AddFlow for HTML5 v1.2
Tutorial

January 2014

Lassalle Technologies

http://www.lassalle.com

CONTENTS

1) Introduction

AddFlow for HTML5 is a general purpose Flowcharting/Diagramming HTML5 component, which lets you quickly build flowchart-enabled HTML5 applications.

AddFlow for HTML5 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 component 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, …

Notice that there are two editions of AddFlow for HTML5 :

- AddFlow for HTML5 **Standard** Edition.
- AddFlow for HTML5 **Professional** Edition which includes also a set of graph layout algorithms.

Purpose of this tutorial

This tutorial provides information on:

- licensing
- creating diagrams programmatically, using the AddFlow for HTML5 API
- creating diagrams interactively

Who should use this tutorial?

This guide is intended for application programmers building HTML5 applications.

Samples

AddFlow for HTML5 is installed with one demo sample. Its source code (HTML, Javascript) is provided.

2) Last Version enhancements

2.1 Version 1.2

2.1.1 Graph Layout Algorithms

We provide now either a **Standard Edition**, either a **Professional Edition** of AddFlow for HTML5.

The Professional Edition offers also a set of graph layout algorithms (hierarchic, force-directed, orthogonal, series-parallel, tree, radial).

See paragraph #8.Automatic Graph Layout for more informations.

2.1.2 xCustomOffset and yCustomOffset properties

xCustomOffset and **yCustomOffset** are addflow properties that allow adding horizontal and vertical offsets to the mouse coordinates. This may be useful in some development environments.

2.1.3 isDstPointAdjustable and isOrgPointAdjustable properties

isDstPointAdjustable and **isOrgPointAdjustable** are link properties that determine whether the last or the first link point can be changed.

2.2 Version 1.1

2.2.1 Better speed performance

This new release provides better speed performance by using a quadtree structure. By default, then the quadtree structure is used. Otherwise, brute force is used. You may use the **useQuadtree** method to determine if the quadtree structure is used.

See the paragraph about Performance tuning for more explanations.

2.2.2 isFixedSize property

By default, when adding new items in a diagram, the addflow canvas size is adjusted to be the sum of the size of the diagram and the size of the viewport (the size of the div element containing the addflow canvas).

However, now, if the **isFixedSize** property is true, then the size of the addflow canvas does not change, whatever the size of the diagram may be.

There is an example in the "Network" example of the demo (**network.htm)**.

2.2.3 selectionChange event

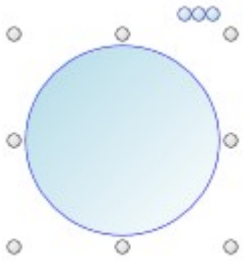
We could live without this event but it is better to have it. The **selectionChange** event is fired each time an item is selected or unselected. However it is possible to prevent this event to be fired if the addflow **canSendSelectionChangedEvent** property is false.

There is an example of the use of this event in the "Custom undo" example of the demo (**customundo.htm)**.

2.2.4 Context handle

If the **isContextHandle** property of an item (node or link) is true, then, if you select it, you will see a context handle ooo and clicking on this handle causes the firing of a **context** event allowing you for instance to display a context menu or a dialog box.

On the following image, you can see the context handle "ooo" at the top right of the node.



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" (**statediagram.htm**) and in the "Social network" example(**socialnetwork.htm)** of the demo. In the second example, only nodes have a context handle.

3) Licensing

If you do not own a commercial license, this file shall be governed by the license agreement that can be found at: http://www.lassalle.com/html5/license_evaluation.pdf

If you own a commercial license, this file shall be governed by the license agreement that can be found at: http://www.lassalle.com/html5/license_commercial.pdf

The key points are the following:

- it is royalty free
- one license per individual developer
- the evaluation version can be used only for evaluation and testing purpose.
- if you purchase a commercial license, you get the source code, support and the right to use AddFlow for business purposes.
- you may purchase either the Standard Edition of AddFlow, either the Professional Edition of AddFlow. The professional version provides also a set of graph layout algorithms.

The following FAQ gives more details.

1. What is the difference between the evaluation license and the commercial license ?

With the evaluation license, no support is provided and the javascript source code is obfuscated (minified): spaces and comments are removed and variables and function names are replaced by meaningless names.

With the commercial license, you are entitled to obtain free support and you get also the full javascript source code and have the right to modify it.

However, you have not the right to divulge, publish or distribute the source code of AddFlow or of a modified version of AddFlow. You must previously obfuscate the source code before distributing or publishing it.

2. What is the difference between the Standard Edition and the Professional Edition

The Professional Edition includes also a set of graph layout algorithms (see paragraph #8.Automatic Graph Layout)

3. If purchase AddFlow, shall I have to purchase also an obfuscator ?

You may purchase such a software but you can also use a free obfuscator like Google's Closure Compiler. It is very easy to use. A simple command like the following is working:

```
java -jar compiler.jar --js="addflow.js" --js_output_file="addflow-min.js"
```

(this supposes that the file addflow.js is placed in the same directory as the Closure program)

Then, you have to add the header comment you can find at the beginning of the file addflow-min.js file of the evaluation version.

```
/* AddFlow for HTML5
   v1.2.0.2 - January 2014
   Copyright (c) 2012-2014 Lassalle Technologies. All rights reserved.
   http://www.lassalle.com
```

```
  Author: Patrick Lassalle mailto:plassalle@lassalle.com
  If you do not own a commercial license, this file shall be governed by the license
  agreement that can be found at: http://www.lassalle.com/html5/license_evaluation.pdf
  If you own a commercial license, this file shall be governed by the license
  agreement that can be found at: http://www.lassalle.com/html5/license_commercial.pdf
*/
```

4. Can I just minify the source code ?

Only if variables and function names are replaced by meaningless names. Just removing spaces and comments is not enough.

5. Do you provide an Open Source license ?

While we acknowledge Open Source, we currently do not license AddFlow as an Open Source software.

6. Is AddFlow runtime-royalty free ?

Yes.

7. How many developers can use AddFlow for HTML5 ?

AddFlow is licensed per individual developer. Each developer using AddFlow needs to purchase a license.

8. Do you offer multi-pack discounts ?

Yes, we offer the following type of licenses:

- Single developer license: allows just one developer

- Team license: allows 4 developers

- Site license: allows unlimited developers at a single physical address.

- Enterprise license: allows all developers of an enterprise

9. How many projects can I create with a license of AddFlow for HTML5 ?

You can use your license of AddFlow for HTML5 on as many projects as you like because AddFlow for HTML5 can be distributed on a royalty-free basis.

10. Will purchasing guarantee me upgrades ?

It does not include major version upgrades. However, we will provide bug fixes and minor enhancements free of charge.

11. Do you provide refunds ?

Under no circumstances shall a refund be applied after the source code is sent to the client.

4) Getting Started

1) Add the AddFlow script to your html page, preferably at <head> tag:

```
<script type="text/javascript"
src="http://www.lassalle.com/html5/demo/src/addflow.js"></script>
```

2) Add the following html5 code to include a div element containing the canvas to display the diagram.

```
<div id="div1" style="border-style: solid; width: 900px; height: 400px; overflow:
auto;">
    <canvas id="canvas1" width="900" height="400">
        *** THIS BROWSER DOES NOT SUPPORT THE CANVAS ELEMENT ***
    </canvas>
</div>
```

Then you are ready to use AddFlow interactively or programmatically. Let us first see how to use it interactively.

remark: you are not forced to place the canvas inside a div element. You may find an example in the "Network" example of the demo (**network.htm)**. However you have to do that to obtain a scrolling feature.

5) Interactive creation of a diagram

5.1 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

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.

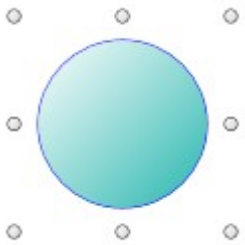
And a set of methods and properties allow implementing a powerful Undo/Redo feature.

5.2 Create a diagram interactively

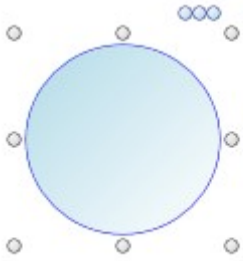
5.2.1 Draw a node

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 circles) are displayed.

The 8 handles allow **resizing the node**. 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.



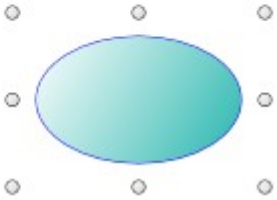
remark: here we suppose that the isContextHandle property of nodes is false (which is the case by default). Otherwise, you would see also a context handle “ooo” as in the following image:



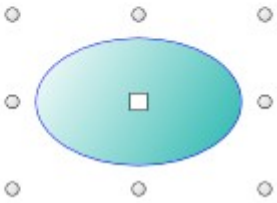
TIP: Notice that, when adding new items in a diagram, the addflow canvas size is adjusted to be the sum of the size of the diagram and the size of the viewport (the size of the div element containing the addflow canvas). However, you may alter this behavior with the **isFixedSize** property. It it is true, then the size of the addflow canvas does not change, whatever the size of the diagram may be.

5.2.2 Draw a link

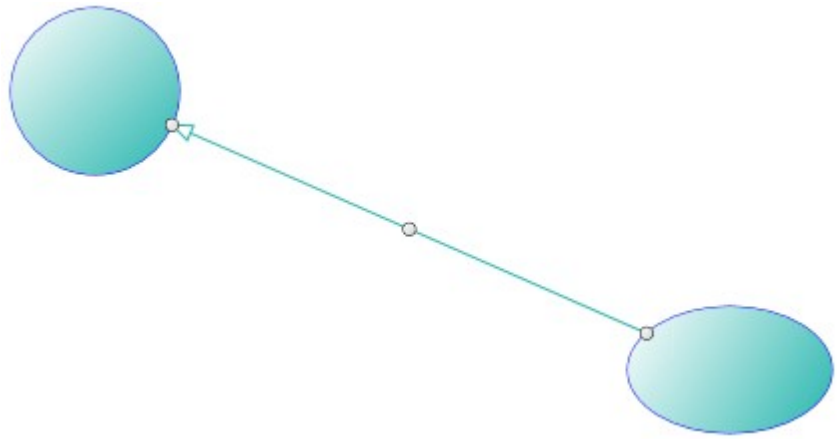
Draw a second node.



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.



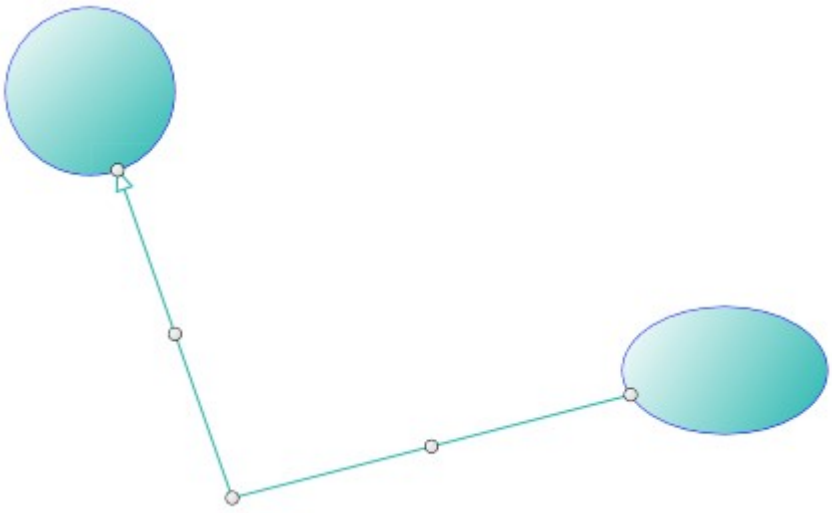
As you can see, the link stretching handles are also displayed as little rectangles. By default, those handles are small rectangles as for the nodes above. But we can change the style of those handles. The DemoFlow sample provided with AddFlow shows many distinct ways to display the node resizing handles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

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

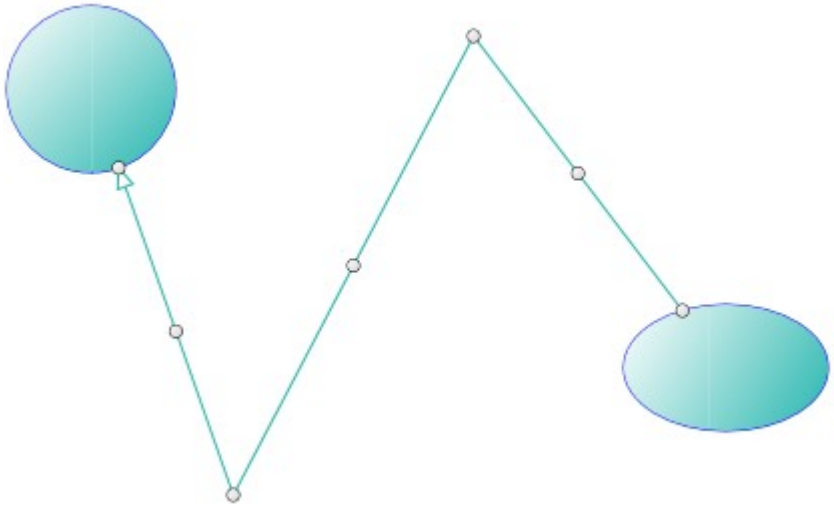
remark: here we suppose that the isContextHandle property of links is false (which is the case by default). Otherwise, you would see also a context handle “ooo”.

5.2.3 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).

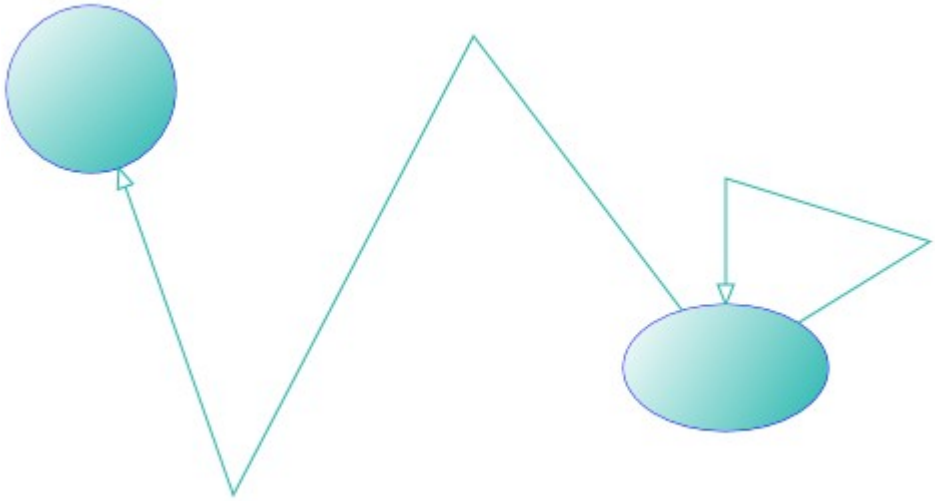


Create another segment



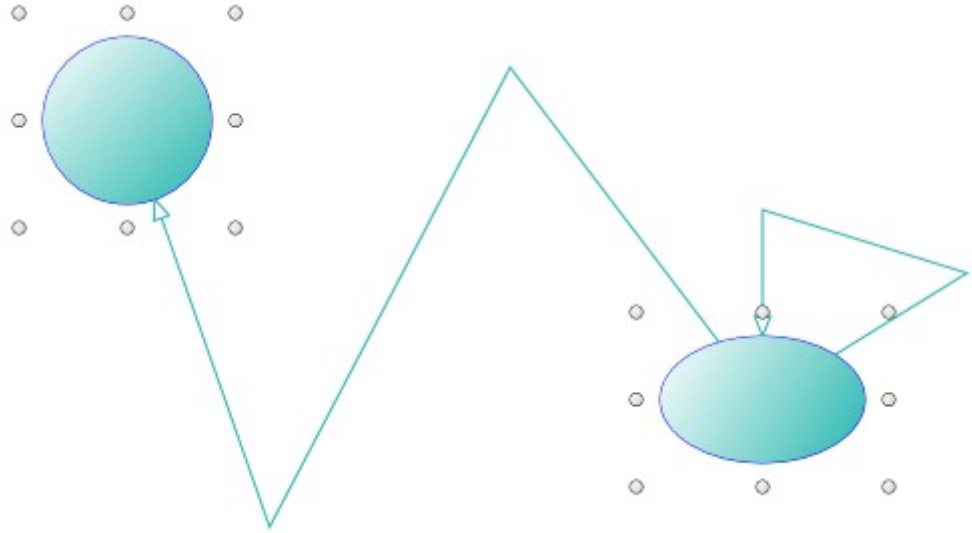
5.2.4 Draw a reflexive link

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.

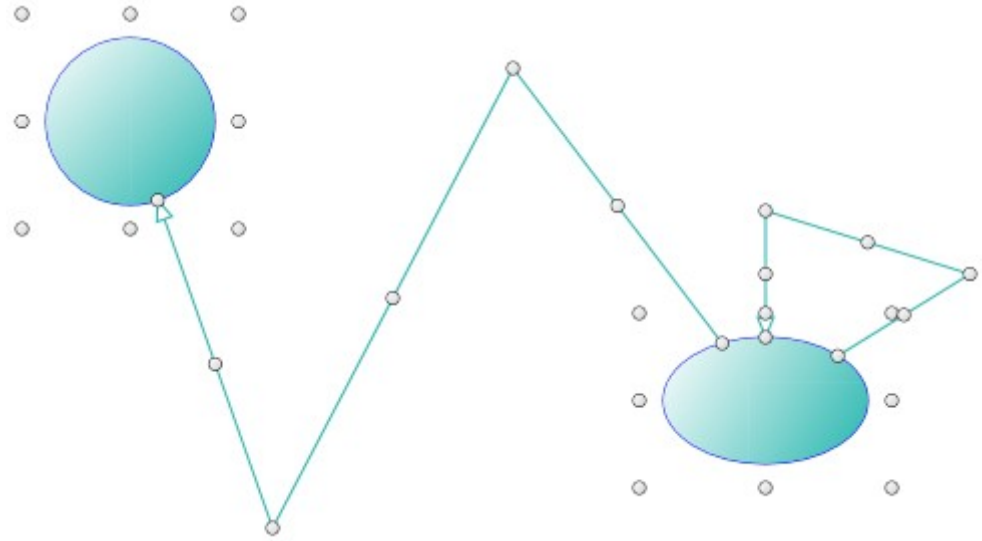


5.2.5 Multiselection

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.



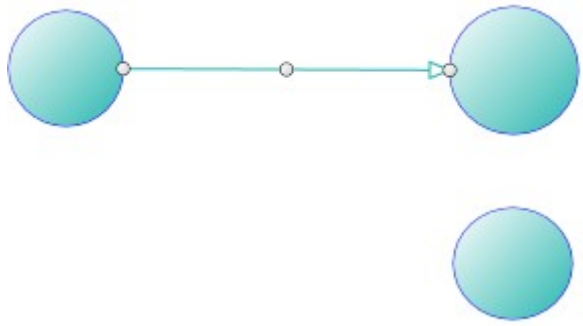
There is another way to perform multiselection, using the **MouseSelection** property and assigning it the **MouseSelection.Selection** value. Then you can select several nodes and links: you bring the mouse cursor into the AddFlow control, press the left button, move the mouse and release the left button. 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.

5.2.6 Change properties of a node or a link

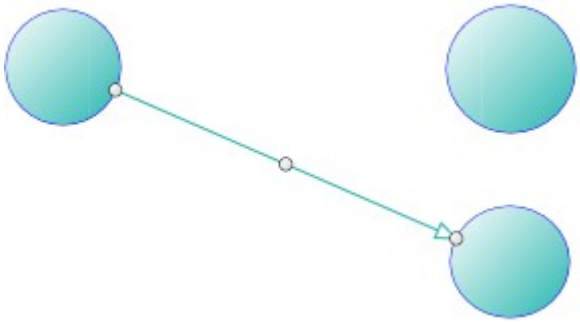
Interactively, without adding any code, you can change the position and the size of a node. You can add segments to a link or remove them. To change other properties (shape, styles, colors, behaviors, etc) of a node or a link, you have to write some code.

5.2.7 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.



5.2.8 Selecting, zooming panning.

Selection

If the **mouseSelection** property of the AddFlow control is set to '**selection**' or '**selection2**', you can select items with the mouse. 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 selecting all items partially (in the case of '**selection**') or completely (in the case of '**selection2**') inside the rectangular area.

If the **canSelectOnMouseMove** property is true, the selection of items with the mouse is made at each mouseMove event. Otherwise, it is made only when the mouseUp event is fired

Zooming

If the **mouseSelection** property of the AddFlow control is set to '**zoom**', you can zoom the diagram interactively with the mouse. 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.

The zooming is isotropic.

Panning

If the **mouseSelection** property of the AddFlow control is set to '**pan**', you can pan the diagram with the mouse. You click on the diagram at a place where there is no item and you move the mouse: the diagram is panned. This will work only if the AddFlow control is inside a Div element. The AddFlow control is scrolled inside this div element.

6) Programmatic creation of a diagram

6.1 Overview

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

The main class is the **Flow** class. It uses a canvas to display the diagram.

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.

WARNING: AddFlow is using a canvas to display the diagram. So the size of the diagram will be limited by the size of a HTML5 canvas. This limitation depends on the browser. Currently, most browsers seem to limit the size to 8192.

6.2 Diagram creation

6.2.1 Our first program

1) Add the AddFlow script to your html page, preferably at <head> tag:

```
<script type="text/javascript"
src="http://www.lassalle.com/html5/demo/src/addflow.js"></script>
```

2) Add the following html5 code to include a div element containing the canvas to display the diagram.

```
<div id="div1" style="border-style: solid; width: 900px; height: 400px; overflow:
auto;">
    <canvas id="canvas1" width="900" height="400">
        *** THIS BROWSER DOES NOT SUPPORT THE CANVAS ELEMENT ***
    </canvas>
</div>
```

3) Add the code to create our first diagram.xml

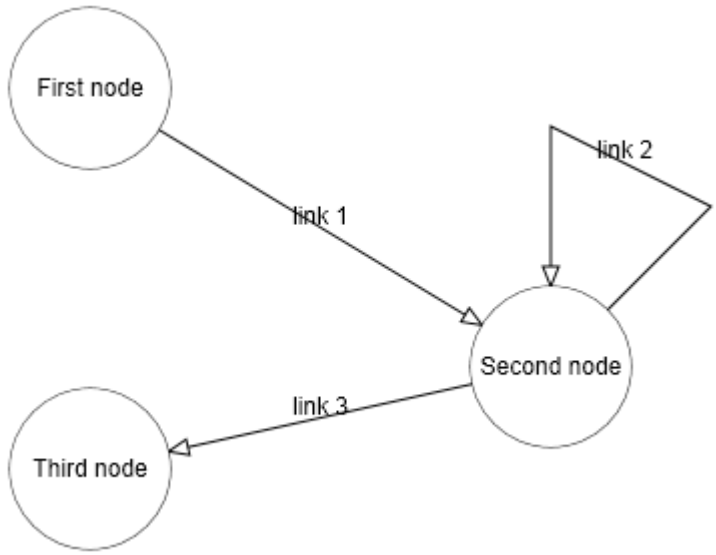
```
function createDiagram() {
      var canvas, flow, node1, node2, node3, link1, link2, link3;

      canvas = document.getElementById('canvas1');
      flow = new Lassalle.Flow(canvas);

      // Create 3 nodes
      node1 = flow.addNode(40, 40, 80, 80, "First node");
      node2 = flow.addNode(270, 179, 80, 80, "Second node");
      node3 = flow.addNode(40, 230, 80, 80, "Third node");

      // Create 3 links
      link1 = flow.addLink(node1, node2, "link 1");
      link2 = flow.addLink(node2, node2, "link 2");
      link3 = flow.addLink(node2, node3, "link 3");
```

```
        flow.refresh();
}
```

4) Call the createDiagram() function from the appropiate place, for example at body "onload" event:

```
<body onload="createDiagram()">
```

If we execute this program (**tutorial_firstprogram.htm**), 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, the last parameter is optional. For instance, to create the first node, you could have written:

```
node1 = flow.addNode(40, 40, 80, 80);
node1.Text = "First node";
```

To create a link, you have to use the **addLink** method. There is no other method. However, the third parameter is optional. For instance, to create the first link, you could have written:

```
link1 = flow.addLink(node1, node2);
link1.Text = "link 1";
```

The **addLink** method has also two other optional parameters that allow setting the index of the origin pin and the index of the destination pin.

WARNING: Note the call to the **refresh** method in the last line. This call is necessary to cause the diagram to be drawn. The only case where this call is not needed is when you encapsulate your diagram creation code by the calls to the **beginUpdate** end **endUpdate** methods.

6.2.2 Node Properties

The node properties can be grouped in several categories:

- Layout properties: **x, y, w, h. (**See the next remark)
- Appearance properties: **strokeStyle, fillStyle, gradientFillStyle, textFillStyle, lineWidth, shapeFamily, polygon, drawShape, fillShape, isShadowed**
- Node label properties: **text, textMargin, textPosition, font, textLineHeight**
- Node image properties: **image, imageMargin, imagePosition**
- Graph properties: the **getLinks** method that allows getting the array of all the links (in and out) of the node.
- Behaviour properties: **isSelected, isSelectable, isXMoveable, isYMoveable, isXSizeable, isYSizeable, isInLinkable, isOutLinkable, isContextHandle**
- Connection properties defining how a link is attached to a node: **pins**

WARNING: you should not use the x, y, w, h properties directly except for serialization purposes. To manipulate the location and size of a node, you should use instead the methods **getLeft, setLeft, getTop, setTop, getWidth, setWidth, getHeight, setHeight.**

6.2.3 Link Properties

The link properties can be grouped them in several categories:

- Appearance properties and methods: **roundedCornerSize, strokeStyle, fillStyle, textFillStyle, lineWidth, isShadowed, orthoMargin, setLineStyle (and getLineStyle)**
- Link label properties: **text, font, isOrientedText, isOpaque.**
- Arrow head properties: **arrowDst, arrowOrg.**
- 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.
- Connection properties defining how a link is attached to a node: **pinOrg**, **pinDst, isOrgPointAdjustable, isDstPointAdjustable.**
- Behaviour properties: **isSelected, isSelectable, isStretchable, isContextHandle**

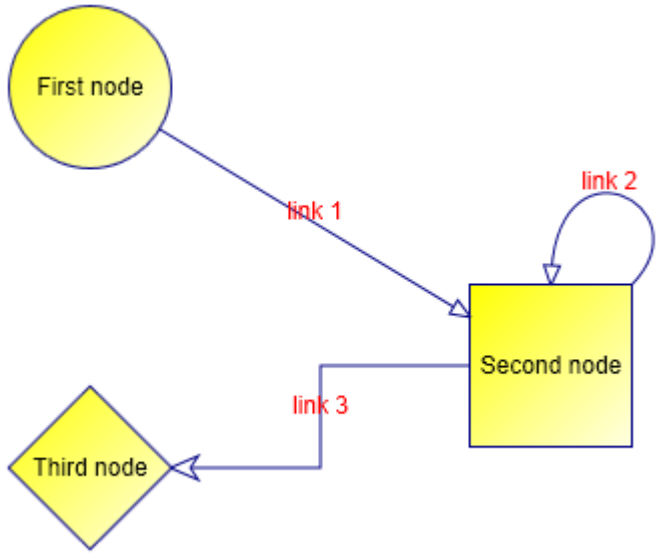
WARNING: There is also the **points** collection. A link is composed of several segments defined by a collection of points. However, you should not use this collection directly except for serialization purposes. To manipulate the collection the link points, you should use instead the methods **addPoint, removePoint, clearPoints, setPoint, getPoint** and **countPoints.**

6.2.4 Changing property values

Now let us replace the createDiagram method by the following new one:

```
function createDiagram() {
      var canvas, flow, node1, node2, node3, link1, link2, link3;

      canvas = document.getElementById('canvas1');
      flow = new Lassalle.Flow(canvas);

      // Create 3 nodes
      node1 = flow.addNode(40, 40, 80, 80, "First node");
      node1.fillStyle = 'yellow';
      node1.gradientFillStyle = 'lightyellow';
      node1.strokeStyle = 'navy';
      node1.lineWidth = 2;

      node2 = flow.addNode(270, 179, 80, 80, "Second node");
      node2.fillStyle = 'yellow';
      node2.gradientFillStyle = 'lightyellow';
      node2.strokeStyle = 'navy';
      node2.lineWidth = 2;
      node2.shapeFamily = "rectangle";

      node3 = flow.addNode(40, 230, 80, 80, "Third node");
      node3.fillStyle = 'yellow';
      node3.gradientFillStyle = 'lightyellow';
      node3.strokeStyle = 'navy';
      node3.lineWidth = 2;
      node3.shapeFamily = "polygon";
      node3.polygon = [[0, 50], [50, 0], [100, 50], [50, 100]];

      // Create 3 links
      link1 = flow.addLink(node1, node2, "link 1");
      link1.strokeStyle = 'navy';
      link1.textFillStyle = 'red';

      link2 = flow.addLink(node2, node2, "link 2");
      link2.strokeStyle = 'navy';
      link2.textFillStyle = 'red';
      link2.setLineStyle('bezier');

      link3 = flow.addLink(node2, node3, 'link 3');
      link3.strokeStyle = 'navy';
      link3.textFillStyle = 'red';
      link3.setLineStyle('orthogonal');
      link3.arrowDst = [[0, 0], [-14, -6], [-10, 0], [-14, 6]];

      flow.refresh();
}
```

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



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.

6.2.5 Default property values

Now let us replace the createDiagram method by the following new one:

```
function createDiagram() {
      var canvas, flow, node1, node2, node3, link1, link2, link3;

      canvas = document.getElementById('canvas1');
      flow = new Lassalle.Flow(canvas);

      flow.nodeModel.strokeStyle = 'navy';
      flow.nodeModel.fillStyle = 'yellow';
      flow.nodeModel.gradientFillStyle = 'lightyellow';
      flow.nodeModel.lineWidth = 2;

      flow.linkModel.strokeStyle = 'navy';
      flow.linkModel.textFillStyle = 'red';

      // Create 3 nodes
      node1 = flow.addNode(40, 40, 80, 80, "First node");

      node2 = flow.addNode(270, 179, 80, 80, "Second node");
      node2.shapeFamily = "rectangle";

      node3 = flow.addNode(40, 230, 80, 80, "Third node");
      node3.shapeFamily = "polygon";
      node3.polygon = [[0, 50], [50, 0], [100, 50], [50, 100]];

      // Create 3 links
      link1 = flow.addLink(node1, node2, "link 1");

      link2 = flow.addLink(node2, node2, "link 2");
```

```
        link2.setLineStyle('bezier');

        link3 = flow.addLink(node2, node3, 'link 3');
        link3.setLineStyle('orthogonal');
        link3.arrowDst = [[0, 0], [-14, -6], [-10, 0], [-14, 6]];

        flow.refresh();
}
```

If we execute this new program (**tutorial_defaultproperties.htm**), it will create the same diagram. However, our program is smaller because we have used the **nodeModel** and the **linkModel** properties of AddFlow. The type of nodeModel is Node whereas the type of linkModel is Link. However these objects are not part of AddFlow diagram. Both properties just allow specifying default property values for nodes and links.

For instance, writing:

```
flow.nodeModel.fillStyle = 'yellow';
```

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

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

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

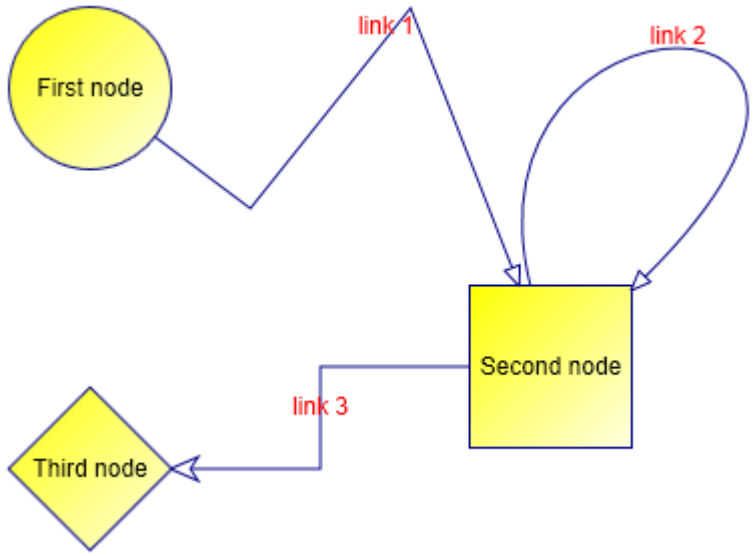
6.2.6 Stretching the links

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

```
function createDiagram() {
      var canvas, flow, node1, node2, node3, link1, link2, link3;

      canvas = document.getElementById('canvas1');
      flow = new Lassalle.Flow(canvas);

      flow.nodeModel.strokeStyle = 'navy';
      flow.nodeModel.fillStyle = 'yellow';
      flow.nodeModel.gradientFillStyle = 'lightyellow';
      flow.nodeModel.lineWidth = 2;

      flow.linkModel.strokeStyle = 'navy';
      flow.linkModel.textFillStyle = 'red';

      // Create 3 nodes
      node1 = flow.addNode(40, 40, 80, 80, "First node");

      node2 = flow.addNode(270, 179, 80, 80, "Second node");
      node2.shapeFamily = "rectangle";

      node3 = flow.addNode(40, 230, 80, 80, "Third node");
      node3.shapeFamily = "polygon";
      node3.polygon = [[0, 50], [50, 0], [100, 50], [50, 100]];

      // Create 3 links
      // The second link has a bezier lineStyle, the color of its text is red
      // The third link has a orthogonal lineStyle.
      link1 = flow.addLink(node1, node2, "link 1");
```

```
    // Add 2 points (therefore 2 segments) to this first link
    link1.addPoint(160, 140);
    link1.addPoint(240, 40);

    link2 = flow.addLink(node2, node2, "link 2");
    link2.setLineStyle('bezier');

    // Stretch this reflexive link
    link2.setPoint(260, 20, 1);
    link2.setPoint(520, 20, 2);

    link3 = flow.addLink(node2, node3, 'link 3');
    link3.setLineStyle('orthogonal');
    link3.arrowDst = [[0, 0], [-14, -6], [-10, 0], [-14, 6]];

    flow.refresh();
}
```

If we execute this program (**tutorial_stretchinglinks.htm**), you will see that the 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 **points** property has always at least 2 points.
- 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 last point of the **points** collection only if the destination node has several pins.

- You can change the first point of the **points** collection only if the origin node has several pins.
- You can change each other point of the **points** collection in any case.

6.3 Nodes

A node may contain a text and an image. Its shape can be customized. You may define for each node a set of pins to connect links.

6.3.1 Colors

Four properties allow setting colors for a node:

- **strokeStyle** It is the color of the node border.
- **fillStyle** It is the node filling color.
- **gradientFillStyle** It is used in conjunction with the .fillStyle property to set a gradient color.
- **textFillStyle** It is the color of the node text.

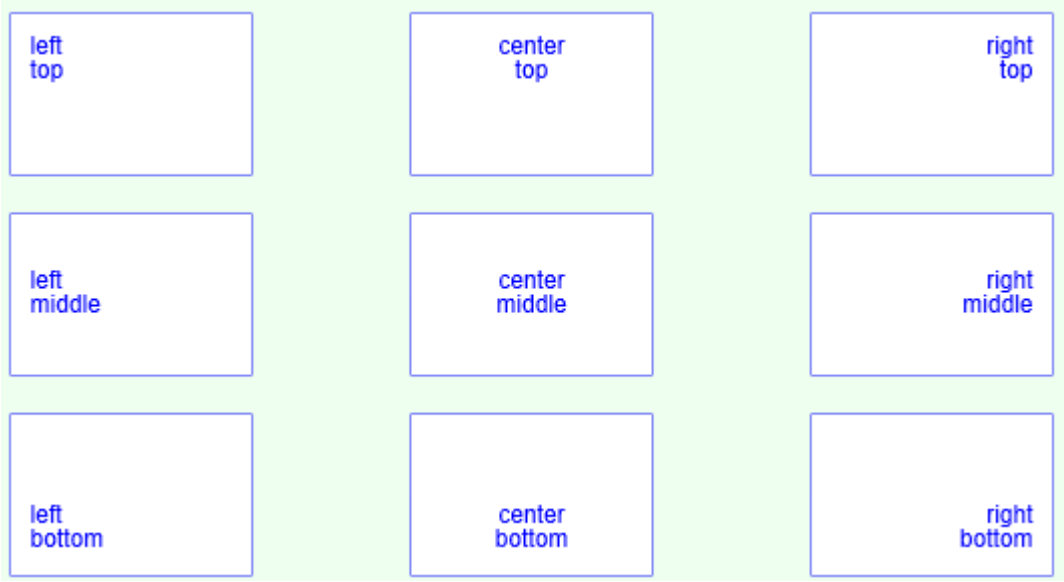
WARNING: to avoid a gradient filling color, you have to set the **gradientFillStyle** color to be the same value as the **fillStyle** color.

6.3.2 Displaying a text in a node

You can associate a text to a node with the **text** property. Two properties allow placing the text inside the node: **textPosition** and **textMargin**.

The **tutorial_nodetext.htm** page in the demo shows the effect of the textPosition property. In this example, the textMargin is set to be equal to 10 at each side:

```
flow.nodeModel.textMargin = { left: 10, top: 10, right: 10, bottom: 10 };
```



The **font** property allows changing the font used to display the node text. Example:

```
node.font = "12px Arial";
```

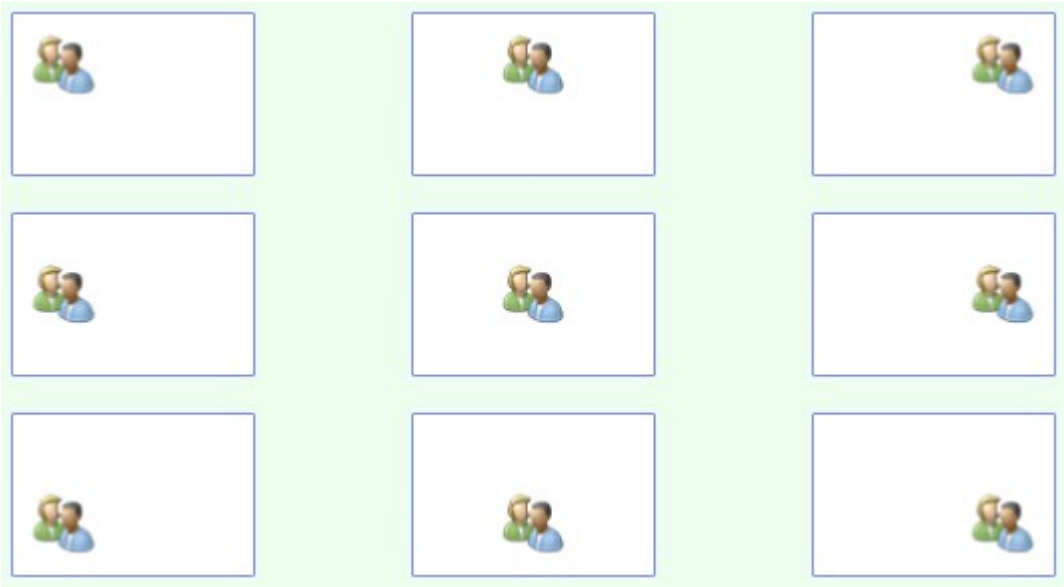
Note also the property **textLineHeight**. This property allows setting the height of a line of text. This is needed because the canvas doesn't give us a method for measuring the height of a string.

6.3.3 Displaying an image in a node

You can associate an image to a node with the **image** property. Two properties allow placing the image inside the node: **imagePosition** and **imageMargin**.

The **tutorial_nodeimage.htm** page in the demo shows the effect of the imagePosition property. In this example, the imageMargin is set to be equal to 10 at each side:

```
flow.nodeModel.imageMargin = { left: 10, top: 10, right: 10, bottom: 10 };
```

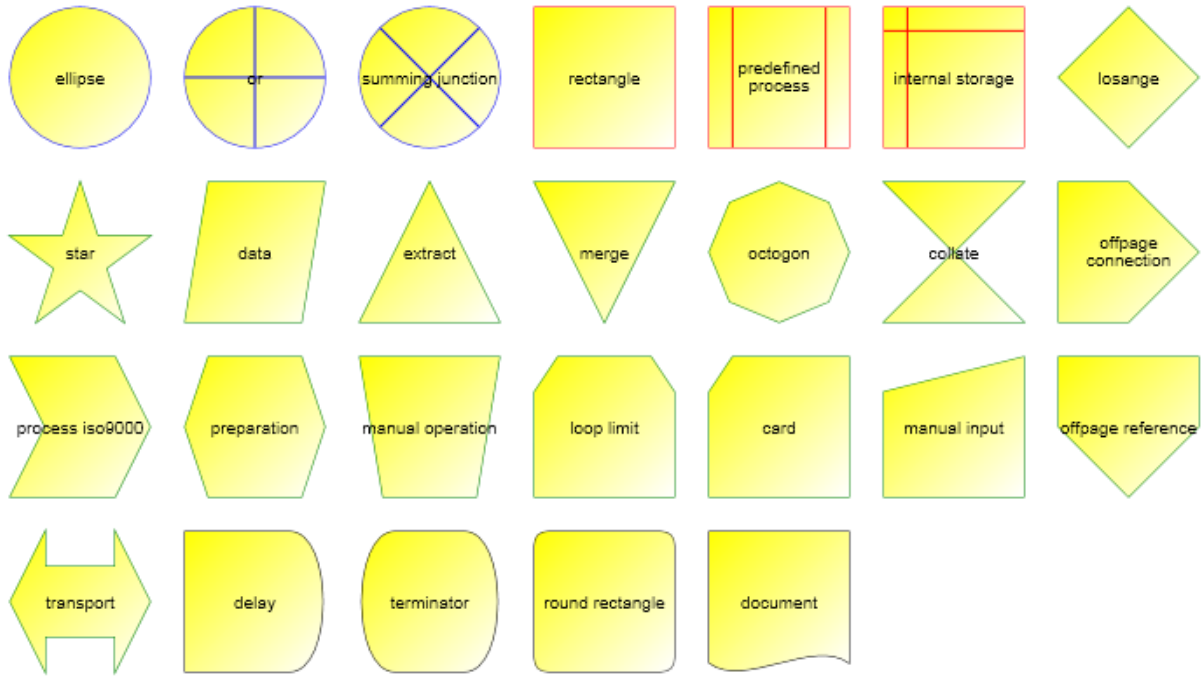


6.3.4 Node shapes

The rules are the following:

- The **shapeFamily** is the first property to consider in order to define the shape of a node. It may have four values: **ellipse**, **rectangle**, **polygon**, **other**.
- If you select 'polygon', then you have to set the **polygon** property.
- If you select 'other', then you have to define the **drawShape** method.
- In every case, you may customize the way the node is drawn inside its border with the **fillShape** property.

The file **tutorial_shapes.htm** provide some examples.



- The first 3 nodes ('ellipse', 'or', 'summingjunction') have a shapeFamily equal to 'ellipse'. However the 'or' and 'summingjunction' nodes have a custom fillShape property value to draw the crosses inside the node. For instance, the 'or' node shape is defined like that:

```
node = flow.addNode(0, 0, 80, 80, "or");
node.shapeFamily = "ellipse";
node.fillShape = function (ctx, x, y, w, h) {
                ctx.beginPath();
                ctx.moveTo(x + w / 2, y);
                ctx.lineTo(x + w / 2, y + h);
                ctx.stroke();
                ctx.beginPath();
                ctx.moveTo(x, y + h / 2);
                ctx.lineTo(x + w, y + h / 2);
                ctx.stroke();
            };
```

- The 3 following nodes ('rectangle', 'predefined process', 'internal storage') have a shapeFamily equal to 'rectangle'. However the 'predefined process' and 'internal storage' nodes have a custom fillShape property value to draw the lines inside the node.

- The 16 following nodes have a shapeFamily equal to 'polygon'. For instance the losange node shape is defined like that:

```
node = flow.addNode(0, 0, 80, 80, "losange");
node.shapeFamily = "polygon";
node.polygon = [[0, 50], [50, 0], [100, 50], [50, 100]];
```

- The last 4 nodes have a shapeFamily equal to 'other'. The drawShape property has been defined for these 4 nodes. For instance the 'delay' node shape is defined like that:

```
node = flow.addNode(0, 0, 80, 80, "delay");
node.shapeFamily = "other";
node.drawShape = function (ctx, x, y, w, h) {
       ctx.beginPath();
       ctx.moveTo(x + 3 * w / 4, y);
       ctx.lineTo(x, y);
       ctx.lineTo(x, y + h);
```

```
        ctx.lineTo(x + 3 * w / 4, y + h);
        ctx.bezierCurveTo(x + w + w / 16, y + h,
                          x + w + w / 16, y, x + w - w / 4, y);
        ctx.closePath();
    };
```

6.3.5 Node pins

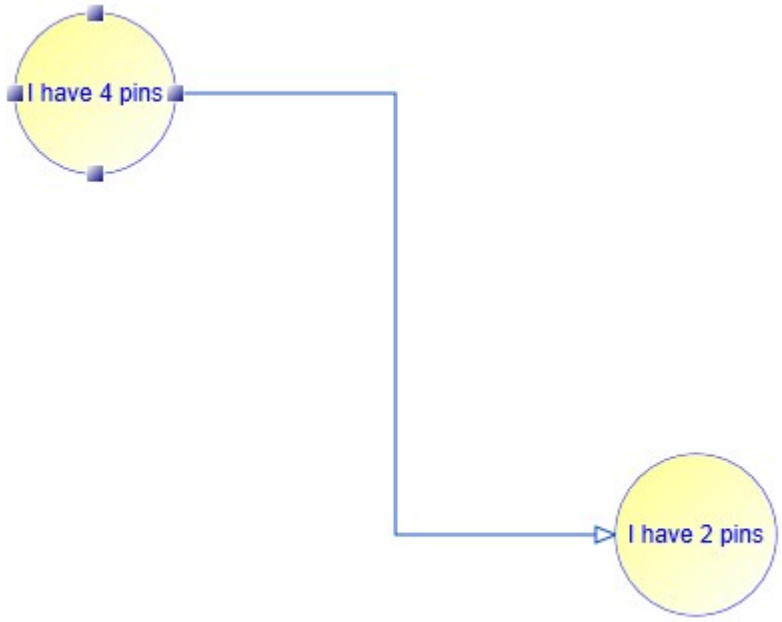
By default, to create a link, you use the 'central pin' of a node.

However, you are not limited to this 'central pin'.

Using the **pins** property, you may attach a set of pins (also called connectors) to a node. For instance, the following line of code create a set of 4 pins for a node:

```
flow.nodeModel.pins = [[0, 50], [50, 0], [100, 50], [50, 100]];
```

The pins property is an array of points whose coordinates is between 0 and 100.



In this example, the line style of the link is 'orthogonal'.

The code used to create such a link is the following:

```
link = flow.addLink(node1, node2, "", 2, 0);
```

Notice the last two parameters that allow setting the index of the origin pin and the index of the destination pin.

6.4 Links

6.4.1 Colors

Two properties allow setting colors for a link:

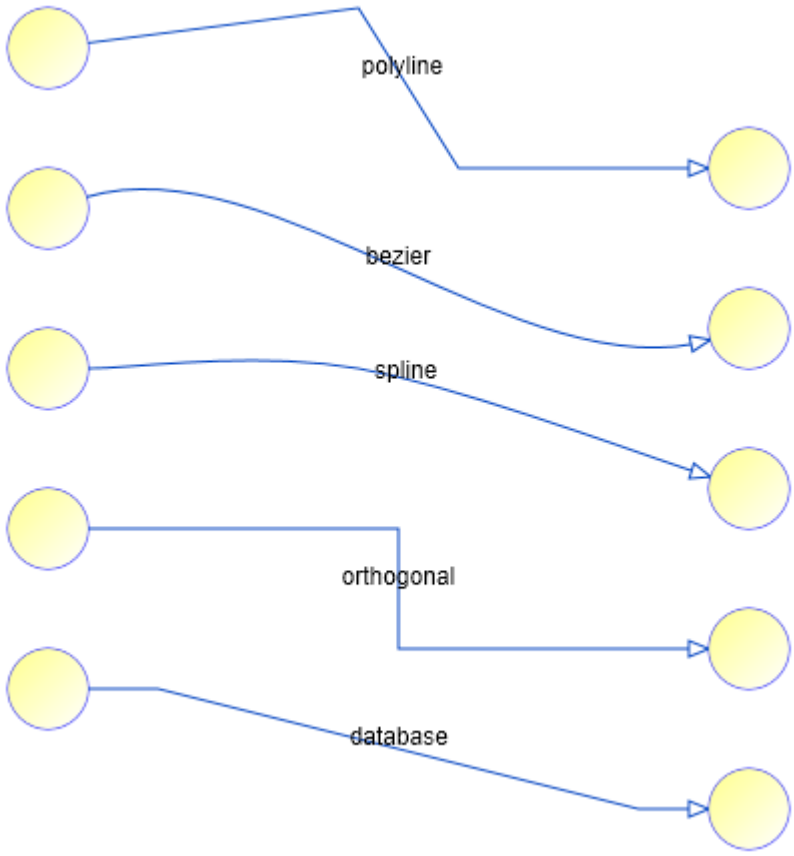
- **strokeStyle** It is the color of the link line.
- **textFillStyle** It is the color of the link text.

6.4.2 Link line style

You can define the line style of a link using the **setLineStyle** method.

WARNING: don't use directly the **lineStyle** property, use the methods **setLineStyle** and **getLineStyle** instead

There are 5 possible values demonstrated in the file **tutorial_linestyle.htm**.



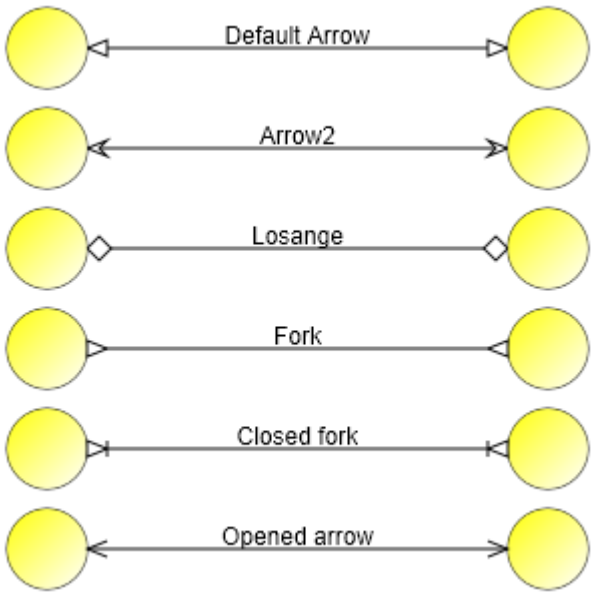
If the line style is 'polyline' or 'spline', you can add as many points as you wish to the link whereas in the other cases, the number of points is fixed: a 'bezier' link or a 'database link' has 4 points. You can move these points but you cannot add new points. The number of points of an orthogonal link is also fixed but at the creation time, depending of the origin and destination pins.

6.4.3 Link arrows

You can define the the origin and destination arrow of a link, using the **arrowOrg** and **arrowDst** properties. The value for both properties is an array of points. For instance:

```
link.arrowDst = [[0, 0], [-10, -4], [-6, 0], [-10, 4]];
```

The file **tutorial_arrows.htm** provide some examples.



6.5 Selection of items

6.5.1 Programmatic selection

To manage the selection a node or a link, you have to use the **getIsSelected** and **setIsSelected** methods. For instance, to select a node:

```
node.setIsSelected(true);
```

6.5.2 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 '**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.

6.5.3 getSelectedItems method

The **getSelectedItems** method of AddFlow allows getting the array of selected items. For instance:

```
// Make each selected nodes red
```

```
var node, selItems, i;

selItems = flow.getSelectedItems();
for (i in selItems) {
       node = selItems[i];
       if (flow.isNode(node)) {
              node.fillStyle = 'red';
              node.refresh();
       }
}
```

6.6 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 **tutorial_navigation.htm** page in the demo sample provided with AddFlow.

We have already spoken of these properties.

- **getItems**. This method returns the array of all nodes and links of the diagram.
- **getSelectedItems**. This method returns the array of all selected nodes and links of the diagram.
- **getLinks**. This method returns the array 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.

For instance, to change the color of all nodes of the diagram:

```
var items = flow.getItems();
for (var i in items) {
       if (flow.isNode(items[i])) {
              emphasizeNode(items[i], 'red', 3);
       }
}
```

For instance, to change the color of each 'out' node of each selected node:

```
selItems = flow.getSelectedItems();
for (i in selItems) {
    if (flow.isNode(selItems[i])) {
        selnode = selItems[i];
        links = selnode.getLinks();
        for (j in links) {
            link = links[j];
            if (link.getOrg() === selnode) {
                node = link.getDst();
                emphasizeNode(node, 'red', 3);
            }
        }
    }
}
```

The emphasizeNode function just changes the color and width of the node border:

```
function emphasizeNode(node, strokeStyle, lineWidth) {
      node.strokeStyle = strokeStyle;
      node.lineWidth = lineWidth;
      node.refresh();
}
```

6.7 Zooming

6.7.1 Programmatic zoom

The z**oom** property allows zooming a diagram. It is a numeric 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 items (nodes, links) in view?

You can implement this feature using the **zoomRectangle, getXExtent and getYExtent** methods.

```
flow.zoomRectangle(0, 0, flow.getXExtent(), flow.getYExtent()));
```

6.7.2 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 '**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.

7) Avanced topics

7.1 Undo/Redo

7.1.1 General features

AddFlow has a property named **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 object. You can also enable/disable the undo/redo with the **canUndoRedo** property of AddFlow.

7.1.2 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.

7.1.3 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.

7.1.4 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.

7.1.5 Undo/Redo customization

The undo/redo can be customized. For that, you have to create a custom task class and then you can insert it in the Undo/Redo buffer with the **submitTask** method. The custom task class must contain a 'redo' and an 'undo' method.

The file **customundo.htm** of the demo 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.

```
var NodePropertiesTask = function (node, oldText, oldTextFillStyle) {
      this.currentItem = node;
      this.code = "NodeProperties";
```

```
        this.group = -1;
        this.groupCode = "AF_none";

        this.oldText = oldText;
        this.oldTextFillStyle = oldTextFillStyle;

        this.redo = function () {
            this.undo();
        };

        this.undo = function () {
            var oldText = this.currentItem.text;
            this.currentItem.text = this.oldText;
            this.oldText = oldText;

            var oldTextFillStyle = this.currentItem.textFillStyle;
            this.currentItem.textFillStyle = this.oldTextFillStyle;
            this.oldTextFillStyle = oldTextFillStyle;

            this.currentItem.refresh();
        };
};
```

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:

```
function submit() {
      var selectedItems, node, textzone, combo;

      selectedItems = flow.getSelectedItems();
      if (selectedItems.length > 0) {
          if (flow.isNode(selectedItems[0])) {
              node = selectedItems[0];

              flow.taskManager.submitTask(
                  new NodePropertiesTask(node, node.text, node.textFillStyle));

              // Set text
              textzone = document.getElementById('text');
              node.text = textzone.value;

              // Set text color
              combo = document.getElementById('colorSelection');
              node.textFillStyle = combo.value;

              node.refresh();
          }
      }
}
```

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

```
       flow.taskManager.submitTask(
             new NodePropertiesTask(node, node.text, node.textFillStyle));
```

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

7.1.6 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.

7.2 Serialization

AddFlow does not provide any serialization feature. However, the **json.htm** example in the demo sample shows how to deals with serialization. In this example, the diagram is saved in JSON format. Only the properties different from the default values defined in the **nodeModel** and **linkModel** properties are saved.

7.3 Performance tuning

7.3.1 beginUpdate / endUpdate

To maintain performance while items are added to the AddFlow control, call the **beginUpdate** method. The beginUpdate method prevents the control from calculating the size of the.The size is updated only when the **endUpdate** method is called.

There is an example in the Stress example of the demo (**stress.htm**).

```
function randomCreation(flow, maxnodes, xarea, yarea, nodesize) {
      var org, dst, link, i, x, y;

      // So that the user will be able to undo the diagram in one time
      if (flow.taskManager.canUndoRedo) {
          flow.taskManager.beginAction("creatediagram");
      }

      flow.beginUpdate();

      org = null;

      for (i = 0; i < maxnodes; i++) {
          // Create a node at a random position
          x = Math.floor(Math.random() * xarea);
          y = Math.floor(Math.random() * yarea);
          dst = flow.addNode(x, y, nodesize, nodesize);

          // and add a link from previous node (link not added if org is null)
          link = flow.addLink(org, dst);

          // The current destination node is the origin node for the next addd link.
          org = dst;
      }

      flow.endUpdate();

      if (flow.taskManager.canUndoRedo) {
          flow.taskManager.endAction();
      }
}
```

7.3.2 Quadtree structure

Better speed performance is provided by using a quadtree structure and it is the case by default. Otherwise. You may use the **useQuadtree** method to determine if the quadtree structure is used or not.

A quadtree is a data structure in which the coordinate space is broken up into 4 quadrants that contain items. If too many items are added into a quadrant, then that quadrant is divided into 4 sub-quadrants. This can provide very fast lookup of items based on the coordinates.

With a quadtree structure, loading a big diagram may take more time because the quadtree structure must be created. However, interactive actions, for instance selecting an item or moving nodes, will be

faster. You may test that with the **stress.htm** file of the demo sample. You may compare how fast is selected an item if the parameter of the useQuadTree method is true or if it is false.

```
flow.useQuadtree(true); // By default, it is true so this call is not needed here
```

7.4 Customizing the user interface

7.4.1 Capabilities

Following properties allow to set capabilities for an AddFlow control and therefore to customize it.

For instance, if you wish to allow only one link between two nodes, you have just to unset the **canMultiLink** property.

Or if you wish to prevent the user from creating links, you have just to unset the **canDrawLink** property.

canChangeDst	Determines whether the user can interactively change the destination of a link.
canChangeOrg	Determines whether the user can interactively change the origin of a link.
canDragScroll	Determines whether drag scrolling is allowed or not.
canDrawNode	Determines whether interactive creation of nodes is allowed or not.
canDrawLink	Determines whether interactive creation of links is allowed or not.
canReflexLink	Determines whether interactive creation of reflexive links is allowed or not.
canMoveNode	Determines whether interactive dragging of nodes is allowed or not.
canSizeNode	Determines whether interaction resizing of nodes is allowed or not.
canStretchLink	Determines whether interactive stretching of links is allowed or not.
canMultiLink	Determines whether you can create several links between two nodes.

canMultiSelect	Determines whether multiselection of nodes is allowed or not.
canSelectOnMouseMove	Indicates whether the selection of items with the mouse is made only when the mouseUp event is fired or at each mouseMove event
canSendSelectionChangedEvent	Determines whether the selectionChanged event is fired or not.
canShowContextHandle	Determines whether context handles are displayed for selected items

7.4.2 Appearances

fillStyle	Returns/sets the canvas background color.
linkModel	Defines the default property values for links.
nodeModel	Defines the default property values for nodes.
ownerDraw	A method allowing making custom drawingg on the AddFlow canvas.
roundedCornerSize	Returns/sets the size of the rounded corners of the link segments.
zoom	The zooming factor

7.4.3 Shadow properties

shadowOffsetX	Returns/sets the X offset of the shadow used to display items
shadowOffsetY	Returns/sets the Y offset of the shadow used to display items
shadowBlur	Returns/sets the amount of blur on the shadow used to display items, in pixels.
shadowColor	Returns/sets the color of the shadow used to display items.

7.4.4 Grid properties

Five properties are provided to manage the grid.

gridSizeX	Returns/sets the horizontal grid size.
gridSizeY	Returns/sets the vertical grid size.
gridSnap	Determines whether nodes are aligned on the grid
gridDraw	Determines whether the grid is displayed or not.
gridStrokeColor	Returns/sets the grid color.

7.4.5 Handle properties

Several properties allow customizing the size and color of the handles used to resize a node or stretch a link.

handleSize	Returns/sets the size of the handles used to select items.
handleGradientColor1	Returns/sets the first color defining the gradientstyle used for handles
handleGradientColor2	Returns/sets the second color defining the gradient style used for the selection handles of nodes and links.
handleStrokeStyle	Returns/sets the color used to draw a selection handle of a node or a link.
contextHandleSize	Returns/sets the size of a context handle. It is the horizontal size. The vertical sise is equal to the horizontal size multiplied by 2 and divided by 5.
contextHandleGradientColor1	Returns/sets the first color defining the gradient style used for context handles
contextHandleGradientColor2	Returns/sets the second color defining the gradient style used for context handles

contextHandleStrokeStyle	Returns/sets the drawing color of the context handles

7.4.6 Pin properties

Several properties allow customizing the size and color of the pins used to create a link.

pinSize	Returns/sets the size of the pins used to draw links.
pinGradientColor1	Returns/sets the first color defining the gradient style used for pins.
pinGradientColor2	Returns/sets the second color defining the gradient style used for pins.
pinStrokeStyle	Returns/sets the color used to draw pins
centralPinGradientColor1	Returns/sets the first color defining the gradient style used for central pins.
centralPinGradientColor2	Returns/sets the second color defining the gradient style used for central pins.
centralPinStrokeColor	Returns/sets the color used to draw central pin

7.4.7 Miscellaneous

bezierSelectionLinesStrokeStyle	Returns/sets the drawing color of the lines used for selected bezier links.
selRectStrokeStyle	Returns/sets the selection rectangle color.
selRectLineWidth	Returns/sets the selection rectangle width.
linkSelectionAreaWidth	Determines the width of the area where the user has to click to select a link.
removePointDistance	Returns/sets a value that determines if the user can remove a link point by dragging the handle to a position where it has a very obtuse angle to its surrounding link points.

8) 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, as part of the AddFlow for HTML5 Professional Edition, a set of 5 graph layout algorithms:

- *Hierarchic layout*
- *Orthogonal layout*
- *Force directed (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, this set of graph layout algorithms is an AddFlow extension and you cannot use it without AddFlow.
- The demo 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.

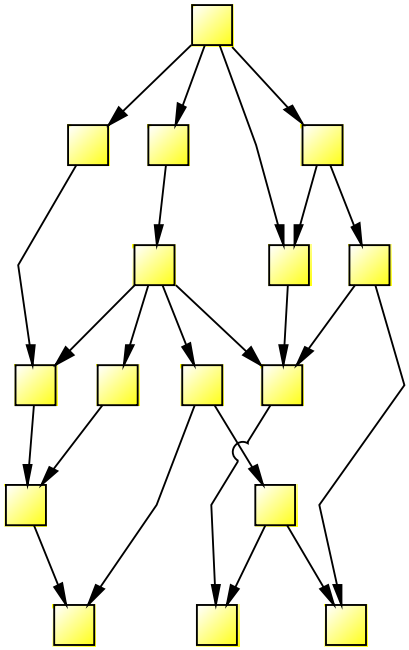
TIP: How to manage so that the layout algorithm applies only to a subset of the graph?

You have to use the **isExcludedFromLayout** property. Only nodes and links whose isExcludedFromLayout property is false are involved in a layout. This will allow you to make the layout algorithm to ignore some nodes or links.

8.1.1 Hierarchic layout

8.1.1.1 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 -

8.1.1.2 Code example

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

```
LayoutFlow.Hierarchic(flow,
       50, // Sets the distance between adjacent levels
       50, // Sets the distance between adjacent nodes
       'north', // The orientation of the graph
       5, // x Margin
       5, // y 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.

8.1.1.3 Limitation

It works with any graph, connected or not.

8.1.1.4 Side Effect

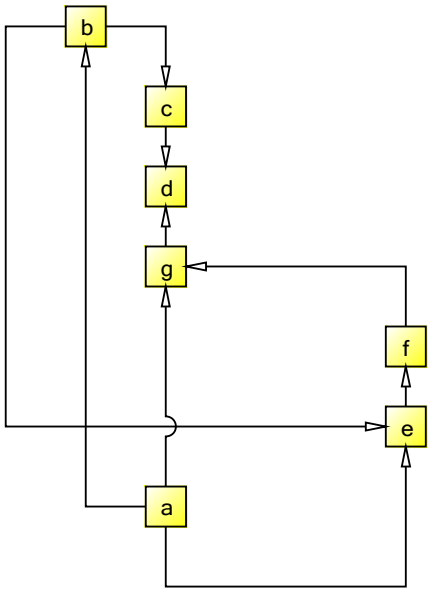
After the layout execution:

- the line style of the links is **polyline**
- the **isOrgPointAdjustable** property of the destination node of a link is set to false.
- the **isDstPointAdjustable** property of the origin node of a link is set to false.

8.1.2 Orthogonal layout

8.1.2.1 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 -

8.1.2.2 Code example

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

```
LayoutFlow.Orthogonal(flow,
       'north', // The orientation of the graph
       40, // The horizontal grid size
       40, // The vertical grid size
       50, // The node size (in percentage of the grid size)
       5,  // x Margin
       5); // y Margin
```

8.1.2.3 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.

8.1.2.4 Side Effect

After the layout execution:

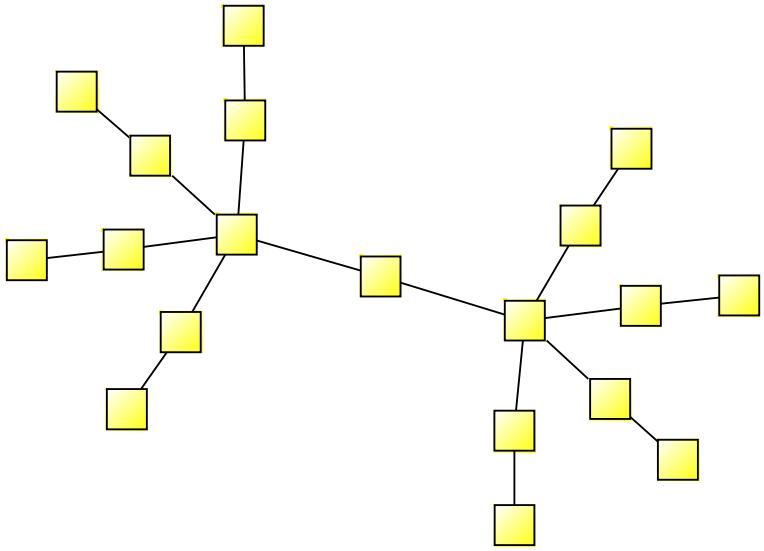
- the size of the nodes is changed. If the graph is a graph of maximum degree four, then each node has the same size (determined by the **xgridSize** and **ygridSize** properties). If the degree of a node is higher than four, then the height of the node is expanded.

- the line style of the links is **polyline.**
- the **isOrgPointAdjustable** property of the destination node of a link is set to true.
- the **isDstPointAdjustable** property of the origin node of a link is set to true.

8.1.3 Force directed (symmetric) layout

8.1.3.1 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 -

8.1.3.2 Code example

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

```
LayoutFlow.ForceDirected(flow,
       50,    // Sets the distance between nodes
       5,     // x Margin
       5,     // y Margin
       true,  // The layout takes account of the isXMoveable and isYMoveable
              // properties of the nodes.
       true,  // The nodes are placed randomly at the beginning of the layout
       true); // The step event is fired
```

8.1.3.3 Limitation

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

8.1.3.4 Side Effect

After the layout execution:

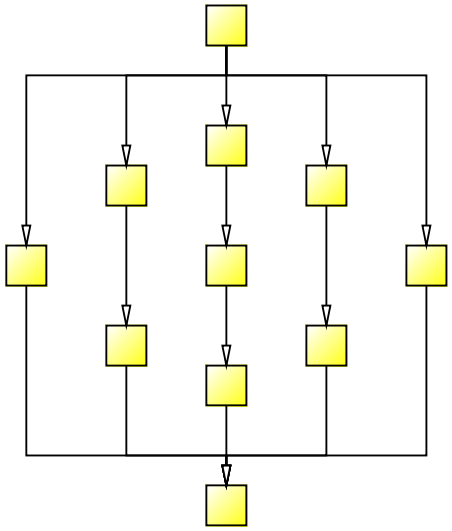
- the line style of the links is **polyline** and each link is composed of only one segment.
- the **isOrgPointAdjustable** property of the destination node of a link is set to false.
- the **isDstPointAdjustable** property of the origin node of a link is set to false.

8.1.4 Series-parallel layout

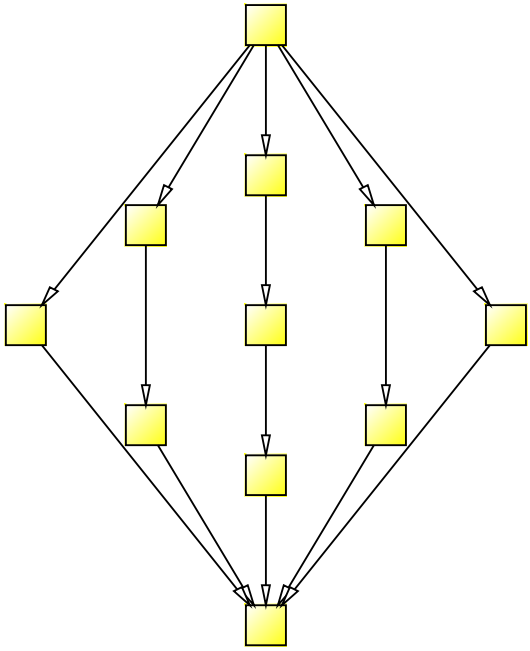
8.1.4.1 Purpose

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.

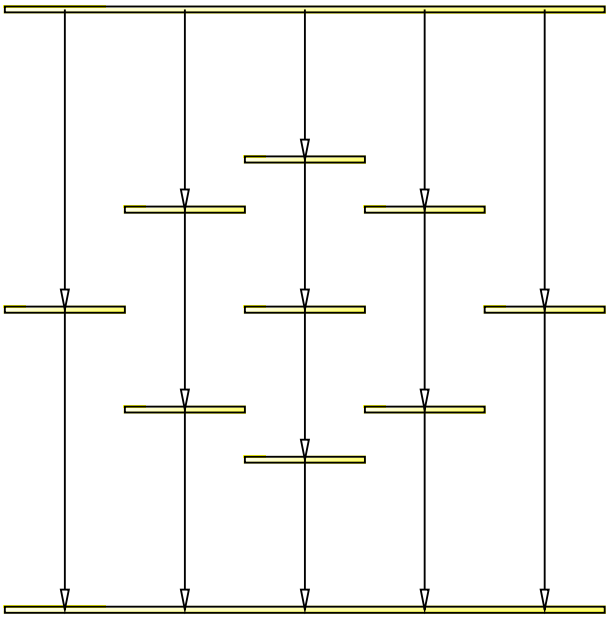
We use an algorithm (described in the book "Drawing Graphs" Michael Kaufmann - Dorothea Wagner) that allows drawing series-parallel digraphs with as much symmetry as possible.



- SP layout: drawingStyle = 'busOrthogonalDrawing'



- SP layout: drawingStyle = 'straightLine' -



- SP layout: drawingStyle = 'visibilityDrawing' -

8.1.4.2 Code example

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

```
LayoutFlow.SP(flow,
       'busOrthogonalDrawing', // The drawing style
       'north', // The orientation of the graph
       80, // Sets the distance between adjacent levels
       80, // Sets the distance between adjacent nodes
       30, // The vertex horizontal size
```

```
30, // The vertex vertical size
5,  // x Margin
5); // y Margin
```

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

8.1.4.3 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).

8.1.4.4 Side Effect

After the layout execution:

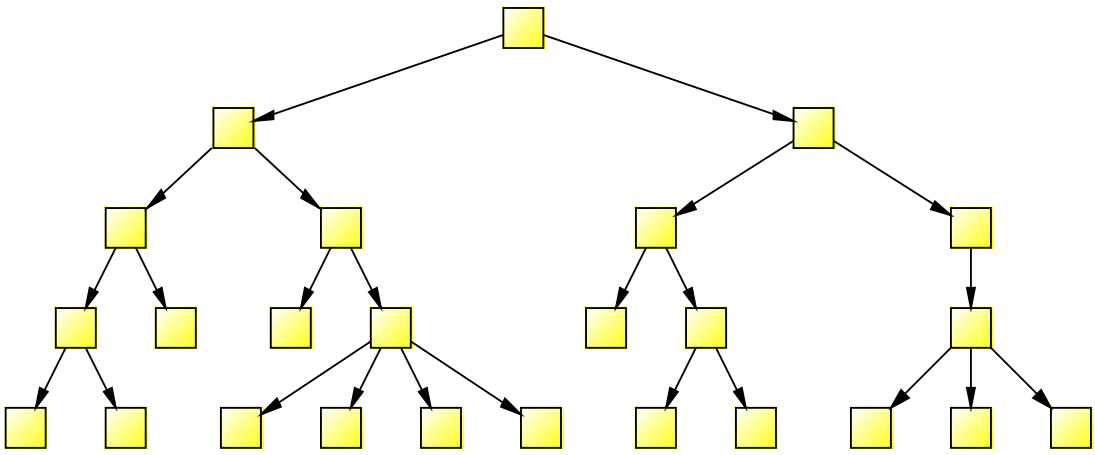
- the line style of the links is **polyline**.
- the **isOrgPointAdjustable** property of the destination node of a link is set to false.
- the **isDstPointAdjustable** property of the origin node of a link is set to false.

8.1.5 Tree layout

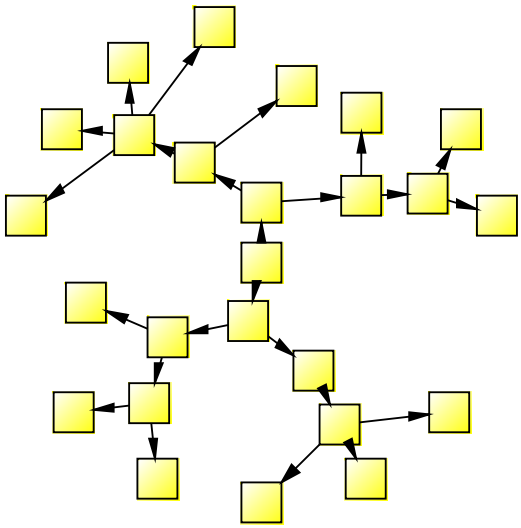
8.1.5.1 Purpose

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 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 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' -

8.1.5.2 Code example

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

```
LayoutFlow.Tree(flow,
       50, // Layer distance
       50, // Vertex distance
       'layered', // The drawing style (layered or radial)
       'north', // The orientation of the graph
       5, // x Margin
       5, // y Margin
```

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

8.1.5.3 Limitation

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

8.1.5.4 Side Effect

After the layout execution:

- the line style of the links is **polyline**
- the **isOrgPointAdjustable** property of the destination node of a link is set to false.
- the **isDstPointAdjustable** property of the origin node of a link is set to false.