Logical	None
Marked	None
MaxDegree	None
MaxInDegree	None
MaxOutDegree	None
Moveable	None
OutLinks	None
Picture	Node.ImageUri
PictureIndex	None
PicturePosition	node.ImageHorizontalAlignment, node.ImageVerticalAlignment
Selectable	node.IsSelectable
Selected	node.IsSelected
Shadow	None
Shape	None (node.Content)
ShapeOrientation	None
Sizable	None
Tag	Node.Tag
TagVariant	None
Text	node.Text
Tooltip	None
Top	node.Location.Y
Transparent	None
UserData	None
Width	node.Size.Width
xMoveable	node.IsXMoveable
xTextMargin	None (node.Content)
xScrollable	None
xSizeable	node.IsXSizeable
yMoveable	node.IsYMoveable
yTextMargin	None (node.Content)
yScrollable	None
ySizeable	node.IsYSizeable
ZOrder	Use Addflow methods SendToBack, BringToFront, SendBackward,

	BringForward
ZOrderIndex	Use Addflow methods SendToBack, BringToFront, SendBackward, BringForward

6.4.2.5 Node methods

AddFlow ActiveX	**AddFlow for Silverlight**
Clone	None
EnsureVisible	None
GetLinkedNode	None
PropertyPage	None

6.4.2.6 Link properties

AddFlow ActiveX	**AddFlow for Silverlight**
AdjustDst	link.Dst.InConnectionMode
AdjustOrg	link.Org.OutConnectionMode
ArrowDst	link.ArrowDstGeometry, link.ArrowDstAngle, link.ArrowDstFill, etc...
ArrowMid	None
ArrowOrg	link.ArrowOrgGeometry, link.ArrowOrgAngle, link.ArrowOrgFill, etc...
BackMode	None
DrawColor	link.Stroke
DrawStyle	link.DashStyle
DrawWidth	link.StrokeThickness
ExtraPoints	link.Points
Dst	link.Dst
Font	link.FontFamily, link.FontSize
ForeColor	link.Foreground
Hidden	Link.Visibilty
InIndex	None
Key	None
LinkStyle	link.LineStyle
Logical	None
Marked	None
Org	link.Org
OrientedText	None
OrthogonalDynamic	None
OutIndex	None
Rigid	None
RoundedCorner	link.RoundedCornerSize
Selectable	link.IsSelectable
Selected	link.IsSelected
ShowJump	None
SizeArrowDst	link.ArrowOrgWidth, link.ArrowOrgHeight
SizeArrowMid	None
SizeArrowOrg	link.ArrowGeometryOrg
Stretchable	link.IsStretchable
Tag	Link.Tag
TagVariant	None (see previous paragraph about Data Customization)
Text	link.Text
TextSegment	None
Tooltip	None
UserData	None

ZOrder	Use Addflow methods SendToBack, BringToFront, SendBackward, BringForward
ZOrderIndex	Use Addflow methods SendToBack, BringToFront, SendBackward, BringForward

6.4.2.7 Link methods

AddFlow ActiveX	**AddFlow for Silverlight**
Clone	None
EnsureVisible	None
PropertyPage	None
Reverse	None

6.4.3 Conversion guide from AddFlow for .NET

6.4.3.1 AddFlow properties

AddFlow for .NET	AddFlow for Silverlight
BackColor	addflow.Background
BorderStyle	None
CanChangeDst	addflow.CanChangeDst
CanChangeOrg	addflow.CanChangeOrg
CanDrawNode	addflow.CanDrawNode
CanDragScroll	addflow.CanDragScroll
CanDrawLink	addflow.CanDrawLink
CanFireError	None
CanMoveNode	addflow.CanMoveNode
CanMultiLink	addflow.CanMultiLink
CanReflexLink	addflow.CanReflexLink
CanSizeNode	addflow.CanSizeNode
CanStretchLink	addflow.CanStretchLink
CanUndoRedo	addflow.CanUndoRedo
DisplayHandles	None
DefLinkProp	addflow.LinkStyle
DefNodeprop	addflow.NodeStyle
Extent	addflow.Extent
Grid.Color	addflow.GridColor
Grid.Draw	addflow.GridDraw
Grid.Style	None
Grid.Size	addflow.GridSize
Grid.Snap	addflow.GridSnap
Items	None (use LINQ)
JumpSize	addflow.CanShowJump and link.JumpSize
LinkCreationMode	None
LinkingHandleSize	addflow.StretchHandleStyle
MouseAction	addflow.MouseSelection
MultiSel	None
Nodes	None (use LINQ)
Images	None
PointedArea	None
PointedItem	None
RedoCode	addflow.TaskManager.RedoCode
RemovePointAngle	addflow.RemovePointDistance
SelectedItem	None
ScrollPosition	None (see ScrollViewer property)
SelectionHandleSize	addflow.ResizeHandleStyle
SelectedItems	addflow.SelectedItems
ShowToolTips	None
SkipUndo	addflow.TaskManager.SkipUndo
StretchingPoint	None
UndoCode	addflow.TaskManager.UndoCode
UndoSize	addflow.TaskManager.UndoLimit
Zoom	addflow.Zoom

6.4.3.2 AddFlow methods

AddFlow for .NET	AddFlow for Silverlight
AddLink	addflow.AddLink
AddToLastUserAction	addflow.TaskManager.AddToLastUserAction
BeginAction	addflow.TaskManager.BeginAction
BeginUpdate	None
CanRedo	addflow.TaskManager.CanRedo
CanUndo	addflow.TaskManager.CanUndo
CreateLink	addflow.AddLink
DeleteSel	addflow.Delete
EndAction	addflow.TaskManager.EndAction
EndUpdate	None
ExportMetafile	None
GetDiagramSize	None
GetItemAt	None
GetItemsInRectangle	None
IsChanged	None
IsSelChanged	None
Redo	addflow.TaskManager.Redo
ReadXML	None (see paragraph about Serialization)
Render	None
ResetUndoRedo	addflow.TaskManager.Clear
SelectAll	addflow.SelectAll
SelectRectangle	None
SetChangedFlag	addflow.ResetChangedFlag
SetSelChangedFlag	None
StartEdit	Node.BeginEdit
SubmitTask	addflow.TaskManager.SubmitTask
Undo	addflow.TaskManager.Undo
WriteXML	None (see paragraph about Serialization)
ZoomPoint	None
ZoomRectangle	None

6.4.3.3 AddFlow events

AddFlow for .NET	AddFlow for Silverlight
AfterAddLink	addflow.LinkCreated
AfterAddNode	addflow.NodeCreated
AfterChangeDst	addflow.DstChanged
AfterChangeOrg	addflow.OrgChanged
AfterEdit	Addflow.AfterEdit
AfterMove	None
AfterRemoveLink	addflow.NodeDeleted
AfterRemoveNode	addflow.LinkDeleted
AfterResize	None
AfterSelect	None
AfterStretch	None
BeforeAddLink	None
BeforeAddNode	None
BeforeChangeDst	addflow.DstChanging

BeforeChangeOrg	addflow.OrgChanging
BeforeEdit	Addflow.BeforeEdit
BeforeReadXMLLink	None (see paragraph about Serialization)
BeforeReadXMLNode	None (see paragraph about Serialization)
BeforeRemoveLink	None
BeforeRemoveNode	None
BeforeWriteXMLLink	None (see paragraph about Serialization)
BeforeWriteXMLNode	None (see paragraph about Serialization)
CancelEdit	Addflow.CancelEdit
DiagramOwnerDraw	None
Error	addflow.Error
ExtentChange	None
LinkOwnerDraw	None
NodeOwnerDraw	None
SelectionChange	addflow.SelectionChanged
ReadXMLLinkExtraData	None (see paragraph about Serialization)
ReadXMLNodeExtraData	None (see paragraph about Serialization)
WriteXMLLinkExtraData	None (see paragraph about Serialization)
WriteXMLNodeExtraData	None (see paragraph about Serialization)

6.4.3.4 Node properties

AddFlow for .NET	**AddFlow for Silverlight**
Alignment	node.TextHorizontalAlignment, node.TextVerticalAlignment
AutoSize	None
BackMode	None
DrawColor	None (node.Content)
DrawStyle	None (node.Content)
DrawWidth	None (node.Content)
EditMode	Node.IsInEditMode
FillColor	None (node.Content)
Font	node.FontFamily, node.FontSize, etc...
ForeColor	node.Foreground
Height	node.Size.Height
Hidden	Node.Visibility
Index	None
InLinks	None
Left	node.Location.X
Links	node.Links
Logical	None
OutLinks	None
ImageIndex	node.ImageUri
ImageLocation	None
ImagePosition	node.ImageHorizontalAlignment, node.ImageVerticalAlignment
Selectable	node.IsSelectable
Selected	node.IsSelected
Shadow	None
Shape	None (node.Content)
Tag	Node.Tag
Text	node.Text
TextMargin	None

Tooltip	None
Top	node.Location.Y
Transparent	None
Width	node.Size.Width
xMoveable	node.IsXMoveable
xSizeable	node.IsXSizeable
yMoveable	node.IsYMoveable
ySizeable	node.IsYSizeable
ZOrder	Use Addflow methods SendToBack, BringToFront, SendBackward, BringForward

6.4.3.5 Node methods

AddFlow for .NET	**AddFlow for Silverlight**
Clone	None
BringIntoView	None
GetLinkedNode	None
Remove	None

6.4.3.6 Link properties

AddFlow for .NET	**AddFlow for Silverlight**
AdjustDst	link.Dst.InConnectionMode
AdjustOrg	link.Org.OutConnectionMode
ArrowDst	link.ArrowDstGeometry, link.ArrowDstAngle, link.ArrowDstFill, etc...
ArrowMid	None
ArrowOrg	link.ArrowOrgGeometry, link.ArrowOrgAngle, link.ArrowOrgFill, etc...
BackMode	None
DrawColor	link.Stroke
DrawStyle	None
DrawWidth	link.StrokeThickness
Points	link.Points
Dst	link.Dst
Font	link.FontFamily, link.FontSize etc...
TextColor	link.Foreground
Hidden	Link.Visibility
Line.Style	link.LineStyle
Line.OrthogonalDynamic	None
Line.RoundedCorner	link.RoundedCornerSize
Logical	None
Org	link.Org
OrientedText	None
Rigid	None
Selectable	link.IsSelectable
Selected	link.IsSelected
Jump	link.JumpSize
Stretchable	link.IsStretchable
Tag	Tag
Text	link.Text
TextSegment	None
Tooltip	None
ZOrder	Use Addflow methods SendToBack, BringToFront, SendBackward, BringForward

6.4.3.7 Link methods

AddFlow for .NET	AddFlow for Silverlight
Clone	None
EnsureVisible	None
Reverse	None