Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

AddFlow Script v 1.0
Tutorial

June 2024

Lassalle Technologies

http://www.lassalle.com

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

CONTENTS

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

1) Introduction

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

AddFlow Script 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, …

AddFlow Script is the successor of AddFlow for HTML5. It is faster and it provides more features like labels or jumps. It is written in TypeScript. It can be used with Angular.

Purpose of this tutorial

This tutorial provides information on:

- licensing
- creating diagrams programmatically, using the AddFlow Script API
- creating diagrams interactively

Who should use this tutorial?

This guide is intended for application programmers building web applications.

Samples

AddFlow Script is installed with one demo sample. Its source code (html, Javascript or TypeScript) is provided.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

2) 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/script/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/script/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.

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. 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.

3. Do you provide an Open Source license ?

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

4. Is AddFlow runtime-royalty free ?

Yes.

5. How many developers can use AddFlow Script ?

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

6. Do you offer multi-pack discounts ?

Yes, we offer the following type of licenses:

- Single developer license: allows just one developer

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

- Team license: allows 4 developers

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

- Enterprise license: allows all developers of an enterprise

7. How many projects can I create with a license of AddFlow Script ?

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

8. Will purchasing guarantee me upgrades ?

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

9. Do you provide refunds ?

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

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

3) Getting Started

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

```
<script>var exports = {"__esModule": true};</script>
<script src="../../addflow/flow.js" type="text/javascript"></script>
```

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

```
<div id="divFlow" style="border: 1px solid green; width: 900px; height: 500px;
   overflow: hidden;">
      <canvas id="canvasFlow" width="900" height="500" style="touch-action: none;">
      </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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

4) Interactive creation of a diagram

4.1 Overview

It includes:

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

It supports also the scrolling of diagrams and the use of grids.

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

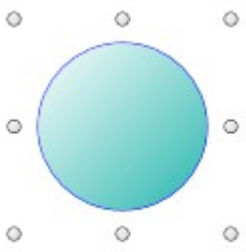
And a set of methods and properties allow implementing a powerful Undo/Redo feature.

4.2 Create a diagram interactively

4.2.1 Draw a node

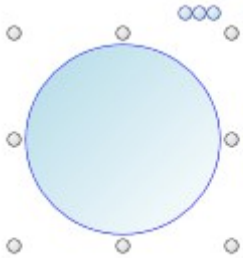
Bring the mouse cursor into the control, press the left button, move the mouse and release the left button. You have created an elliptic node. This node is selected: that's why 8 handles (little circles) are displayed.

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 **isContext** 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:

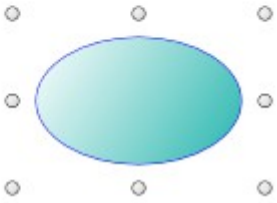
Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



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.

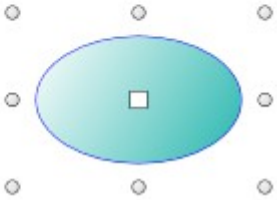
4.2.2 Draw a link

Draw a second node.

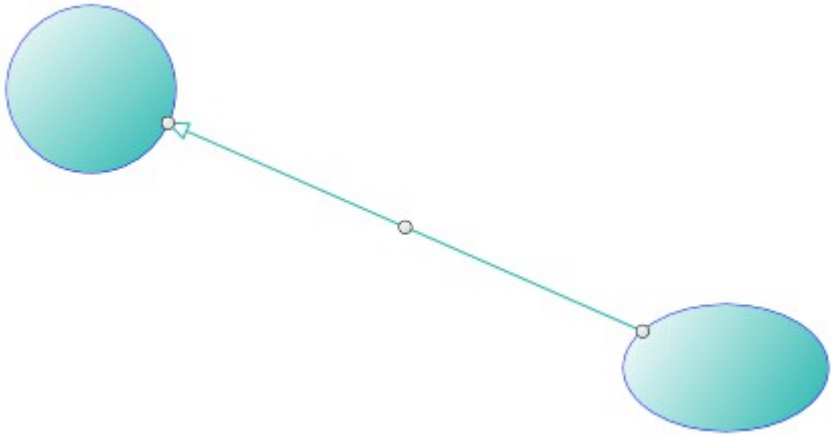


Then bring the mouse cursor above the second node. A small circle handle is then displayed at the center of the selected node.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



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.

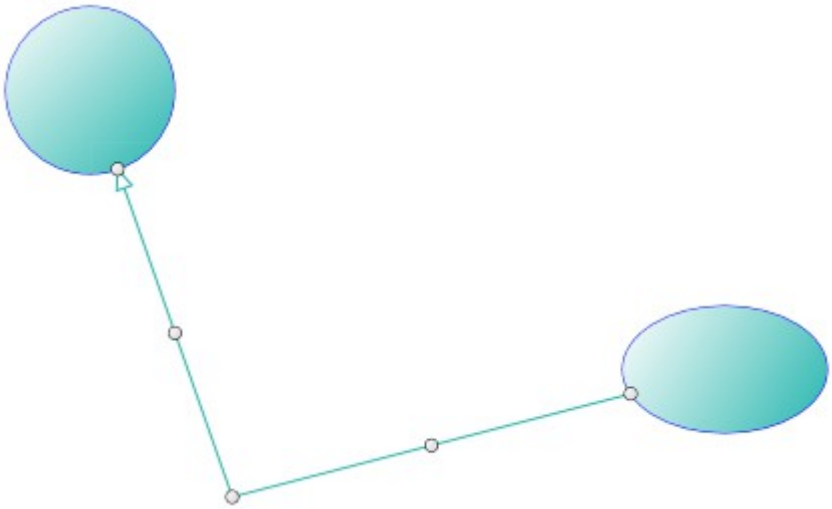
But, as you will see later in this tutorial, you can also use another way to create links by using **pins**.

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

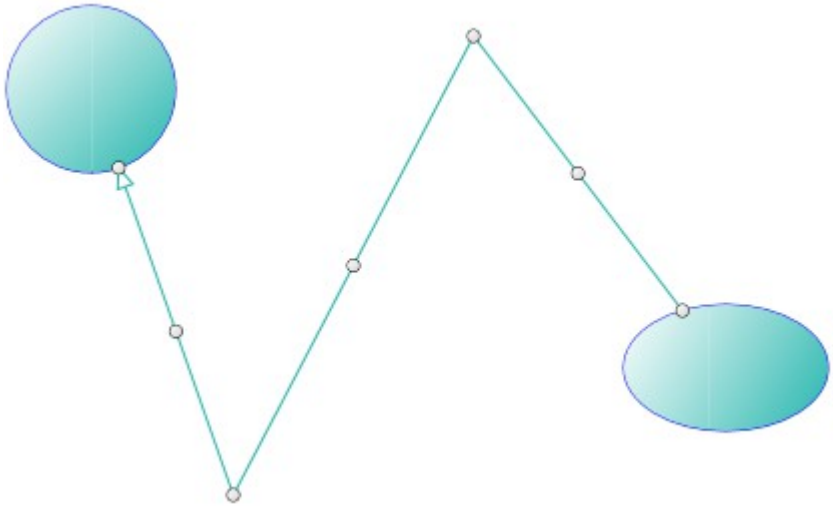
4.2.3 Stretch a link

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).

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



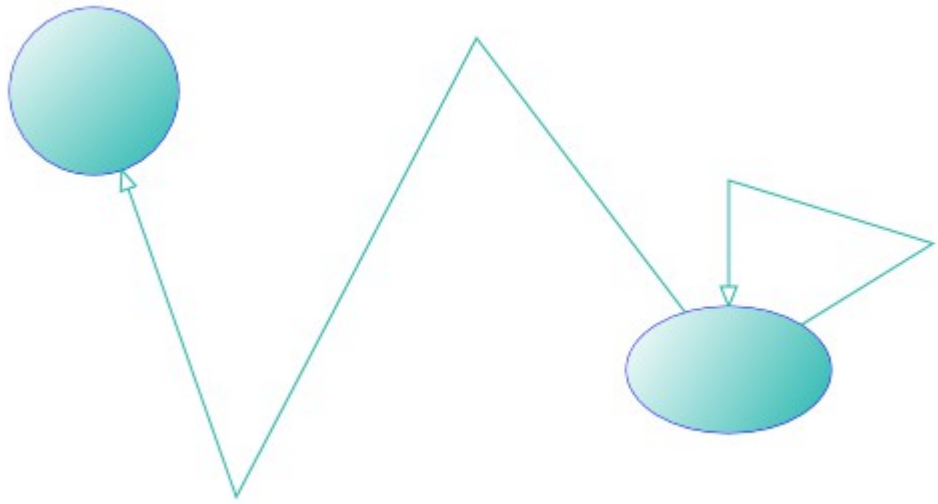
Create another segment



4.2.4 Draw a reflexive link

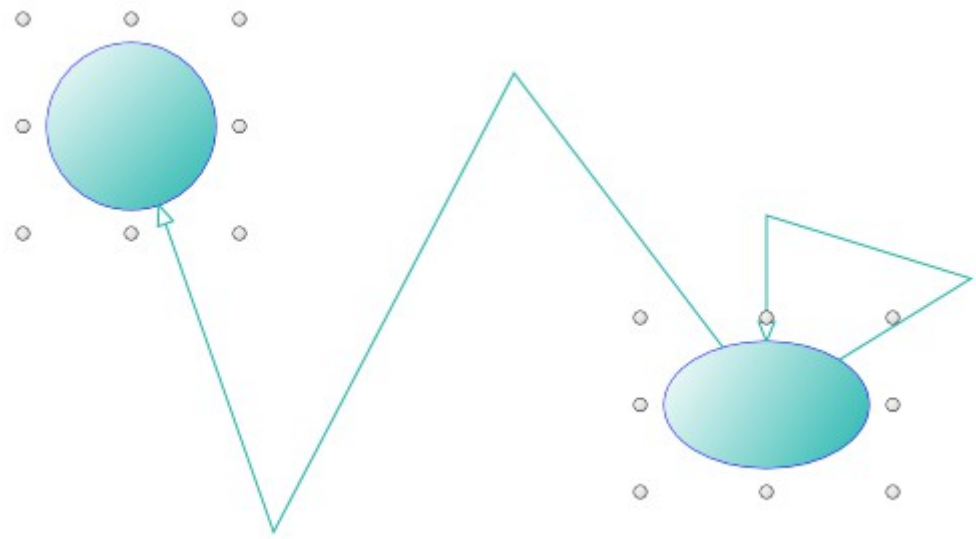
Select a node by clicking on it. Then bring the mouse cursor above the small diamond handle at the center of the selected node. Press the left button, move the mouse outside the selected node, then move it inside the selected node again, then release the left button. You have created a reflexive link,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

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

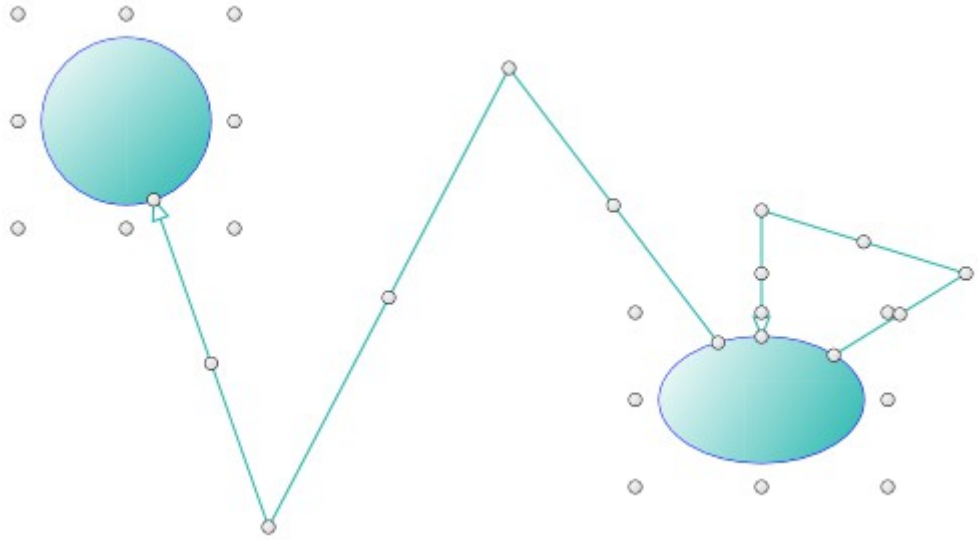


4.2.5 Multiselection

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



Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



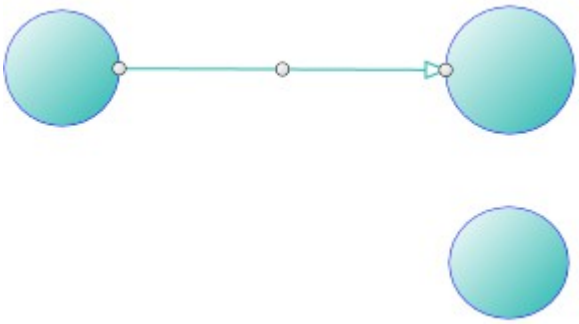
There is another way to perform multiselection, using the **mouseAction** property and assigning it the **mouseAction.Selection** value. Then you can select several nodes and links: you bring the mouse cursor into the AddFlow control, press the left button, move the mouse and release the left button.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

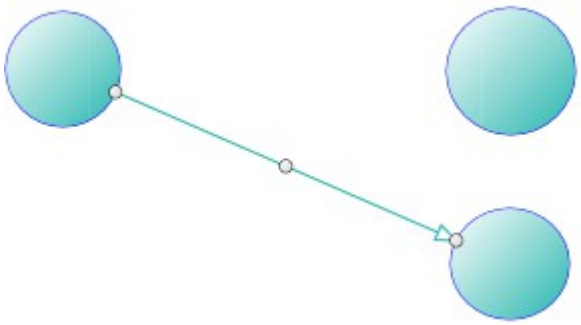
4.2.7 Change the destination or the origin node of a link

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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



4.2.8 Selecting, zooming panning.

Selection

If the **mouseAction** 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 **mouseAction** 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 **mouseAction** 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

5) Programmatic creation of a diagram

5.1 Overview

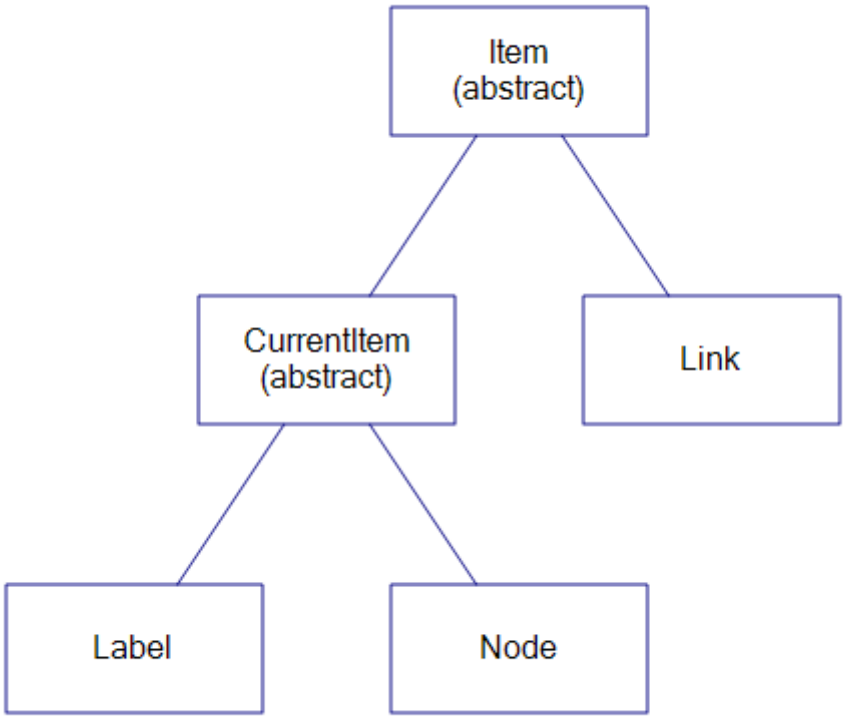
In this chapter we will focus on how to create a diagram programmatically.

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

The main class is the **AddFlow** class that derives from the Canvas class. It contains a DrawingVisual object that which is used to display the diagram. The **Visual** property returns this DrawingVisual object.

An AddFlow diagram contains three kinds of objects, **Node, Link** and **Label** objects. The Node and Label classes derives from the **ContentItem** class. The ContentItem and Link classes derive from the **Item** class.

The Item and ContentItem classes are abstract classes. You cannot use them directly but instead use objects that derive from the Node, Link and Label classes.



5.2 AddFlow Items

5.2.1 Item

The Item class represents an item in the diagram. All classes representing diagram elements derive from Item.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

The main purpose is to provide common methods and properties of every diagram element: nodes, links, labels.

Note that it is an abstract class. You cannot use it directly but instead use objects that derive from the Node, Link or Label classes.

The Item class provides some style properties, data properties and behavior properties used by all the items of an AddFlow diagram.

Style properties

- **strokeStyle** the color used to draw the item
- **fillStyle** the color used to fill the item
- **gradientFillStyle** the color used to create a gradient with the fillStyle color
- **textFillStyle** the color used to display the text of the item
- **lineWidth** the thickness of the lines used to draw the item
- **isShadowed** determines whether a shadow is displayed for the item
- **font** the font used to display the text of the item
- **opacity** opacity of the item

Data properties

- **text** defines the string displayed inside or near the item.
- **tag** defines a tag associated to the item.
- **labels** is the list of labels associated to the item.

Behaviour properties

- **isSelected** returns/sets a flag indicating if the item is selected or not.
- **isHitTestVisible** determines whether the item can be hit tested
- **isSelectable** determines whether the item is selectable by clicking on it with the mouse or unselectable (readonly or inactive)
- **isHidden** determines whether the item is hidden
- **isExcludedFromLayout** determines whether the item is excluded by a layout algorithm

5.2.2 ContentItem

A ContentItem object is an Item object that has a content which may be a string or/and an image. Nodes and labels are ContentItem objects.

The main purpose is to provide common methods and properties for nodes and labels. Note that it is an abstract class. You cannot use it directly but instead use objects that derive from Node or Label classes.

Style properties

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

- **shapeFamily** ellipse, rectangle, polygon, other
- **polygon** the set of points defining the content item shape if if the shapeFamily property is 'polygon'
- **drawShape** a method used to draw the content item shape if the shapeFamily property is 'other'
- **fillShape** a method used to fill the content item shape if the shapeFamily property is 'other'
- **textPosition** specify the position of the text in the content item
- **imagePosition** specify the position of the image in the content item
- **textLineHeight** the height of a line of the text of the content item
- **textMargin** the margin of the text in the content item
- **imageMargin** the margin of the image in the content item

Data properties

- **image** the image displayed in the content item

Behaviour properties

- **isXMoveable** determines whether the content item can be moved horizontally moved or not
- **isYMoveable** determines whether the content item can be moved vertically moved or not
- **isXSizeable** determines whether the content item can be resized horizontally moved or not
- **isYSizeable** determines whether the content item can be resized vertically moved or not

5.2.3 Node

A node (also called vertice or entity) is a content item that can be linked to another node.

Connection properties

- **links** collection property that allows getting all the links of the node.
- **pins** defining a list of anchor points where a link can be attached to the node.

Behaviour properties

- **isInLinkable** determine if “in” links are allowed.
- **isOutLinkable** determine if “out” links are allowed.
- **isContext** determines if a context handle is displayed when the link is selected

5.2.4 Link

A link (also called edge, relation or arc) is an Item object allowing 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.

Connection properties

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

- **org** origin node of the link
- **dst** destination node of the link
- **pinOrgIndex** origin pin index of the link
- **pinDstIndex** the destination pin index of the link

Layout properties

- **points** collection of points that define the segments of the link

Style properties

- **lineStyle** the link line style (polyline, orthogonal, bezier, spline, database)
- **arrowOrg** origin arrow shape
- **arrowDst** destination arrow shape
- **roundCornerSize** the size of the rounded corners of the link segments
- **jumpSize** determines the size of the jump displayed at the intersection of 2 links.
- **isOrientedText** determines whether the link text can be drawn in the same direction as the link itself

Behaviour properties

- **isStretchable** determines whether the link is stretchable or not. When a link is not stretchable, the user cannot interactively stretch it with the mouse
- **isContext** determines if a context handle is displayed when the link is selected
- **isOrgPointAdjustable** determines whether the first link point can be changed
- **isDstPointAdjustable** determines whether the last link point can be changed

WARNING: A link is composed of several segments defined by the **points** property, a list 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.**

5.2.5 Label

A label is a content item that can be owned by any item (node, link or even another label).

- **owner** allows getting and setting the owner of the label.
- **dock** returns/sets the DockStyle of the label. This property is relevant only if the label is attached to a ContentItem object.
- **anchorPositionOnLink** returns/sets a value which defines the position of the label near the link. This property is relevant only if the label is attached to a link.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

5.3 Collections of items

AddFlow provide the following collections:

- **items**: the collection of all items of the diagram
- **selectedItems**: collection of all selected items of the diagram
- **links**: collection of all links (in and out) of a node
- **labels**: collection of labels of an AddFlow item.

WARNING: Those collections are provided for only for the AddFlow infrastructure and for enumeration purposes. Don't use them for adding or removing items.

5.4 Diagram creation

5.4.1 Our first program

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

```
<script>var exports = {"__esModule": true};</script>
<script src="../../addflow/flow.js" type="text/javascript"></script>
```

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

```
<div id="divFlow" style="border: 1px solid green; width: 900px; height: 500px;
   overflow: hidden;">
      <canvas id="canvasFlow" width="900" height="500" style="touch-action: none;">
      </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('canvasFlow');
   flow = new Lassalle.Flow(canvas);

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

    node2 = new Node(flow, 270, 179, 80, 80, "Second node");
    flow.addNode(node2);

    node3 = new Node(flow, 40, 230, 80, 80, "Third node");
    flow.addNode(node3);

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

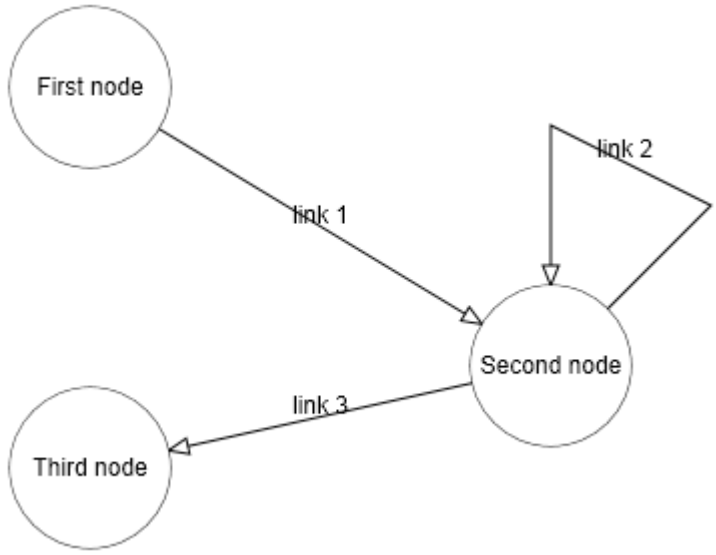
Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

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

    link3 = new Link(flow, node2, node3, "link 3");
    flow.addLink(link3);

   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. The last parameter of the Node constructor is optional. For instance, to create the first node, you could have written:

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

To create a link, you have to use the **addLink** method. There is no other method. Only the three first parameters are mandatory for the Link constructor. For instance, to create the first link, you could have written:

```
link1 = new Link(flow, node1, node2);
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

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

The Link constructor 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.

5.4.2 Changing property values

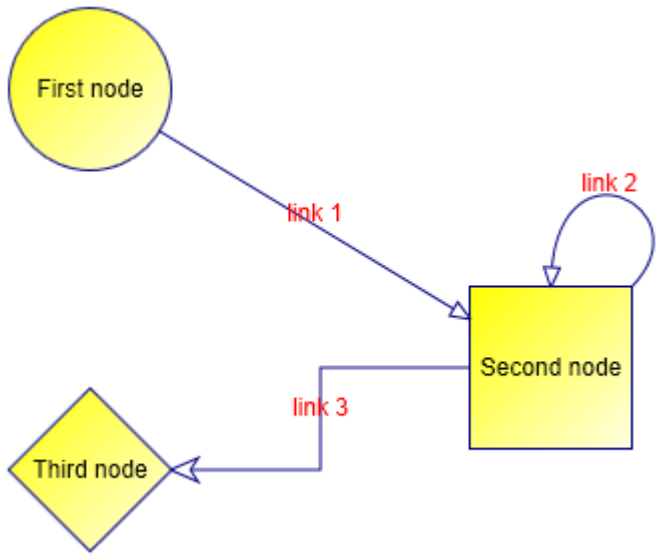
Now let us replace the createDiagram method by the following new one:

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

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

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

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

   node3 = new Node(flow, 40, 230, 80, 80, "Third node");
   node3.fillStyle = 'yellow';
   node3.gradientFillStyle = 'lightyellow';
   node3.strokeStyle = 'navy';
   node3.lineWidth = 2;
   node3.shapeFamily = ShapeFamily.Polygon;
   node3.polygon = [{ x:0, y:50 }, { x:50, y:0 }, { x:100, y:50 }, { x:50, y:100 }];
   flow.addNode(node3);

   // Create 3 links
   link1 = new Link(flow, node1, node2, "link 1");
   link1.strokeStyle = 'navy';
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
  link1.textFillStyle = 'red';
  flow.addLink(link1);

  link2 = new Link(flow, node2, node2, "link 2");
  link2.strokeStyle = 'navy';
  link2.textFillStyle = 'red';
  link2.lineStyle = LineStyle.Bezier;
  flow.addLink(link2);

  link3 = new Link(flow, node2, node3, 'link 3');
  link3.strokeStyle = 'navy';
  link3.textFillStyle = 'red';
  link3.lineStyle = LineStyle.Orthogonal;
  link3.arrowDst = [{ x:0, y:0 }, { x:-14, y:-6 }, { x:-10, y:0 }, { x:-14, y:6 }];
  flow.addLink(link3);

  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 items.

5.4.3 Default property values

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

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

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

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

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

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

  node2 = new Node(flow, 270, 179, 80, 80, "Second node");
  node2.shapeFamily = ShapeFamily.Rectangle;
  flow.addNode(node2);

  node3 = new Node(flow, 40, 230, 80, 80, "Third node");
  node3.shapeFamily = ShapeFamily.Polygon;
  node3.polygon = [{ x:0, y:50 }, { x:50, y:0 }, { x:100, y:50 }, { x:50, y:100 }];
  flow.addNode(node3);

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

  link2 = new Link(flow, node2, node2, "link 2");
  link2.lineStyle = LineStyle.Bezier;
  flow.addLink(link2);

  link3 = new Link(flow, node2, node3, 'link 3');
  link3.lineStyle = LineStyle.Orthogonal;
  link3.arrowDst = [{ x:0, y:0 }, { x:-14, y:-6 }, { x:-10, y:0 }, { x:-14, y:6 }];
  flow.addLink(link3);

  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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

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.

5.4.4 Stretching the links

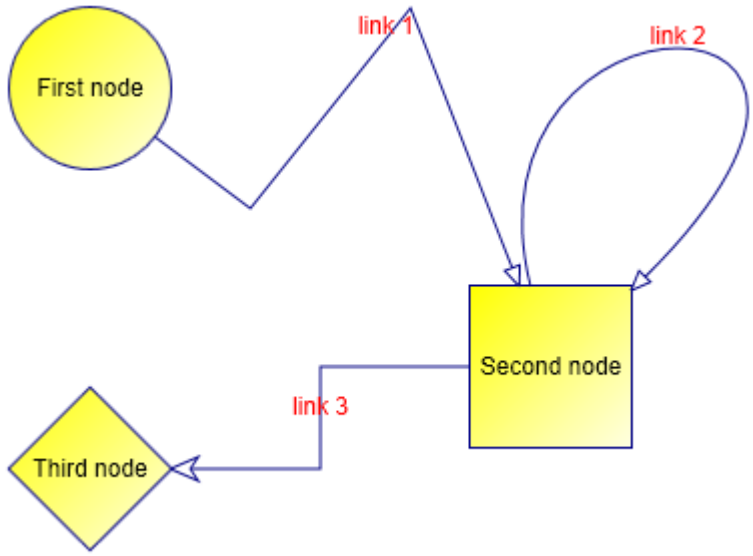
We would like to add segments to our links. The following createDiagram method demonstrates how to do that.

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

  canvas = document.getElementById('canvasFlow');
  flow = new 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 = new Node(flow, 40, 40, 80, 80, "First node");
  flow.addNode(node1);

  node2 = new Node(flow, 270, 179, 80, 80, "Second node");
  flow.addNode(node2);
  node2.shapeFamily = ShapeFamily.Rectangle;

  node3 = new Node(flow, 40, 230, 80, 80, "Third node");
  flow.addNode(node3);
  node3.shapeFamily = ShapeFamily.Polygon;
  node3.polygon = [{ x:0, y:50 }, { x:50, y:0 }, { x:100, y:50 }, { x:50, y: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 = new Link(flow, node1, node2, "link 1");
  flow.addLink(link1);

  // Add 2 points (therefore 2 segments) to this first link
  link1.addPoint({ x: 160, y: 140 });
  link1.addPoint({ x: 240, y: 40 });

  link2 = new Link(flow, node2, node2, "link 2");
  flow.addLink(link2);
  link2.lineStyle = LineStyle.Bezier;
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
  // Stretch this reflexive link
  link2.setPoint({ x: 260, y: 20 }, 1);
  link2.setPoint({ x: 520, y: 20 }, 2);

  link3 = new Link(flow, node2, node3, 'link 3');
  flow.addLink(link3);
  link3.lineStyle = LineStyle.Orthogonal;
  link3.arrowDst = [{ x:0, y:0 }, { x:-14, y:-6 }, { x:-10, y:0 }, { x:-14, y: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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

- You cannot change the first point of the Points collection except if the **isOrgPointAdjustable** property is true or if the origin node has pins (you can change the link pin)
- You cannot change the last point of the Points collection except if the **isDstPointAdjustable** property is true or if the destination node has pins (you can change the link pin)
- You can change each other point of the **points** collection in any case.

5.5 More informations about ContentItem objects

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.

5.5.1 Colors for contentItem objects and nodes

Four properties allow setting colors for a content item:

- **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.

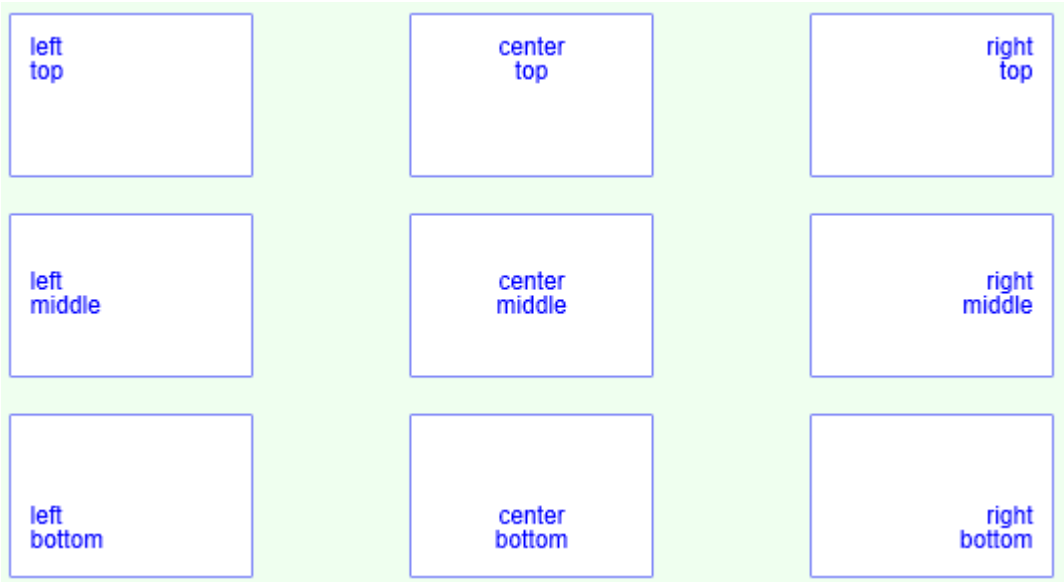
5.5.2 Displaying a text in a ContentItem object

You can associate a text to a content item with the **text** property. Two properties allow placing the text inside the content item: **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 };
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



The **font** property allows changing the font used to display the content item 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.

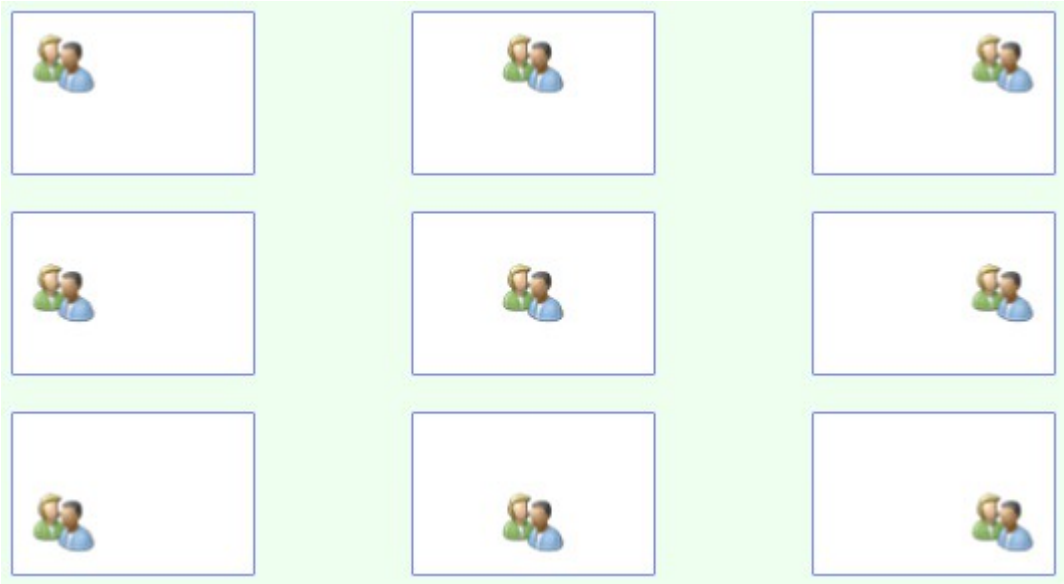
5.5.3 Displaying an image in a node

You can associate an image to a content item with the **image** property.

Two properties allow placing the image inside the content item: **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 };
```



Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

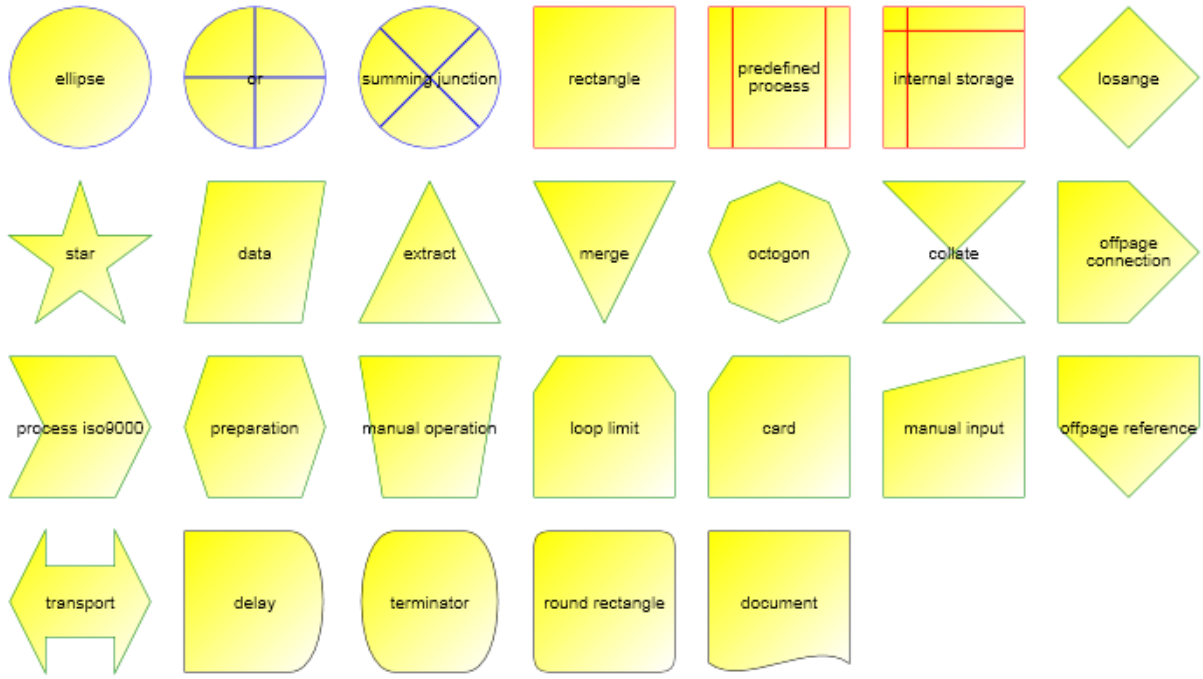
5.5.4 Node shapes

(remember that the Node class derives from the ContentItem class.)

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();
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
        };
```

- 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();
    };
```

5.5.5 Node pins

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

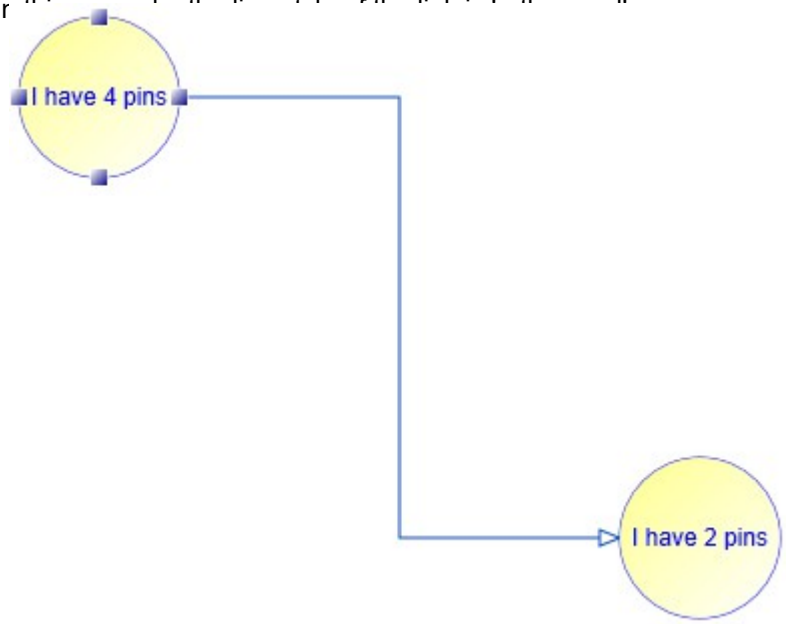
However, you are not limited to this 'central pin'.

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 = [{x:0, y:50},{x:50, y:0},{x:100, y:50},{x:50, y:100}];
```

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

Ir



Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

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

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

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

5.6 More informations about links

5.6.1 Colors

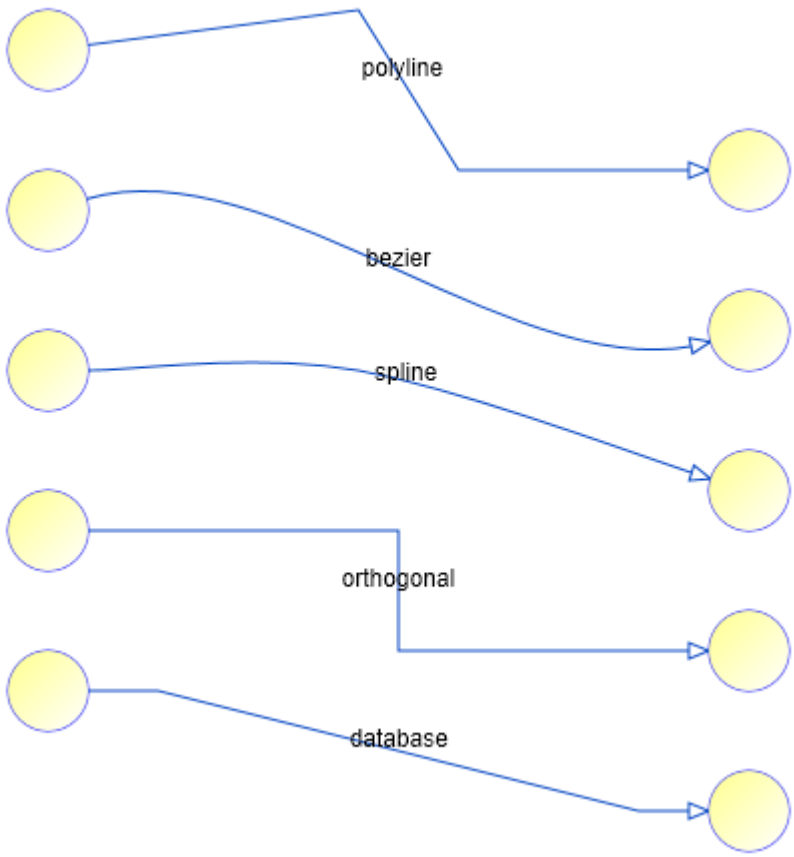
Two properties allow setting colors for a link:

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

5.6.2 Link line style

You can define the line style of a link using the **lineStyle** property.

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.

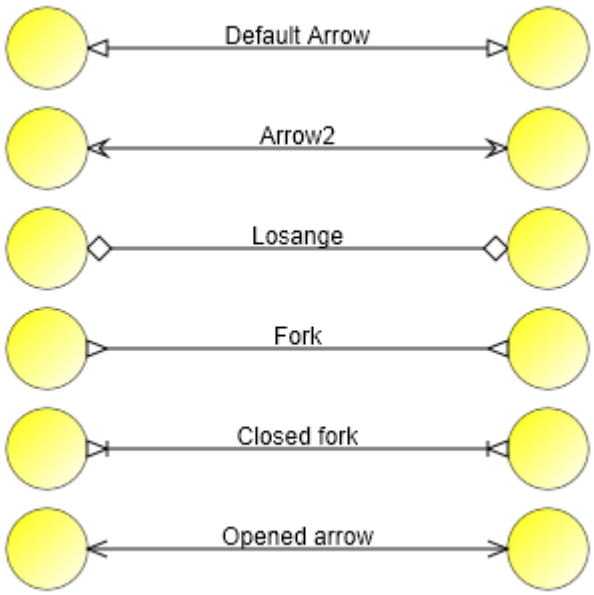
Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

5.6.3 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.



5.7 Displaying link intersections

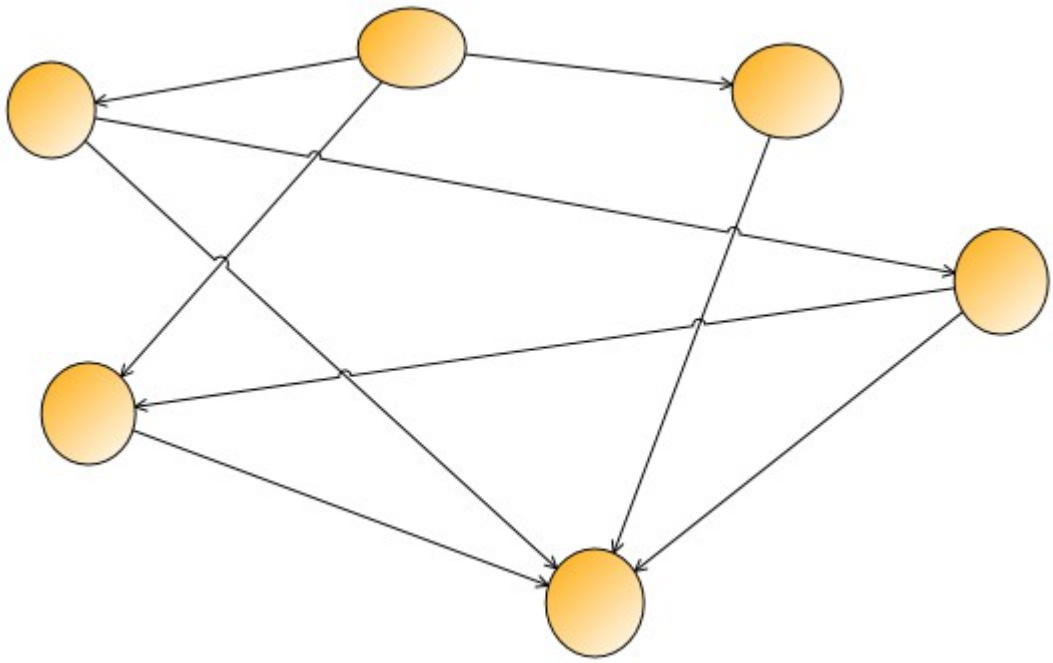
AddFlow uses a poweful algorithm to find intersections between the link segments.

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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



5.8 More informations about labels

A label may be used just to display a legend in a diagram. It may also be attached to an AddFlow item. The property that allows attaching a label to an item is the **owner** property. For an AddFlow item, the l**abels** property returns the collection of labels owned by this item. For instance:

```
for (let label of node.labels) {
    label.strokeStyle = 'red';
}
```

WARNING: The Captions collection property is provided only for the AddFlow infrastructure and for enumeration purposes. Don't use it for adding or removing captions. Use instead the Owner property of captions.

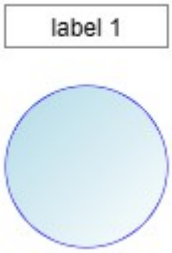
This is demonstrated in the **tutorial_labels.htm** file of the demo.

In the following code, a node and a label attached to this node are created:

```
node1 = new Node(flow, 40, 100, 80, 80, "");
flow.addNode(node1);
labelOwnedByNode = new Label(flow, 40, 60, 80, 20, "label 1", node1);
flow.addLabel(labelOwnedByNode);
```

We obtain the following diagram:

node, its label will follow it.

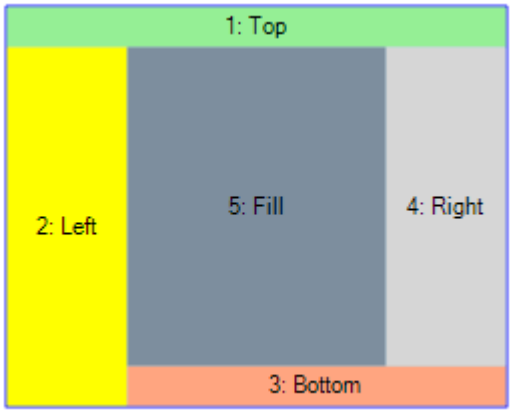


Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

The **dock** property of the Label class allows placing several labels inside a node. It works the same way as the Dock property for controls. For instance, in the following example, the first label is placed at the top of the owner item because its dock property is set to DockStyle.Top.

```
// Dock property demo
// Create the "parent" node
node2 = new Node(flow, 460, 40, 250, 200, "");
node2.shapeFamily = ShapeFamily.Rectangle;
flow.addNode(node2);

// Create 5 child nodes
child1 = new Label(flow, 50, 130, 90, 20, "1: Top", node2);
child1.strokeStyle = 'transparent';
child1.fillStyle = 'lightgreen';
child1.isHitTestVisible = false;
child1.dock = DockStyle.Top;

child2 = new Label(flow, 50, 160, 60, 20, "2: Left", node2);
child2.strokeStyle = 'transparent';
child2.fillStyle = 'yellow';
child2.isHitTestVisible = false;
child2.dock = DockStyle.Left;

child3 = new Label(flow, 50, 190, 60, 20, "3: Bottom", node2);
child3.strokeStyle = 'transparent';
child3.fillStyle = 'lightsalmon';
child3.isHitTestVisible = false;
child3.dock = DockStyle.Bottom;

child4 = new Label(flow, 50, 220, 60, 20, "4: Right", node2);
child4.strokeStyle = 'transparent';
child4.fillStyle = 'lightgray';
child4.isHitTestVisible = false;
child4.dock = DockStyle.Right;

child5 = new Label(flow, 50, 250, 90, 20, "5: Fill", node2);
child5.strokeStyle = 'transparent';
child5.fillStyle = 'lightslategray';
child5.isHitTestVisible = false;
child5.dock = DockStyle.Fill;

// Add the items to the diagram.
flow.addLabel(child1);
flow.addLabel(child2);
flow.addLabel(child3);
flow.addLabel(child4);
flow.addLabel(child5);
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



As you can see, the labels have been placed automatically in the owner node.

Labels may also be owned by a link. The **anchorPositionOnLink** property allows defining the position of the label near the link. It is a value between 0 and 1. If it is 0, then the label is placed near the origin node of the link. If it is 1, then the label is placed near the destination node of the link. If it is for instance 0.5, then the label is placed near the middle of the link.

```
node3 = new Node(flow, 60, 400, 40, 40, "");
flow.addNode(node3);

node4 = new Node(flow, 660, 400, 40, 40, "");
flow.addNode(node4);

link = new Link(flow, node3, node4, "", -1, -1);
flow.addLink(link);

labelOwnedByLink1 = new Label(flow, 100, 380, 140, 20, "AnchorPositionOnLink 0",
link);
labelOwnedByLink1.anchorPositionOnLink = 0;
flow.addLabel(labelOwnedByLink1);

labelOwnedByLink2 = new Label(flow, 310, 380, 140, 20, "AnchorPositionOnLink 0.5",
link);
labelOwnedByLink2.anchorPositionOnLink = 0.5;
flow.addLabel(labelOwnedByLink2);

labelOwnedByLink3 = new Label(flow, 520, 380, 140, 20, "AnchorPositionOnLink 1",
link);
labelOwnedByLink3.anchorPositionOnLink = 1;
flow.addLabel(labelOwnedByLink3);
```

5.9 Selection of items

5.9.1 Programmatic selection

To manage the selection a node or a link, you have to use the **isSelected** property. For instance, to select a node:

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
node.isSelected = true;
```

5.9.2 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 **mouseAction** 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.

5.9.3 selectedItems property

The Flow **selectedItems** property allows getting the array of selected items. For instance:

```
// Make each selected nodes red
for (let item of flow.selectedItems) {
  if (item instanceof Node) {
    node.fillStyle = 'red';
    node.refresh();
  }
}
```

5.9.4 selectionChange event

A **selectionChange** 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.

5.9.5 Hit testing

You can also know what object is under the mouse with the **hittedItem** property that returns the reference of the item under the mouse. If several objects are under the mouse, the returned object is the one that is at the top of the Z-order list. AddFlow provide some methods allowing changing this order (**bringToFront**, **sendToBack**, ...)

If the **isHitTestVisible** property of an item is false, then this item cannot be hit tested.

5.10 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

- **items**. This property returns the array of all items of the diagram.
- **selectedItems**. This property returns the array of all selected items of the diagram.
- **links**. This property 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:

```
for (let item of flow.items) {
  if (item instanceof Node) {
    emphasizeItem(item, 'red', 2);
  }
}
```

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

```
for (let item of flow.selectedItems) {
  if (item instanceof Node) {
    let selnode = item;
    for (let link of selnode.links) {
      if (link.org === selnode) {
        let node = link.dst;
        emphasizeItem(node, 'red', 3);
      }
    }
  }
}
```

The emphasizeItem function just changes the color and width of the item border:

```
function emphasizeItem(item, strokeStyle, lineWidth) {
  item.strokeStyle = strokeStyle;
  item.lineWidth = lineWidth;
  item.refresh();
}
```

5.11 Zooming

5.11.1 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.

The **zoomCenter** method allows applying a specified zoom factor and arranging so that a specified point is placed at the center of the viewport

TIP: How to autofit the diagram; i.e. how to adjust the zoom to its maximum while still keeping all the items in view?

You can implement this feature using the **zoomRectangle** method and the **xExtent and yExtent** properties.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
flow.zoomRectangle(0, 0, xExtent, flow.yExtent));
```

5.11.2 Interactive zoom

Notice that you may also zoom the diagram interactively with the mouse if the **mouseAction** 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

6) Avanced topics

6.1 Undo/Redo

6.1.1 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.

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.

6.1.5 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 = 'node properties';
```

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
  this.group = -1;

  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.selectedItems;
  if (selectedItems.length > 0) {
    if (selectedItems[0] instanceof Node) {
      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).

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

6.2 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, labelModel** and **linkModel** properties are saved.

6.3 Performance tuning

6.3.1 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();
      }
}
```

6.3.2 Quadtree structure

Better speed performance is provided by using a quadtree structure and it is the case by default. Otherwise. You may use the **isQuadtree** property 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 isQuadTree property is true or if it is false.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

```
flow.isQuadtree = true; // By default, it is true so this call is not needed here
```

6.4 Customizing the user interface

6.4.1 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.
canMultiLink	Determines whether you can create several links between two nodes.
canMoveNode	Determines whether interactive dragging of nodes is allowed or not.
canMultiSelect	Determines whether multiselection of nodes is allowed or not.
canReflexLink	Determines whether interactive creation of reflexive links is allowed or not.
canSizeNode	Determines whether interaction resizing of nodes is allowed or not.
canShowJumps	Determines whether jumps are displayed at the intersection of links.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

canShowContextHandle	Determines whether context handles are displayed for selected items
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.
canStretchLink	Determines whether interactive stretching of links is allowed or not.

6.4.2 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

6.4.3 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

6.4.4 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.

6.4.5 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

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

contextHandleStrokeStyle	Returns/sets the drawing color of the context handles

6.4.6 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

6.4.7 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

7) 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 provide a set of 3 graph layout algorithms:

- *Hierarchic layout*
- *Force directed (Symmetric) 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

- 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?

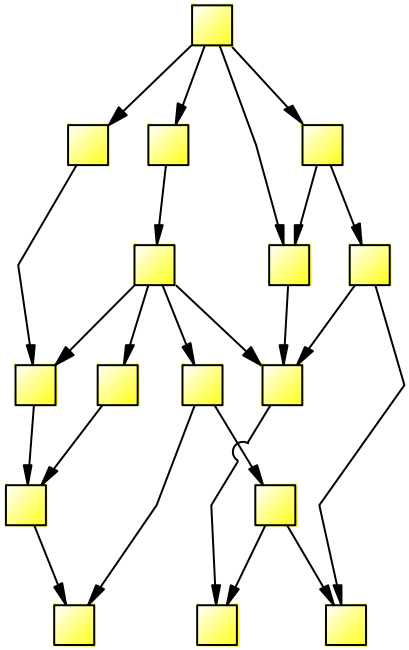
You have to use the **isExcludedFromLayout** property. Only nodes and links whose isExcludedFromLayout property is false are involved in a layout. This will allow you to make the layout algorithm to ignore some nodes or links.

7.1.1 Hierarchic layout

7.1.1.1 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved



- Hierarchic layout -

7.1.1.2 Code example

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

```
flow.doHierarchicLayout(
       50, // Sets the distance between adjacent levels
       50, // Sets the distance between adjacent nodes
       Orientation.North, // The orientation of the graph
       { width: 5, height: 5 }, // x and y margins
       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.

7.1.1.3 Limitation

It works with any graph, connected or not.

7.1.1.4 Side Effect

After the layout execution:

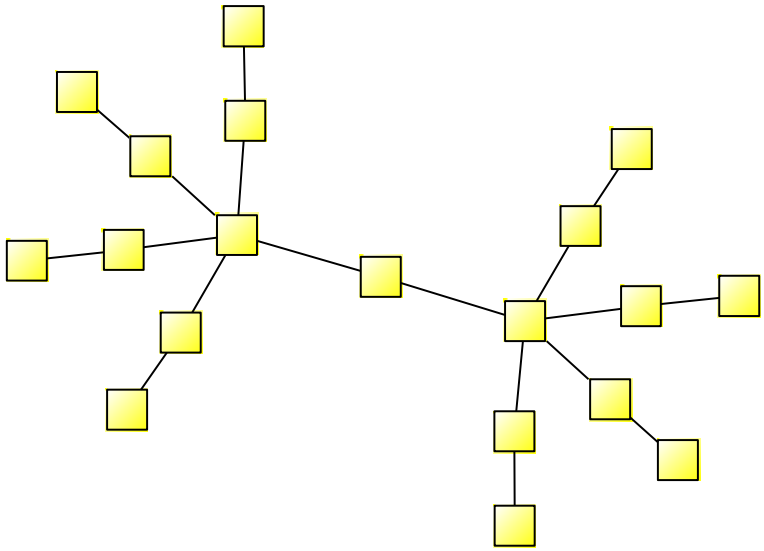
- the line style of the links is **polyline**
- the **isOrgPointAdjustable** property of the destination node of a link is set to false.
- the **isDstPointAdjustable** property of the origin node of a link is set to false.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

7.1.2 Force directed (symmetric) layout

7.1.2.1 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 -

7.1.2.2 Code example

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

```
flow.doForceDirectedLayout(
       40,      // Sets the distance between nodes
       { width: 5, height: 5 }, // x and y margins
       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
       false); // No animation provided
```

7.1.2.3 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.

7.1.2.4 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.

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

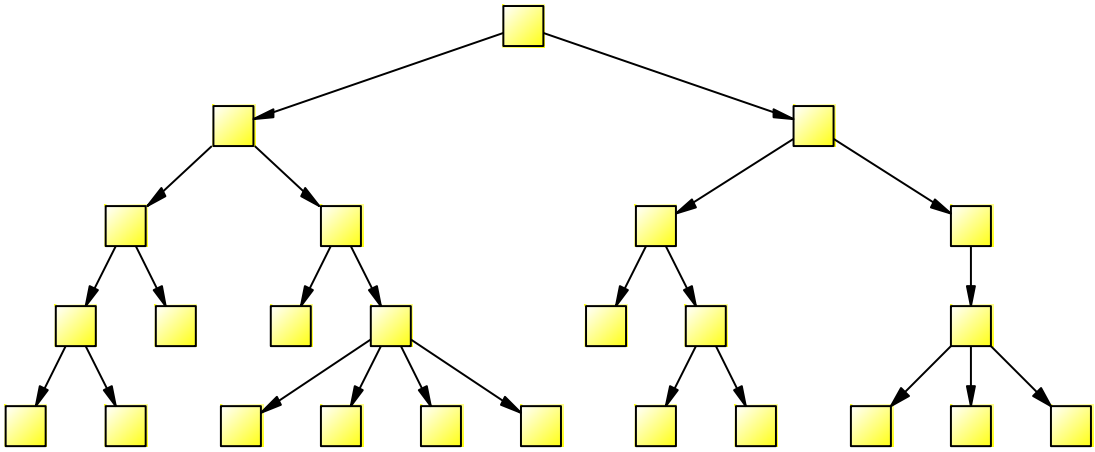
- the **isDstPointAdjustable** property of the origin node of a link is set to false.

7.1.3 Tree layout

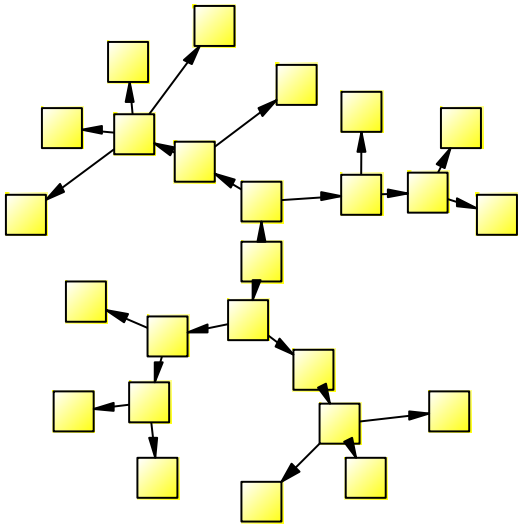
7.1.3.1 Purpose

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' -

Copyright (c) 2024 Lassalle Technologies. All Rights Reserved

7.1.3.2 Code example

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

```
flow.doTreeLayout(
       50, // Sets the distance between adjacent levels
       50, // Sets the distance between adjacent nodes
       TreeDrawingStyle.Layered, // The drawing style (layered or radial)
       Orientation.North, // The orientation of the graph
       { width: 5, height: 5 }); // x and y margins
```

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

7.1.3.3 Limitation

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

7.1.3.4 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.